

Étude et mise en œuvre d'une plateforme de développement rapide au travers de 2 projets

Frédéric Dauba

▶ **To cite this version:**

Frédéric Dauba. Étude et mise en œuvre d'une plateforme de développement rapide au travers de 2 projets. Génie logiciel [cs.SE]. 2014. dumas-01342922

HAL Id: dumas-01342922

https://dumas.ccsd.cnrs.fr/dumas-01342922

Submitted on 7 Jul 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

le cnam

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE REGIONAL ASSOCIE DE PAU

MEMOIRE

présenté en vue d'obtenir

le DIPLOME D'INGENIEUR CNAM

SPECIALITE : Informatique

OPTION : IRSM

par

Frédéric DAUBA

Etude et mise en œuvre d'une plateforme de développement rapide au travers de 2 projets

Soutenu le 1er juillet 2014

JURY

PRESIDENT : **M. Pierre Paradinas**, *Professeur, Conservatoire National des Arts et Métiers*

MEMBRES : **M. Mohamed Mosbah**, *Professeur, Institut Polytechnique de Bordeaux*

M. Alain Teste, *Maitre de Conférences, UFR Sciences et techniques de Pau*

M. Richard Castanet, *Professeur émérite, Institut Polytechnique de Bordeaux*

M. Laurent Fallot, *Maître de Conférences, Institut Polytechnique de Bordeaux*

M. Frédéric SOURBES, *Responsable, AIS Informatique*

M. Eric RIBBENS, *Responsable Système d'Information, CERFRANCE Landes*

Remerciements

Je remercie mon responsable Frédéric SOURBES pour m'avoir fait confiance pour mener à bien en toute autonomie les projets de développements informatiques engagés par AIS.

Je remercie M. Alain TESTE pour sa disponibilité et ses conseils avisés lors de la rédaction de ce mémoire.

Je remercie mon épouse Sandrine pour son soutien quotidien pendant la durée de rédaction de ce mémoire.

Liste des abréviations

ADO.NET : ActiveX Data Object pour le framework .NET
ADSL : Asymetric Digital Subscriber Line
AIS : Agri Informatique Services
ASP.NET : Active Serve Pages pour le framework .NET
BCL : Base Class Library
CER40 : CERFRANCE Landes
CIE : Carte d'Identité Sanitaire
CIS : Carte d'identité Environnementale
CLR : Common Language Runtime
DHCP : Dynamic Host Control Protocol
DLL : Dynamic Link Library
DMZ : DeMilitarized zone
DNS : Domain Name System
EDI : Environnement de Développement Intégré
EXE : Exécutable
FCL : Foundation Class Library
FTP : File Transfert Protocol
GDSAA : Groupement de Défense Sanitaire Aquacole Aquitain
GED : Gestion Electronique de Documents
GUID : Global Unique IDentifier
IG : Innovation Game
L4G : langage de quatrième génération
LDD : Langage de Définition de données
MD5 : Message Digest 5
MMC : Microsoft Management Console
MSIL : MicroSoft Intermediate Language
PDR : Plateforme de Développement Rapide
PMBOK : Project Management Body of Knowledge
PME : Petites et Moyennes Entreprises
PMI : Project Management Institue
PO : Product Owner
RAID : Redundant Array of Independent/Inexpensive Disks
SGBD : Système de Gestion de Base de données
SM : Scrum Master
SMO : SQL Management Objects
SQL : Structured Query Language
SVN : SubVersioN
TFS : Team Foundation System
UP : Unified Process
WCF : Windows Communication Foundation
WSUS : Windows Server Update Services
XML : eXtensible Markup Language
XP : eXtreme Programming

Table des matières

Remerciements **2**

Liste des abréviations **3**

Table des matières **4**

1. Présentation de l'entreprise **6**

1.1. Agri Informatique Services 7

1.2. Infrastructure technique 8

2. Origine et cadre du projet de plateforme de développement rapide **9**

2.1. Historique des développements de logiciels 9

2.1.1. Le mode caractère *9*

2.1.2. Les formulaires graphiques *9*

2.2. Le projet POISSONS V2 10

2.3. Le projet *GESTIONCP* 11

2.4. Constats et solution proposée 11

2.5. Planification 12

2.6. Conditions de réalisation 13

2.7. Démarche agile 14

2.7.1. Choix de la méthode *15*

2.7.2. Scrum *17*

2.8. Outils utilisés 23

2.8.1. La Plateforme .Net *23*

2.8.1. Environnement de développement intégré *25*

2.8.2. Système de Gestion de Base de données *26*

2.8.3. Gestion de code Source *30*

2.8.4. Pilotage de projet *32*

3. Le projet Outils : la plateforme de développement rapide **33**

3.1. Présentation 33

3.2. Gestion des risques 34

3.3. Architecture technique cible 34

3.4. Analyse, conception 35

3.4.1. De la vision aux stories *35*

3.4.2. Version initiale *38*

3.4.3. Maquettage et ergonomie *38*

3.5. Mise en œuvre 40

3.5.1. Mêlée 1 ..... *40*

3.5.2. Mêlée 2 ..... *59*

3.5.3. Mélée 3 ..... *68*

3.5.4. Mélée 4 ..... *82*

3.6. Bilan 86

4. Le projet de GESTIONCP **87**

4.1. Contexte et périmètre 87

4.2. Etude préalable 87

4.2.1. 1 – Recommandations et philosophie du logiciel *87*

4.2.2. Analyse 88
4.3. Mise en œuvre 90
4.3.1. Architecture de la solution 90
4.3.2. Conception de la base de données 91
4.3.3. Construction des écrans de paramétrage 97
5. Le projet POISSONS V2 101
5.1. Amélioration des outils 101
5.1.1. Editeur de table de la base de données 101
5.1.2. Editeur de Listes 105
5.1.3. Editeur d'écran 107
5.1.4. Editeur de code pour les scripts 111
5.2. Exemples d'écrans de l'application 114
6. Conclusion 116
Annexes 118
Bibliographie 128
Liste des figures 129
Liste des tableaux 132

1. Présentation de l'entreprise

CERFRANCE est le 1er réseau associatif de conseil et d'expertise comptable en France au service de 320 000 clients de nombreux secteurs d'activité (agriculture, artisanat, commerce, services, professions libérales et PME). Ces chefs d'entreprises bénéficient des compétences pluridisciplinaires de plus de 11 200 collaborateurs parmi lesquels des conseillers, des juristes, des consultants et des experts comptables. Avec ses 700 agences implantées sur l'ensemble du territoire, CERFRANCE offre une forte proximité géographique et culturelle aux acteurs économiques locaux.

Le réseau a un fonctionnement de type mutualiste comme l'illustre la Figure 1. Il est administré par des clients élus au sein des entreprises associatives territoriales.

Le Conseil d'Administration est constitué de 23 présidents et de 2 directeurs généraux de CERFRANCE territoriaux.

Le bureau du Conseil National est constitué de 7 présidents et de 2 directeurs généraux de CERFRANCE territoriaux.

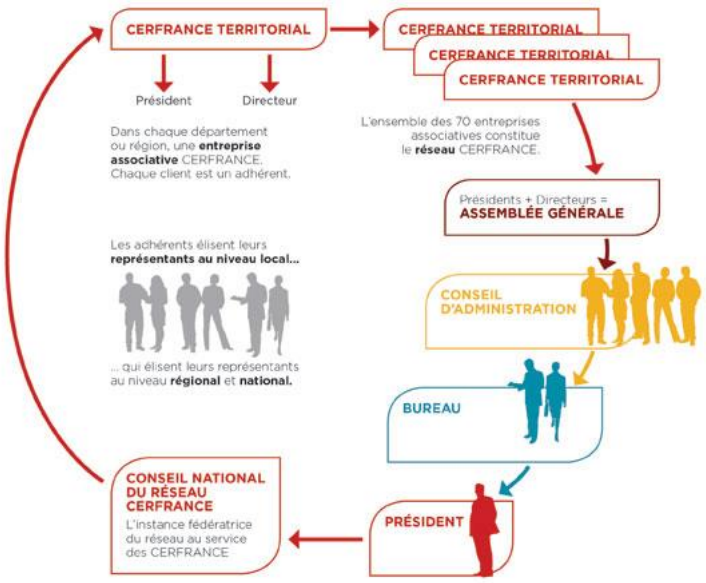


Figure 1 - Gouvernance du CERFRANCE

1.1. Agri Informatique Services

AIS (Agri Informatique Services) est une société de services informatiques rattachée au groupe CER40 (CERFRANCE Landes). Sa mission principale est de répondre aux besoins informatiques des adhérents du CER40. Elle se compose de 3 services :

- **Service commercial** : Il est chargé de la vente du matériel informatique et des services associés aux adhérents du CER40. Il exerce également des actions prospectives vers les entreprises locales afin d'élargir le fichier client. La vente de licence d'utilisation des logiciels de comptabilité, de facturation et de paye constitue une part importante du chiffre d'affaire.
- **Support technique** : Il réalise les prestations techniques en atelier ou sur site. On peut citer par exemple les prestations de maintenance informatique de base (remise en état de système défaillant ou infecté par des virus) ou des prestations de mise en route de nouveau matériel (poste utilisateur, serveur ou infrastructure réseau).
- **Support Logiciel** : Il assure le déploiement, le paramétrage et l'assistance à l'utilisation de 1er niveau des logiciels de facturation et de paye édités par la société ISAGRI et distribués par AIS. L'installation du logiciel de comptabilité est également effectuée pour le compte du CER40. Son activité comprend également la réalisation de logiciels de gestion métiers spécifiques.

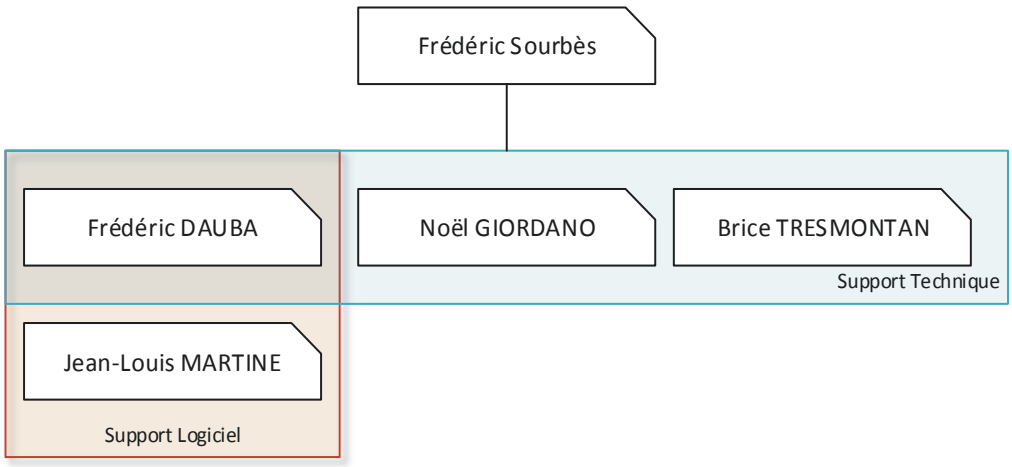


Figure 2 - Organigramme AIS Informatique

C'est au sein de cette entreprise que je travaille depuis 2 ans. Initialement membre du service de support technique, je fais maintenant partie également du service de support logiciel. J'ai en charge la maintenance logicielle des produits de gestion distribués par AIS, la gestion de l'infrastructure informatique interne et l'administration des serveurs de nos clients.

1.2. Infrastructure technique

Elle est composée de plusieurs réseaux isolés par des pare-feux.

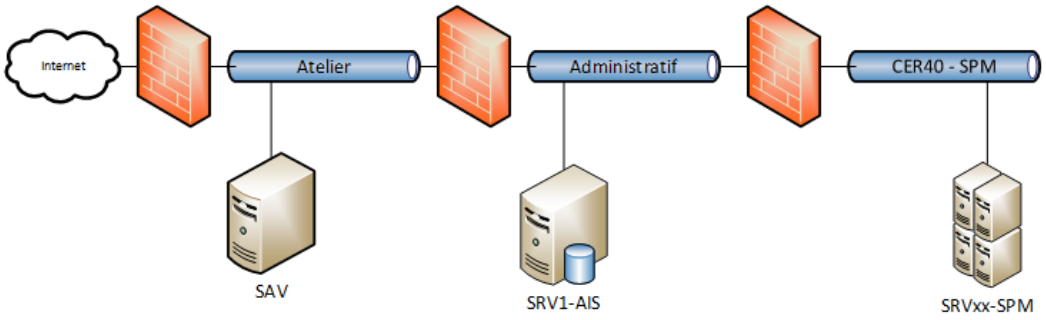


Figure 3 - Architecture technique de AIS

Le réseau « Atelier » dispose d'une connexion internet ADSL de 8Mb/s. Un serveur Windows 2012 Standard assure le partage de fichiers et de programmes utilitaires pour la préparation des machines neuves, un service FTP (File Transfert Protocole) pour l'échange de fichier volumineux ainsi que WSUS (Windows Server Update Services) pour limiter l'usage de la bande passante. L'adressage réseau est identique à celui que l'on utilise à domicile pour faciliter les diagnostics sur les machines des clients. Un réseau Wifi permet la connexion des ordinateurs portables sur ce réseau. L'atelier constitue une zone de faible sécurité appelée DMZ.

Le réseau « Administratif » contient les postes administratifs et ceux du service de support logiciel. Un serveur Windows 2003 R2 gère le domaine avec les services usuels (DNS[1], DHCP[2], Active Directory, Terminal Serveur, partage de fichiers). Il contient également un Server SQL 2008 R2 Express utilisé pour les applications de facturation, de

[1] DNS : Domain Name System

[2] DHCP : Dynamic Host Control Protocol

comptabilité ainsi que les outils internes de gestion d'incidents. La sécurité et l'intégrité des données sont assurées par un système de disque de type RAID 5. Des sauvegardes automatisées sur disques externes permettent d'avoir une copie hors des locaux.

Une liaison directe de 1 Gb/s vers le réseau du CER40 permet l'accès à l'ensemble des ressources du groupe mais seul le serveur de messagerie Zimbra est actuellement utilisé.

2. Origine et cadre du projet de plateforme de développement rapide

2.1. Historique des développements de logiciels

Nous allons aborder l'historique des développements de logiciel informatique réalisés par AIS afin de mieux cerner les besoins d'évolutions des outils de développement. En effet, le démarrage des projets **POISSONS V2** et **GESTIONCP** (voir chapitre 2.2 et 2.3) implique une mise à niveau rapide des outils et des techniques de développement.

2.1.1. Le mode caractère

L'environnement de développement choisi pour les premiers développements de logiciel est distribué par la société MEMSOFT. Il s'agit d'un progiciel de gestion 16 bits utilisant le langage BASIC. Une version baptisée MOD1 2000 apporte le support des systèmes d'exploitations 32 bits de l'époque (Windows 2000 et 2003 Server).

Parmi les programmes réalisés avec cette technologie et qui sont toujours en exploitation on peut citer :

- **la gestion commerciale** : de nombreuses versions spécifiques ont vu le jour avec les années. La fin du support est programmée au 1er janvier 2014 par AIS.
- **Poissons** : un logiciel de traçabilité utilisée par le Groupement de Défense Sanitaire Aquacole Aquitain (GDSAA) dont le début de la réécriture fait l'objet du chapitre 5 de cette étude.

2.1.2. Les formulaires graphiques

La programmation est réalisée avec un langage de quatrième génération (L4G) baptisé Oxygène. La société MEMSOFT propose sa solution gratuite pour la gestion et la

comptabilité. Oxygène permet la réalisation de modules complémentaires. Un éditeur de formulaires, de tables et d'états est mis à disposition. Le langage est traduit en C++ puis compilé afin d'obtenir une DLL qui sera chargée par l'application. Le code écrit sur la version 5 n'est malheureusement plus compatible avec la version 8 de la solution. Toute mise à jour de l'aspect visuel est compromise.

Le logiciel **Trace Landes** est développé à partir de 2002. Il a connu environ 200 utilisateurs avant sa fin de commercialisation en 2006.

Ensuite, c'est au tour du logiciel de facturation **WinFactu** de voir le jour. Il est actuellement toujours commercialisé pour répondre aux besoins de petites entreprises qui ne souhaitent pas basculer sur un outil plus complet.

2.2. Le projet POISSONS V2

Le Groupe de Défense Aquacole Sanitaire d'Aquitaine regroupe pêcheurs et pisciculteurs pour la protection du milieu aquatique et l'amélioration de l'état sanitaire des poissons, dans les pisciculture comme dans les étangs et rivières d'Aquitaine. Il effectue des contrôles sanitaires et des analyses sur l'eau et les poissons, en collaboration avec le laboratoire départemental, ainsi que des études sur l'environnement et la pathologie des poissons.

Avant mon arrivée dans l'entreprise, des démarches ont été entreprises par le GDSAA afin d'étudier la refonte de son système d'information. Eric Labarthe, ancien responsable d'AIS, a établi un cahier des charges avec l'appui de Guy ASSELIN le développeur du logiciel **POISSONS** qui est aujourd'hui à la retraite.

Afin d'anticiper les changements de législation, le client souhaite pouvoir modifier à loisir les différents écrans, états et statistiques que l'application mettra à disposition. Il faut des écrans utilisables à la souris, des listes déroulantes, des chemins plus courts pour accéder aux données que ne le permettent des applications en mode caractères. Oxygène est également trop limité pour offrir le niveau de personnalisation souhaité.

Seule la partie correspondant à l'ancien logiciel a fait l'objet d'une étude préliminaire. De nouvelles données doivent être collectées mais aucune analyse de conception n'a eu lieu. De plus certains modules évoqués sont à l'état d'idée plus ou moins floue. Le projet

bien que signé est en attente de financement. C'est sans compter le changement de direction au GDSAA qui ajoute encore un peu plus d'incertitude.

Dans ce projet, j'occuperai à la fois le poste de chef de projet et de développeur. Plusieurs stagiaires interviendront durant les phases de réalisations.

2.3. Le projet *GESTIONCP*

Le CER40 souhaite se doter d'un logiciel de gestion de dépense de jours de congés. Le DSI (Directeur du Service Informatique), M. RIBBENS, souhaite intégrer cette application dans le futur intranet. C'est pourquoi l'interface principale est souhaitée sous forme web pour intégrer l'intranet en cours de réalisation. Dans un premier temps, le CER40 recherchait un progiciel pour satisfaire leurs besoins. Suite à cette approche infructueuse, AIS a été sollicité pour effectuer un développement spécifique. Aucun cahier des charges n'a été fourni mais les documents de comparaison des fonctionnalités utilisés dans l'évaluation des progiciels permettent de définir les points essentiels.

Là encore, on m'a proposé de mener à bien ce projet en collaboration avec le directeur du service informatique, M. RIBBENS, du CER40. Ma contribution dans ce projet réside dans le choix d'une solution technique et de sa mise en œuvre. Ce projet est en grande partie hors du périmètre de ce mémoire mais il va permettre de valider, avec un cas concret, l'utilisation des composants de base de la plateforme de développement rapide.

2.4. Constats et solution proposée

Il n'y a ce jour aucune méthodologie de conduite de projet particulière. Les développements sont réalisés suivant la méthode *LaRACHE*[3] et les codes sources sont disséminés sur différents postes informatiques vieillissants qui sont les seuls encore aptes à compiler les programmes.

Une plateforme de développement rapide est un ensemble d'outils destiné à construire une application informatique en un minimum de temps. Pour arriver à ce

[3] http://byatoo.com/la-rache/

résultat, une grande partie des fonctionnalités est déjà opérationnelle. On se contente d'assembler des éléments prêts à l'emploi. Le terme de plateforme est souvent remplacé par celui d'environnement ou méthode mais on désigne ce que l'on résume par le concept RAD (Rapid Application Developpement). Il existe un grand nombre d'outils de ce type. On peut citer les plus connus comme Delphi, Visual Basic, C++ Builder, Eclipse, Lazarus, Leonardi, Power Builder ou encore WinDev. La différence d'un outil à l'autre provient de la couverture fonctionnelle prise en charge. Certains s'étendent de la conception de la base de données jusqu'à la réalisation des états en incluant la modélisation des entités métiers et l'écriture de code dans un langage de programmation plus ou moins spécifique. D'autres se contentent d'un paramétrage exhaustif sans possibilité de personnalisation du résultat obtenu. Le point commun entre tous ces outils est qu'ils sont destinés à un public de développeur. Seul Visual Studio Light Switch ouvre la piste de la personnalisation par l'utilisateur final. Ce produit étant basé sur de la génération de code à partir de l'environnement de développement intégré Visual Studio, il ne permet pas de modifier le schéma de la base de données ou d'ajouter de nouveaux écrans après la phase de génération de l'application.

Dans l'attente du démarrage du projet **POISSONS V2**, j'ai proposé la création d'une bibliothèque de développement rapide. Elle doit permettre le changement technologique pour répondre à la problématique de la personnalisation par l'utilisateur final des applications.

2.5. Planification

Le projet Gestion de congés offre l'opportunité de mettre en œuvre une première version de ces outils. Le temps économisé sur la partie paramétrage offrira une plus grande marge de manœuvre pour la réalisation de la partie web supposée couteuse en terme de temps.

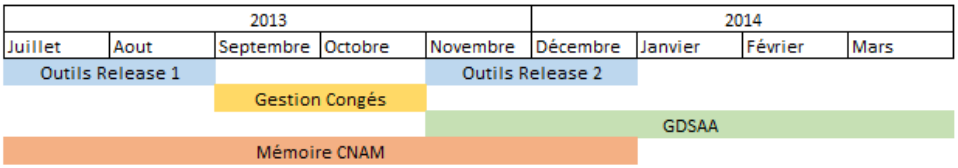


Figure 4 - Planning prévisionnel du projet CNAM

Les 2 premiers mois permettront de constituer une première version des outils sans les fonctionnalités de paramétrage graphique. En effet le périmètre du projet gestion de congés est clairement défini par les recommandations du CER40 et ne nécessite pas de modification par l'utilisateur final. Une fois la gestion de congés en production, le projet GDSAA devrait commencer par le recueil d'informations plus précises. Il sera alors aisé de produire une maquette fonctionnelle pour illustrer les propositions sur la disposition des champs dans les écrans. Au fur et à mesure de la détection des besoins, il faudra ajouter de nouvelles fonctionnalités.

On peut résumer les objectifs de ce mémoire par les points suivants :

- simplification du développement des futures applications avec la création d'une plateforme de développement rapide appelée également « boîte à outils »
- développement d'un logiciel de paramétrage du système de gestion de dépense de jours de congés
- début de la réécriture d'une application de traçabilité du GDSAA avec une **forte personnalisation possible par le client final**.

2.6. Conditions de réalisation

En plus de mon rôle de développeur, j'assurerai celui de chef de projet[4] pour l'étude et la réalisation de la boîte à outils ainsi que lors des 2 projets où elle sera mise en œuvre. L'équipe sera complétée de mon collègue développeur et de stagiaires de niveau bac + 3. Le suivi du projet de gestion de congés sera traité en collaboration avec le directeur informatique du CER40.

Sous réserve d'intervention technique sur des serveurs ou des déploiements de mise à jour sur les logiciels de paye, comptabilité ou facturation, mon temps sera intégralement consacré à ces différents projets.

[4] Le terme est à prendre au sens « responsable de projet » puisque selon la méthode qui sera appliquée, le rôle de chef de projet n'existe pas. Plus précisément j'assurerai les rôles de Product Owner, Scrum Master et développeur.

2.7. Démarche agile

Le rôle de chef de projet est de mener à bien la réalisation d'un projet. De nombreux ouvrages, tels que le PMBOK[5] (Project Management Body of Knowledge) recensent les connaissances et les techniques de gestion de projets. C'est au chef de projet d'estimer leur pertinence selon l'environnement spécifique de son projet. Un projet peut échouer simplement du fait de l'inadéquation de la méthode au contexte. Par exemple, sur un projet de petite taille comme GESTIONCP ou POISSONS V2, une méthode impliquant un formalisme très élevé est une source d'échec. Cela est dû au délai court et au fait que les participants ne maitrisent pas la méthodologie. Le changement de méthodologie devrait dans ce cas être initié comme un projet à part entière.

Les méthodes agiles sont apparues dans les années 2000 suite à la publication d'un manifeste[6]. Elles ont pour objectif d'améliorer les pratiques pour répondre aux besoins spécifiques nécessaires au succès de projets informatiques. Dans le cadre de ce mémoire, compte tenu de l'incertitude élevée au niveau des spécifications des projets et de la disponibilité des clients, j'ai opté pour l'application d'une méthode de développement agile. Le principe de ces méthodes repose sur la réalisation de versions livrables de l'application à la fin de chaque itération. Les tests réalisés à chaque itération évitent la découverte tardive de bugs qui peuvent mettre en péril un projet entier. La Figure 5 présente le cycle de vie d'une itération utilisé par la méthode Scrum.

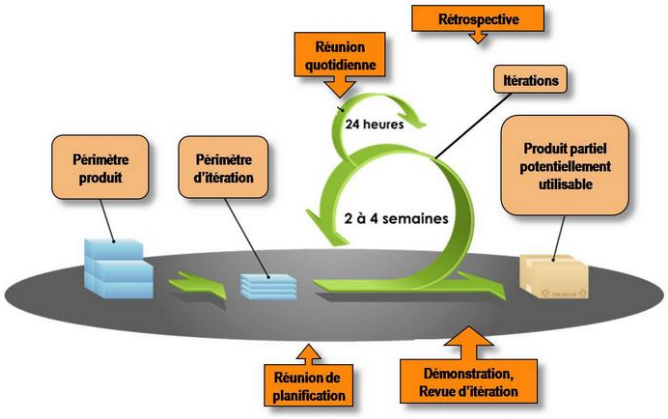


Figure 5 - Cycle de vie simplifié de Scrum

[5] C'est un ouvrage de référence en gestion de projet, édité par le PMI (Project Management Institute)

[6] Les 12 principes du manifeste Agile sont présentés dans l'Annexe 2

2.7.1. Choix de la méthode

Dans [1], on trouve un comparatif des principales méthodes agiles parmi les plus courantes en fonction du nombre d'itérations et de leur degré de formalisme. Un degré de formalisme élevé comprendra plus d'étapes formelles et de documents à produire. Le cycle de vie est réparti entre de nombreuses itérations jusqu'à une séquence unique en cascade.

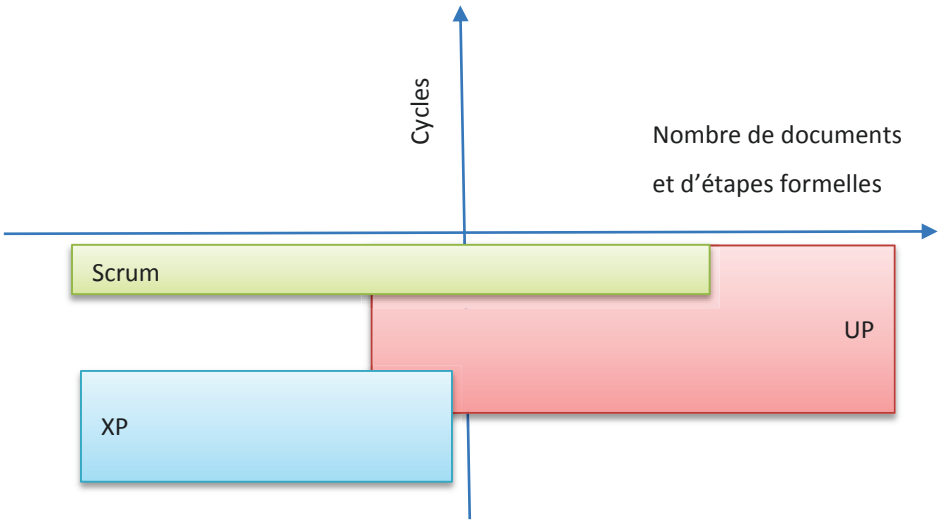


Figure 6 - Classification des méthodes agiles selon leur degré de formalisme et leurs itérations

La durée recommandée des itérations pour XP (eXtreme Programming) est de une à deux semaines. UP (Unified Process) se situe entre deux et six semaines et enfin Scrum préconise une durée de 30 jours. La tendance pour les mêlées (itérations de Scrum également appelées sprints) est plutôt de 2 à 3 semaines selon [2].

Au niveau de la quantité de documents à produire, Scrum laisse le choix à l'équipe de développement. UP est plus exigeant même si en réalité de nombreux documents sont optionnels. XP est minimaliste et recommande de ne spécifier que les fonctionnalités demandées.

Tableau 1 - Forces et faiblesses des principales méthodes agiles

Méthode	Résumé	Points forts	Points faibles
XP	Simplicité	• solides pratiques techniques • favorise la qualité avec le client sur site et les tests omniprésents. • Fréquent feedback grâce à la brièveté des itérations. • La plus connue et sans doute la plus répandue (au Etats-Unis)	• méthode radicale, les pratiques sont poussées à l'extrême. • La programmation en binôme n'est pas toujours bien ressentie par les développeurs. • documentation projet très réduite, donc limites de la méthode dans le cas de projets évolutifs.
Scrum	Valeur ajoutée pour le client	• les priorités sont gérées en fonction de la valeur ajoutée. • bien adaptée au développement d'un progiciel au sein d'équipe produit, grâce à la présence d'un product owner. • favorise la communication et la collaboration (scrum quotidien) • propose un programme de certification	• limite les changements avec des itérations de trente jours et un contenu figé durant le sprint. • limitée à la discipline de gestion de projet • ne propose aucune pratique technique.
UP	Gestion des risques	• très bien documentée • est devenue un standard dans de nombreuses organisations • un bon intermédiaire pour le passage d'une approche classique à une méthode itérative.	• méthode la moins agile de toutes, prescriptive, qui peut être lourde à mettre en œuvre. • beaucoup de livrables à produire • est parfois associée à un cycle en cascade mais c'est alors une mauvaise interprétation non suggérée par la méthode elle-même

Dans la pratique, il ne faut pas négliger la possibilité de mélanger les différentes approches afin d'obtenir un mode de fonctionnement qui correspond bien à l'environnement du projet.

Pour la conduite des projets **Outils**, **GESTIONCP** et **POISSONS V2** j'ai décidé de mettre en place Scrum qui semble être un compromis intéressant entre la trop grande documentation demandée par UP (incompatible avec les délais) et le développement en binôme (nombre de développeurs non fixe dans le temps).

2.7.2. Scrum

La spécificité de Scrum est l'augmentation rapide de la valeur ajoutée au fur et à mesure des livraisons des versions de l'application. C'est possible en donnant la priorité aux fonctionnalités ayant le plus de valeur ajoutée selon le client. Ainsi, une nouvelle idée qui émerge en plein milieu du projet peut être réalisée dès la prochaine itération si le client considère celle-ci comme plus importante que celles initialement planifiées.

2.7.2.1. Cycle

Un cycle agile est caractérisé par de courtes itérations qui ne se chevauchent pas et surtout qui ont des durées identiques. Ces blocs de temps appelés *timebox*, avec une équipe de développement fixe, ont toujours le même coût. Cela simplifie grandement le chiffrage. L'itération dans Scrum est appelée sprint ou mêlée. La production de logiciel est constituée de jalons majeurs et mineurs. Dans Scrum, le jalon mineur est la fin d'une mêlée. Il se traduit par la vérification des objectifs et surtout de la prise en compte des retours d'informations (*feedback*) pour adapter le contenu des mêlées suivantes. Le jalon majeur est constitué par la livraison d'une version (*release*) potentiellement utilisable du logiciel qui contient généralement une nouvelle fonctionnalité principale.

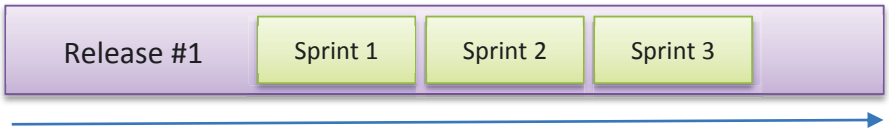


Figure 7 - Une release et ses sprints

Chaque sprint comprend les activités :

- **S**pécification fonctionnelle
- **A**rchitecture (conception)
- **C**odage
- **T**ests d'intégration et recette

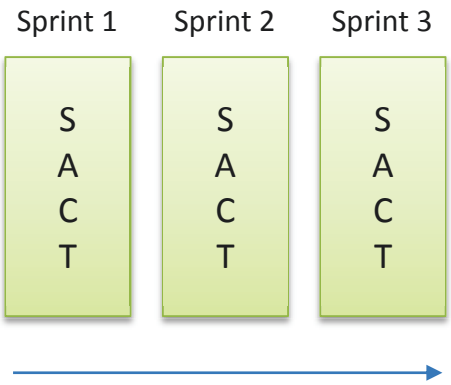


Figure 8 - Activités en parallèle des sprints

Le démarrage de la première release ne commence pas immédiatement par un sprint. Il faut effectuer une série de travaux préparatoires comme la constitution de l'équipe, définir la vision du produit, ou la planification des releases.

2.7.2.2. Les rôles

Scrum introduit deux nouveaux rôles par rapport aux méthodes traditionnelles. Celui de PO (Product Owner) et de SM (Scrum Master).

Le PO est la personne responsable du résultat produit par l'équipe de développement. Il n'a pas un rôle hiérarchique sur l'équipe mais par ses choix stratégiques et tactiques il incarne la vision du produit à long terme en concertation avec les autres acteurs. Il détermine donc dans quel ordre seront développées les parties du produit ainsi que les dates des différents jalons.

Le SM a un rôle de facilitateur. Il doit veiller à l'application de Scrum en assurant la tenue des réunions et favoriser la progression et l'auto-organisation des équipes. Il doit également éliminer le plus rapidement possible tous les obstacles qui pourraient nuire à la bonne réalisation des objectifs fixés pour les sprints.

L'équipe de développement est une entité primordiale dans Scrum. Elle doit en effet s'organiser elle-même et définir précisément les tâches nécessaires pour mener à terme un sprint. Cela inclut l'analyse, la conception, la réalisation et les tests.

2.7.2.3. Les artefacts

Les fonctionnalités émergent tout au long du cycle de développement du produit. Il n'y a pas de spécification détaillée en début de projet. La collaboration continue entre les membres de l'équipe et les retours d'expérience des utilisateurs produisent de nouvelles demandes qui sont collectées dans le backlog. Il n'existe pas de traduction officielle donc c'est le terme anglais qui est couramment utilisé. C'est une liste ordonnée par priorité composée par des stories qui possèdent des attributs et un cycle de vie propre.

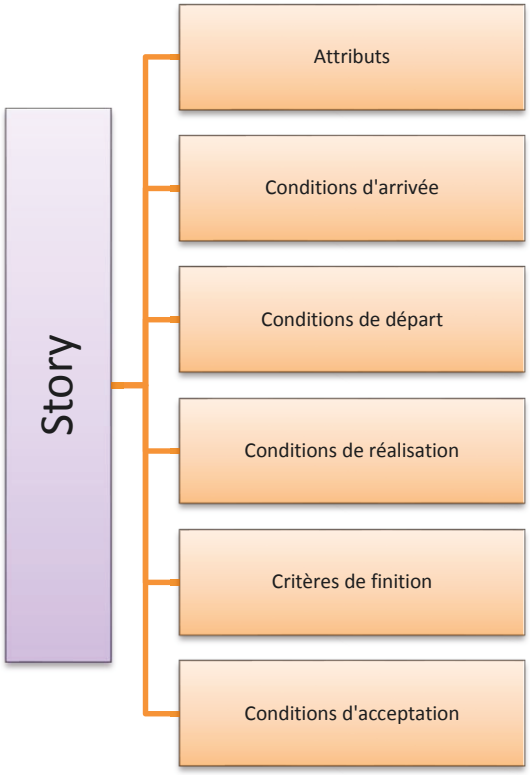


Figure 9 - Composition d'une story

Les attributs définissent en quelque sorte la carte d'identité de la story. Il n'y a pas vraiment d'élément obligatoire à part peut-être l'état actuel, le type et une description. On y intègre généralement le nom du créateur, sa date de création ainsi que l'historique des changements d'états.

Les conditions de départ comprennent une vérification des conditions de réalisations par l'ensemble de l'équipe. C'est-à-dire que les conditions d'acceptation doivent être comprises et que les critères de finition définissant la qualité attendue soient connus. Les

conditions d'arrivées sont satisfaites et la story terminée lorsque le PO juge que les autres conditions sont remplies.

Il existe 4 types de story comme le montre la figure suivante :

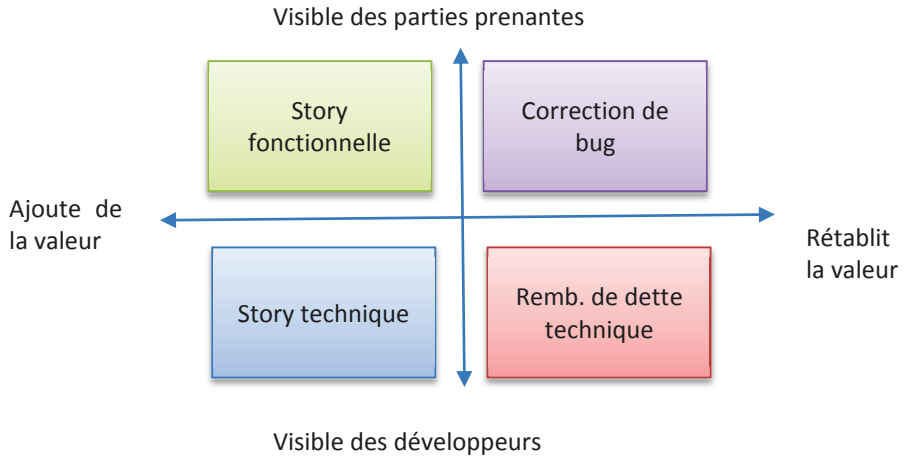


Figure 10 - Les quatre types de story

La story fonctionnelle est un élément à produire qui peut ne pas nécessiter de code. Dans [2], l'auteur donne l'exemple d'une vidéo de présentation à réaliser.

La correction de bug concerne une story finie. L'objectif est d'améliorer la valeur de la story associée qui n'a pas tenu toutes ses promesses malgré les conditions d'acceptations remplies. En effet un bug au sens informatique du terme sera détecté et corrigé plus tôt. Le backlog n'a pas vocation à enregistrer tous les bugs.

La story technique est un élément préalable à la réalisation d'autres stories fonctionnelles. Par exemple, on peut citer un système de gestion de la base de données. L'objectif est de réduire le risque pour les autres stories.

La story de remboursement de dette technique permet de matérialiser dans la planification les travaux nécessaires pour la maintenance du code. Une solution qui paraissait raisonnable avec une story peut entrainer des complications si des nouvelles stories introduites plus tard viennent utiliser une partie de l'existant qui n'a pas été conçu spécifiquement pour cette utilisation. Seul un développeur pourra s'apercevoir que le modèle objet n'est pas parfaitement adapté ou encore qu'une fonctionnalité est détournée de son objectif initial. La pratique montre qu'il vaut mieux reprendre le plus tôt possible ce

qui ne va pas plutôt que de laisser les choses empirer. C'est aussi une des forces des méthodes agiles de permettre de réagir rapidement.

La liste ordonnée des stories alimente les sprints suivants. Ainsi chaque story va passer successivement par les états suivants :



Figure 11 - Etapes d'une story

2.7.2.4. Les Bacs

Le backlog n'est pas une liste dont toutes les stories ont le même niveau de maturation. On divise le backlog en plusieurs bacs afin de trier et faire évoluer les stories.

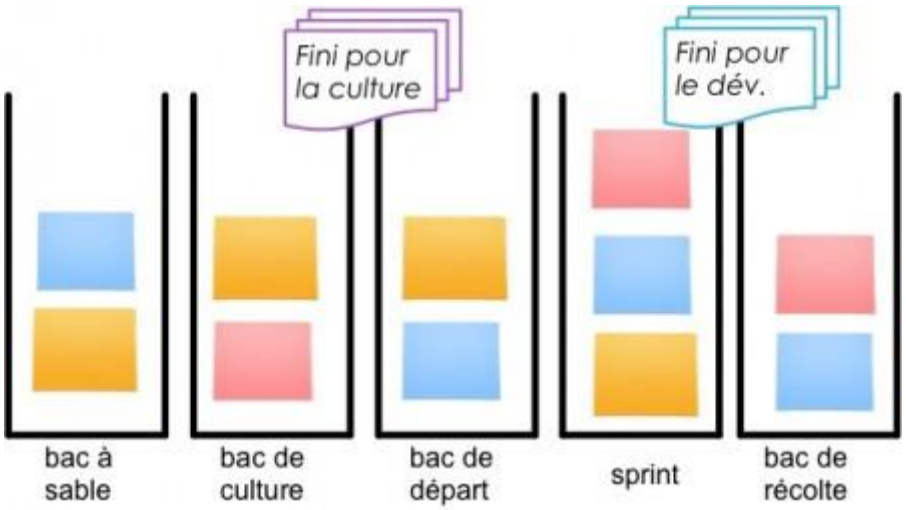


Figure 12 - Les bacs à story

Le bac à sable est destiné à recueillir les nouvelles idées. Le PO peut l'accepter s'il arrive à obtenir suffisamment d'informations de la part de l'équipe et des parties prenantes. Il la transforme ensuite en story pour le bac de culture. La notion de culture est une analogie avec le jardinage. Il faut prendre « soin » d'une story. L'opération consiste à l'étudier pour bien définir sa taille (éventuellement on la décompose en plusieurs stories) et définir ses conditions de réalisation, ses conditions d'acceptation et ses critères de finition. Lorsqu'une story est prête, le PO la place dans le bac de départ et en fonction des

priorités, les stories du bac de départ viendront alimenter les futurs sprints. C'est de cette manière que la planification s'effectue au fur et à mesure de l'avancée du projet.

Le bac de récolte contient simplement les stories terminées et constitue l'historique des fonctionnalités développées et intégrées au logiciel.

2.7.2.5. Planification

Les capacités de réaction au changement des méthodes agiles ne signifient pas qu'il n'y a pas de planification à réaliser. Avec Scrum, on effectue une planification à moyen terme grâce au plan de release. Il comprend la liste des stories des prochains sprints de la release.

On trouve dans [2] la définition des indicateurs qui permettent la gestion des prévisions. La **vélocité** est la mesure de la partie du backlog qui est réalisée par l'équipe pendant un sprint. La **capacité** de l'équipe est une prévision de ce que l'équipe est capable de faire pendant un sprint. Ces deux indicateurs sont exprimés par des quantités qui sont les sommes des valeurs attribuées aux stories. Ces valeurs, généralement choisies parmi la suite Fibbonacci, correspondent à la taille relative des stories estimées par l'équipe lors des réunions de planification.

Un **burndown chart** est une représentation graphique de ce qu'il reste à faire dans une période, actualisée aussi souvent que possible et permettant de montrer la tendance. On réalise des **burndown chart** de sprint actualisés tous les jours et des **burndown chart** de release mis à jour à chaque fin de sprint.

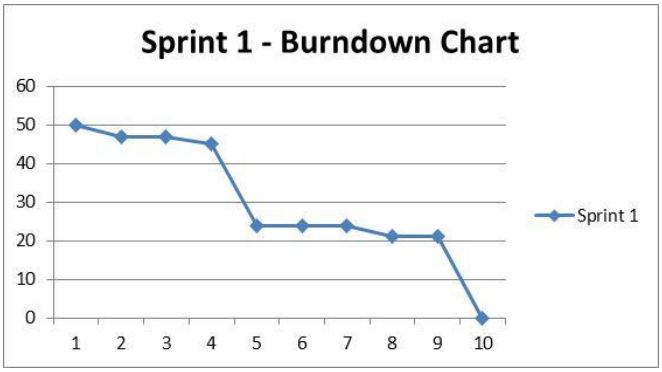


Figure 13 - Exemple de Burndown chart

Chaque état possible d'une story correspond une étape de planification comme le montre la Figure 14.

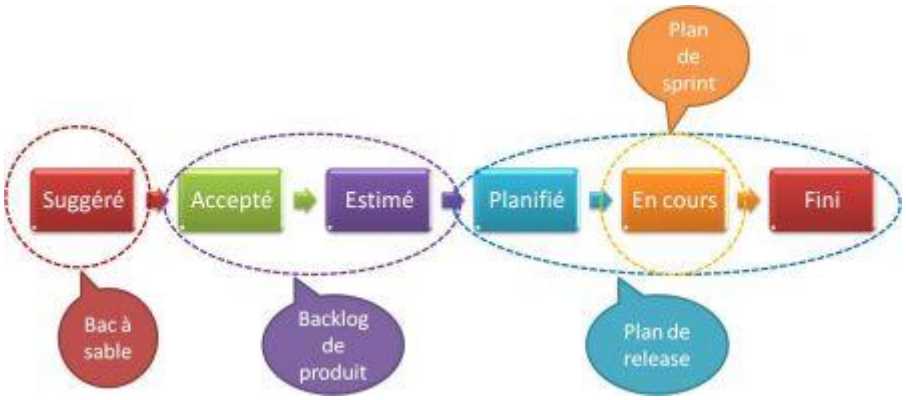


Figure 14 - Etats des stories dans la planification

2.8. Outils utilisés

2.8.1. La Plateforme .Net

Le Framework .Net est une plateforme de développement d'application informatique. C'est-à-dire qu'elle fournit une architecture et des services pour construire, déployer et exécuter des applications de tout type sur de nombreux appareils compatibles avec l'environnement système Windows. On peut citer les applications appelées « de bureau » pour les ordinateurs personnels, les composants et les services exécutés sur des serveurs mais également les applications pour les téléphones ou les tablettes.

La prise en charge de tous les appareils et de tous les types d'application conduit à une unification du processus de développement sur le système d'exploitation Windows.

On décrira dans un premier temps l'architecture du modèle de programmation pour examiner ensuite les services mis à disposition du programmeur.

2.8.1.1. Vue d'ensemble

Un des rôles premier du système d'exploitation est de permettre le chargement et l'exécution de programmes. Cependant, une application réalisée sur la plateforme .Net ne sera pas chargée ni exécutée directement par le système d'exploitation. Elle sera prise en charge par un moteur d'exécution CLR (Common Language Runtime). On parle d'applications managées. Cette approche est similaire au principe de la machine virtuelle. Les applications .Net sont isolées les unes des autres. Il est possible d'avoir simultanément

plusieurs moteurs d'exécution en fonctionnement. On trouve ce cas de figure lorsque l'on utilise le serveur Web IIS (Internet Informations Services) qui met à disposition un site basé sur ASP.Net.

La figure suivante issue du site internet de l'éditeur indique le positionnement de la plateforme .Net par rapport au système d'exploitation.

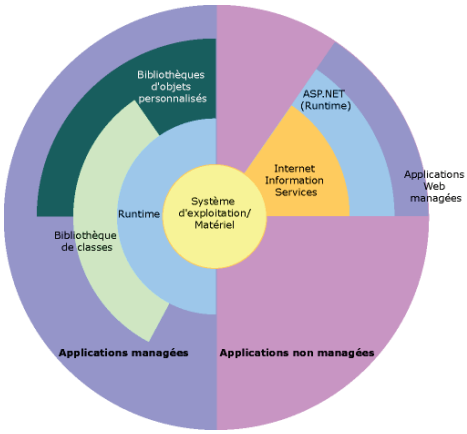
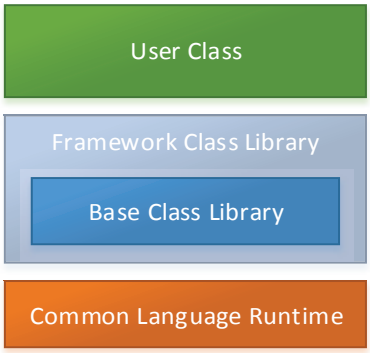


Figure 15 - Architecture du Framework .Net

Le Framework .Net se compose principalement des 3 éléments suivants qui sont fournis avec le système d'exploitation :



- **Common Language Runtime (CLR)** : C'est l'environnement d'exécution des applications.
- **Base Class Library (BCL)** : Les bibliothèques de classes contiennent toutes les routines de bas niveaux sous forme d'objets. Elles couvrent tous les aspects depuis la communication avec le système d'exploitation comme par exemple la gestion des fichiers.

- **Framework Class Library (FCL)** : Contient un ensemble de technologies pour simplifier la création d'application.

2.8.1.2. Processus de développement

Le processus de développement d'une application managée est proche de celui d'une application traditionnelle. Des fichiers sources sont traduits par un compilateur et assemblés dans un fichier binaire (de type EXE ou DLL) avec les ressources associées (images, curseurs, etc …)

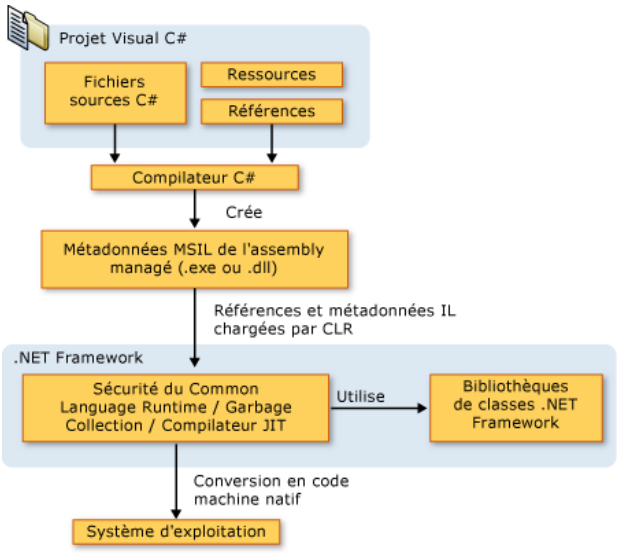


Figure 16 - Schéma de compilation et d'exécution d'un programme C#

Les spécificités de ces assemblages sont l'inclusion de métadonnées et la compilation dans le langage MSIL (Microsoft Intermediate Language) qui n'est pas spécifique à la plateforme d'exécution. C'est en effet le CLR qui finira la compilation au moment de l'exécution afin de vérifier la sécurité des appels de fonctions et d'optimiser le code machine en fonction du processeur. L'inclusion des métadonnées permet à un programme de s'auto examiner et apporte des possibilités de comportements dynamiques.

2.8.1. Environnement de développement intégré

La programmation en langage C# est très simple en utilisant les outils mis à disposition par Microsoft. Visual Studio 2013 Ultimate édition est choisi comme EDI en partant du principe que la version simplifiée Visual C# Express 2013 doit être capable de compiler l'ensemble du projet. Le choix de la version Ultimate est basé sur les capacités de

débogage avancé qui permettent de naviguer graphiquement entre les appels de méthodes dans des graphes d'objets complexes.

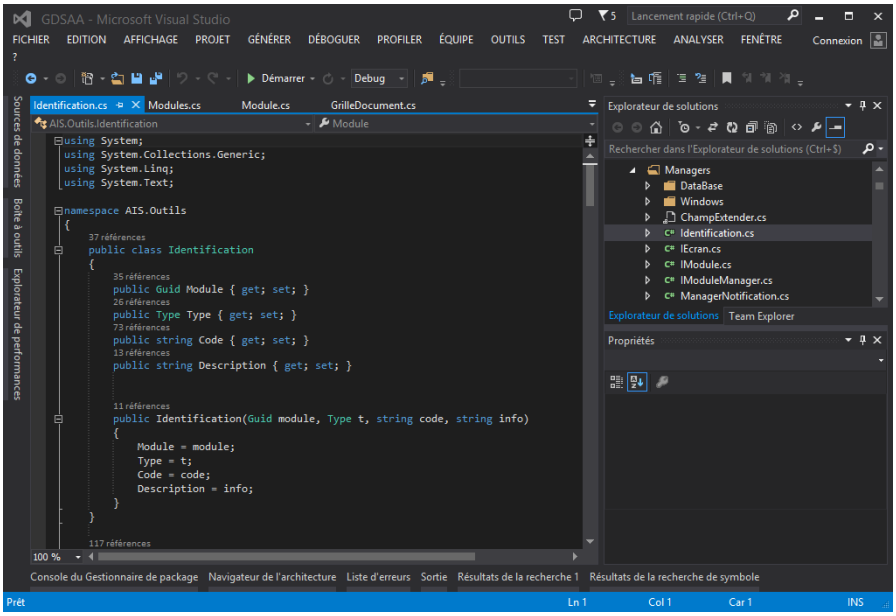


Figure 17 - Visual Studio 2013 Ultimate

2.8.2. Système de Gestion de Base de données

Une base de données structure et mémorise les données pour une utilisation ultérieure par des programmes informatiques. L'organisation des données respecte un modèle indépendant de celui mis en œuvre dans les applications. Historiquement les modèles proposés sont successivement :

- le **modèle hiérarchique** : il décrit les enregistrements logiques par une structure arborescente. Un parent peut avoir plusieurs enfants mais un enfant ne peut avoir qu'un seul parent. Ce modèle est donc adapté pour décrire les relations de type « un à plusieurs ».
- le **modèle réseau** est une extension du modèle précédent, il introduit les relations de type « plusieurs à plusieurs ».
- le **modèle relationnel,** défini en 1970 par Edgar Frank Codd, propose de représenter les enregistrements logiques d'une base de données sous forme de relation. Une relation est une table bidimensionnelle où les colonnes représentent des champs (ou attributs) et les lignes des enregistrements logiques.

- le **modèle de données orienté objets**, représente les données sous forme d'objets. Ce modèle utilise des concepts issus de la programmation et de la conception orientées objet. Un objet représente un élément concret ou abstrait du monde réel. Ce modèle autorise la définition d'attributs par des types structurés ou par héritage.

Le fonctionnement d'une base de données repose sur un ensemble de programmes et de fichiers que l'on appelle SGBD (Système de Gestion de Base de Données). Un SGBD doit assurer

- la **cohérence des données** : un SGBD doit permettre la définition des contraintes d'intégrité au sein de la base de données.
- la **concurrence des accès** : lorsque plusieurs utilisateurs désirent accéder en même temps aux mêmes données, le SGBD doit gérer cette concurrence d'accès en ordonnançant les demandes.
- la **confidentialité des données** : un SGBD doit permettre le contrôle des accès lors de la création, la modification, la consultation et la suppression des données. Ce contrôle est réalisé par l'utilisation de mots de passe ou par le cryptage des données.
- la **sécurité des données** : un SGBD doit assurer la sécurité des données contre les incidents matériels ou logiciels.

Enfin, un SGBD doit également assurer le **suivi des opérations** en fournissant des statistiques sur les utilisations de la base des services de gestion. Ces services sont accessibles par des langages spécifiques de description et de manipulation de données. Le langage de manipulation SQL (Structured Query Language) est actuellement le plus répandu.

Le dictionnaire de données recense l'ensemble des données et leurs caractéristiques définies par LDD (Langage de Définition des Données).

2.8.2.1. Choix du SGBD

Les principales caractéristiques qui affectent le choix d'un SGBD sont la taille de la base de données, le nombre d'utilisateurs pouvant travailler simultanément, les

performances attendues en terme de temps de réponse, l'intégration du SGBD à d'autres systèmes, la disponibilité de fonctions spécifiques, l'éditeur et son coût.

La taille de la base de données dépend du nombre de tables et d'enregistrements logiques nécessaires à la mémorisation des données. Cette taille définit d'une part la capacité de la mémoire auxiliaire nécessaire pour stocker la base de données et, d'autre part, le type de SGBD requis. La plupart des SGBD peuvent gérer des bases de données de quelques Mo mais peu ont la capacité de gérer des bases de plusieurs To. Le nombre d'utilisateurs pouvant accéder simultanément à la base de données est également un critère important pour le choix d'un SGBD car les utilisateurs ne doivent pas être pénalisés dans l'exécution de leur activité quotidienne. L'**évolutivité** est le critère utilisé pour définir la capacité du SGBD à intégrer une augmentation de la taille de la base de données ou du nombre d'utilisateurs.

La rapidité de consultation ou de mise à jour, dans certains cas, est un critère essentiel dans le fonctionnement d'une organisation. La **performance** d'un SGBD est le critère qui englobe les temps de réponse à une requête qu'elle soit de consultation, d'ajout, de mise à jour ou de suppression.

Certaines organisations s'appuient sur différentes bases de données pour gérer leurs activités. Il est donc indispensable dans ce cas de disposer de SGBD capables d'intégrer les données des différentes bases par des mécanismes d'importation et d'exportation. L'**interopérabilité** est la capacité d'un système à fonctionner avec d'autres systèmes (existants ou futurs).

Les fonctions de base d'un SGBD sont généralement fournies par l'ensemble des systèmes. Certains éditeurs portent une attention particulière sur la facilité d'emploi, la documentation, etc. Ces caractéristiques peuvent influer sur le choix du SGBD car elles peuvent diminuer certains coûts tels que les coûts de formation.

La taille, la réputation, la stabilité financière du fournisseur, la proximité et l'assistance fournie, notamment par l'aide en ligne ou la communauté des utilisateurs, sont également des critères qu'il faut prendre en compte lors du choix d'un SGBD.

Le coût des SGBD est très variable. Il oscille entre gratuit et plusieurs milliers d'euros. Il ne faut pas se limiter aux coûts de mise en œuvre, il est également important d'intégrer les coûts d'exploitation et de maintenance.

Le SGBD retenu est SQL Server version Express 2008 R2 pour l'intégration avec la plateforme .Net, son prix et la maitrise actuelle du langage TSQL par l'équipe de développement. Les versions plus récentes ne sont pas compatibles avec Windows XP. En complément du moteur de base de données, on dispose de l'environnement de gestion SQL Management Studio. Il permet, en plus de l'accès à l'intégralité des fonctions de paramétrage de SQL Server, la mise au point de requête avec un support de la complétion et d'un diagnostic des plans d'exécution.

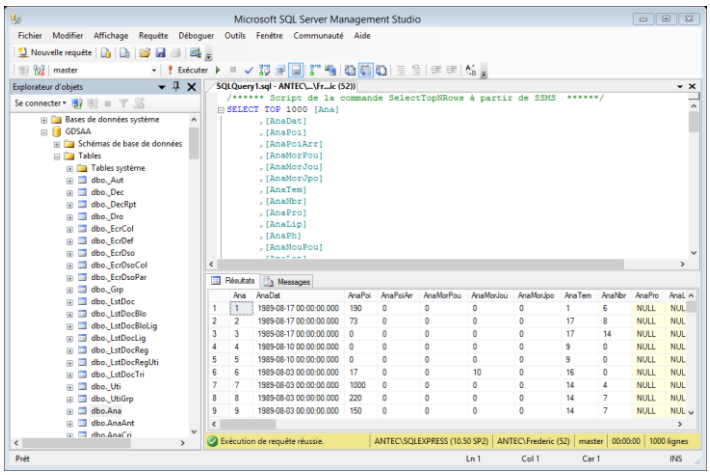


Figure 18 - Gestionnaire SQL Management Studio

Un moniteur d'activité permet également de visualiser en temps réel toutes les commandes SQL reçues pas le moteur de base de données. Il existe également avec chaque version de SQL Server un éditeur d'état. La version 3.0 de **Report Builder** est retenue car c'est la seule qui permet une édition simple des états pour l'utilisateur final. Cet aspect est important pour le futur projet **POISSONS V2**.

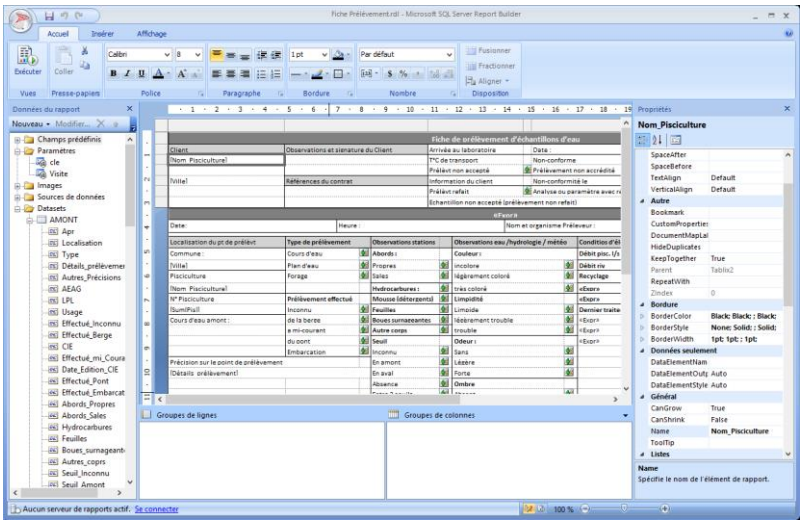


Figure 19 - SQL Server Report Builder 3.0

2.8.3. Gestion de code Source

Il n'est pas raisonnable aujourd'hui d'envisager un développement collaboratif sans une plateforme de gestion de code source. Microsoft fournit la solution TFS (Team Foundation Server) mais elle est limitée en nombre de projets pour la version *Express*. J'ai donc opté pour un serveur *Subversion* avec la solution *Visual SVN Server*. Le client choisi est *Tortoise SVN* à la place du client natif de *Visual Studio* car le plugin *VisualSVN* n'est pas compatible avec les versions *Express* de Visual Studio.

2.8.3.1. Visual SVN Server

Le serveur se présente sous la forme d'un service Windows. Il se configure par l'intermédiaire d'une console de gestion de type MMC (Microsoft Management Console). La liste des dépôts est accessible dans une arborescence dans laquelle on navigue comme dans un explorateur de fichiers. Un assistant permet en quelques clics de créer un nouveau dépôt basé sur un modèle standard. Les fichiers peuvent également être consultés via un navigateur Internet.

La gestion des utilisateurs permet de créer des groupes et d'appliquer sur chaque dépôt des autorisations en lecture et en écriture.

Il est possible de déclencher des traitements lorsque des évènements surviennent au niveau d'un dépôt comme par exemple avant ou après un verrouillage ou une validation. Je n'ai pas eu besoin d'activer cette fonctionnalité.

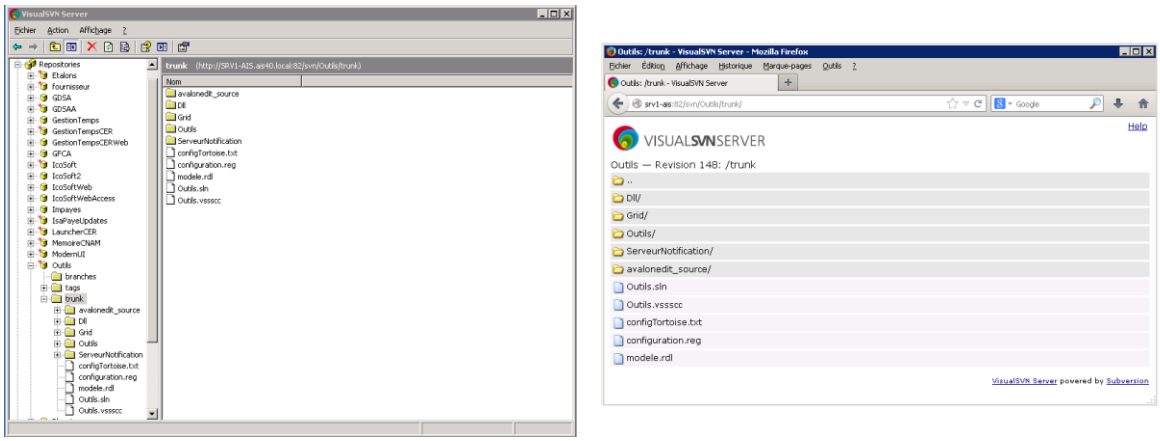


Figure 20 - Visual SVN Server

2.8.3.2. Tortoise SVN

Tortoise SVN est un client Open Source pour le système de contrôle de version *Subversion*. Il s'intègre dans l'explorateur de fichier de Windows sous forme de menu contextuel et d'icônes positionnées en surimpression des icônes de dossier et de fichiers.

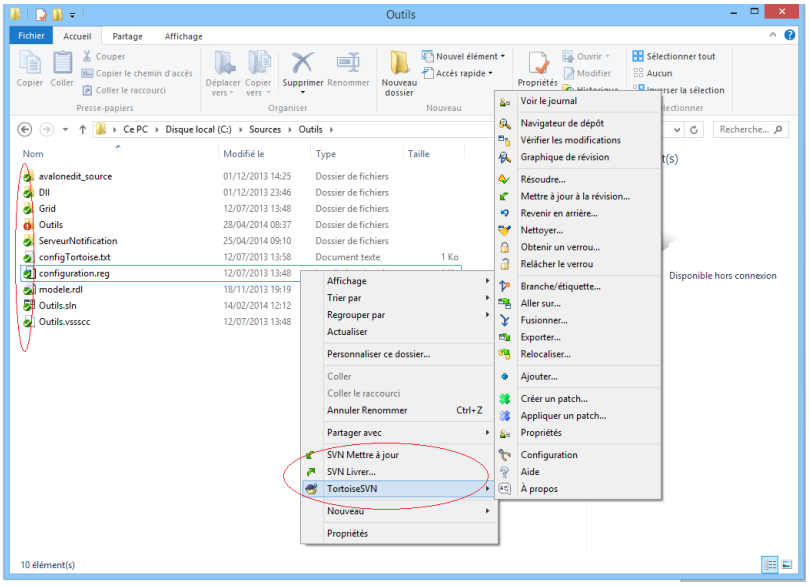


Figure 21 - Tortoise SVN

Ce client n'est pas spécifique à un langage de programmation. Il doit donc être paramétré afin de détecter les nouveaux fichiers à ajouter lors d'une opération de livraison. On évite ainsi de stocker dans le contrôleur de code source tous les fichiers temporaires produits par l'environnement de développement.

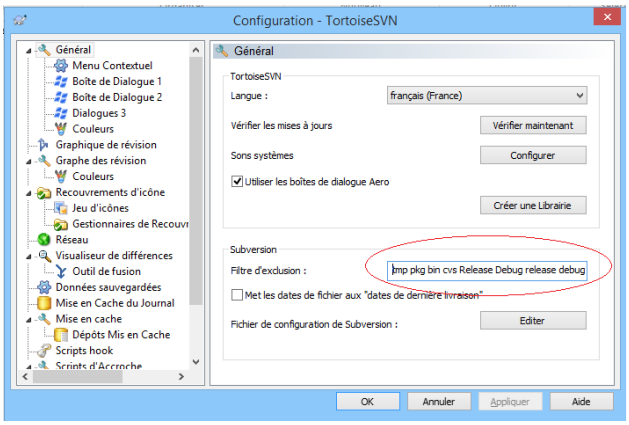


Figure 22 - Configuration de Tortoise SVN

2.8.4. Pilotage de projet

La gestion de projet est réalisée avec l'outil Open Source *IceScrum* qui propose une implémentation de SCRUM très proche de la méthode. La version communautaire n'est pas prévue pour supporter SQL Server. J'ai dû apporter les modifications présentées dans l'Annexe 4 à la configuration par défaut.

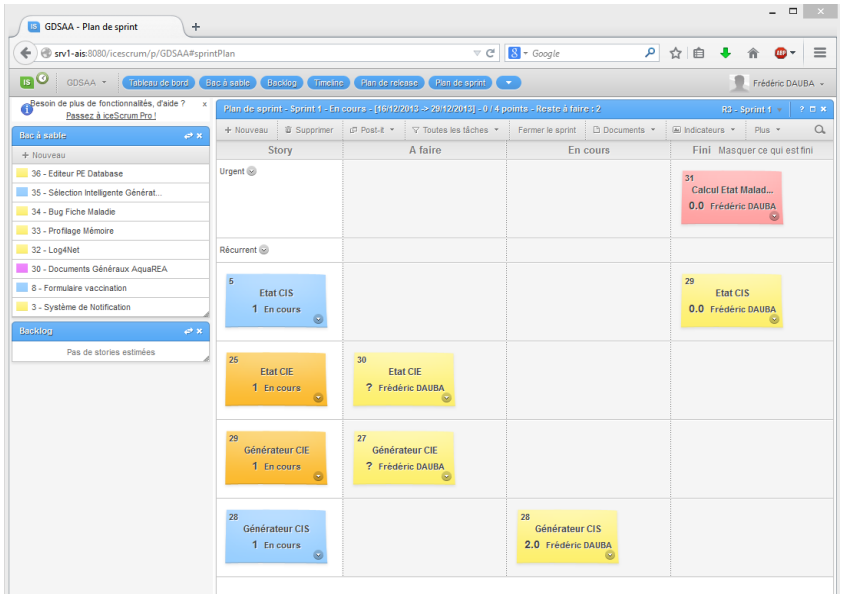


Figure 23 - Interface IceSCRUM

3. Le projet Outils : la plateforme de développement rapide

3.1. Présentation

La plateforme de développement rapide est constituée par un ensemble d'éléments utiles dans le cycle de vie d'une application spécifique de gestion de données. Son objectif est de réduire le temps de réalisation et de maintenance. On vise à moyen terme un gain de réactivité et une plus grande maitrise des coûts de développement. A plus long terme, l'objectif est de garder la maitrise de l'ensemble des outils en interne.

La particularité de cette boite à outils est d'être, en partie, destinée à l'utilisateur final. Sous réserve d'un niveau suffisant en informatique, ce dernier pourra même maintenir l'application ou ajouter de nouvelles fonctionnalités. En effet, comme le montre la Figure 24 suivante, la bibliothèque dispose d'un système de compilation et d'intégration dynamique du code source. Cela signifie qu'une application réalisée avec la boite à outils a la capacité de se modifier elle-même.

Sur cette figure, le niveau de vert correspond au degré d'utilisation par l'utilisateur final des éléments de la boite à outils.

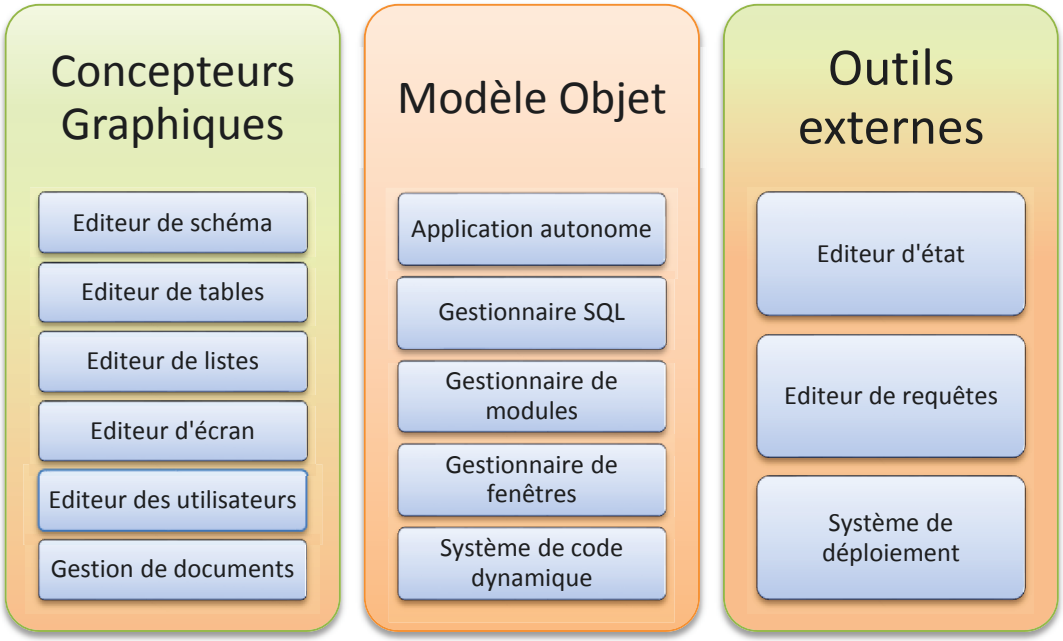


Figure 24 - Eléments de la boite à outils

Le tableau suivant présente les éléments utiles de la boite à outils en fonction des phases du cycle de vie d'une application.

Tableau 2 - Cycle de vie d'une application et élements de la PDR

Phase	Module
Planification	
Spécification	
Conception	Editeur de schéma, Editeur d'écran
Développement	Editeur de tables, Editeur de listes, Editeur d'écran, Editeur d'état, Editeur de requêtes, Gestionnaire SQL, Gestionnaire de modules, Gestionnaire de fenêtres, Système de code dynamique, Application autonome
Tests	
Déploiements	
Maintenance	Editeur de tables, Editeur de listes, Editeur d'écran, Editeur d'état, Editeur de requêtes, Gestionnaire SQL, Gestionnaire de modules, Gestionnaire de fenêtres, Système de code dynamique

3.2. Gestion des risques

L'activité de suivi des risques dans un projet est continue. Un risque est défini dans PMBOK par « Un évènement ou une situation dont la concrétisation, incertaine, aurait un impact négatif sur au moins un objectif du projet. ». Les types de risque selon le PMI se classent parmi :

- les risques techniques
- les risques externes
- les risques organisationnels
- les risques de management

Dans un projet agile, la gestion du risque n'est pas très formalisée mais elle est quotidienne. C'est le SM qui doit détecter et éliminer les sources de blocage de l'équipe de développement.

3.3. Architecture technique cible

Il existe actuellement chez AIS 2 types de profil client :

- l'entreprise individuelle ou familiale (généralement agricole)
- les PME

L'architecture retenue dans ce contexte est une application de type client lourd fonctionnant uniquement en local ou via le réseau local pour contacter le SGBD en mode multi-utilisateurs.

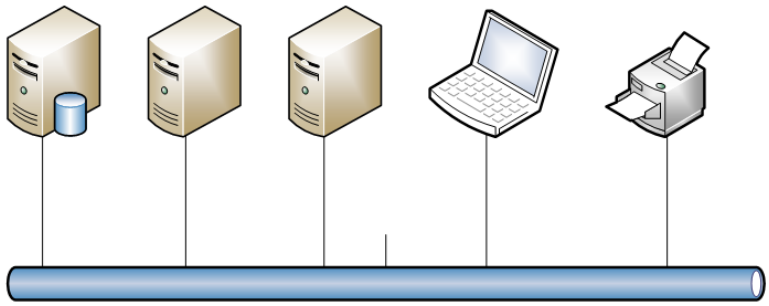


Figure 25 - Réseau local classique

Les environnements systèmes à supporter sont les suivants :

- Windows XP SP3
- Windows Vista
- Windows 7
- Windows 8
- Windows 8.1
- Windows 2003 Server R2
- Windows 2008 Server R2
- Windows 2012 Server

Le choix se limite à la version 4.0 du Framework.Net puisque c'est la version la plus récente, compatible avec l'ensemble de ces systèmes d'exploitation.

3.4. Analyse, conception

3.4.1. De la vision aux stories

La mise en place de méthode Agile conduit à ne pas utiliser de cahier des charges détaillé dans lequel tout ou presque a été imaginé. Cependant, la question de la définition de « par quoi on va commencer » demeure. Le risque est grand de partir sur de mauvaises pistes. Il faut trouver un moyen de garantir l'alignement avec les objectifs métiers du projet.

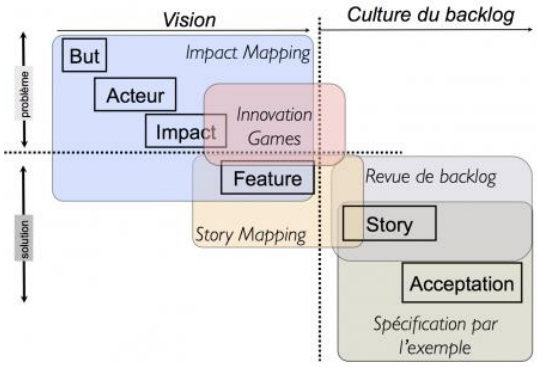


Figure 26 - Etapes de définition de la vision du produit

La technique de l'*Impact-Mapping* lancée par Gojko Adzic vise à répondre à cette problématique. Elle s'appuie sur la réalisation d'une carte heuristique qui contient :

- le but recherché
- les acteurs
- les impacts sur les acteurs pour arriver au but
- les *features* qui sont les solutions aux impacts

Il reste bien sûr le choix du but. Ce doit être quelque chose de mesurable sinon comment savoir s'il est atteint ? La pratique d'*innovation Games* (IG) est suggérée dans [2] pour déterminer des buts possibles. L'une d'entre elles, l'IG du futur est parfaitement adaptée et se déroule de la façon suivante :

On réunit l'équipe et on demande à chacun d'énoncer pourquoi, si on imagine être après la fin du projet, ce dernier a été une réussite. Parmi les propositions il faut ensuite déterminer celles qui pourraient être indispensables à la 1ere *release*.

Les choix proposés sont les suivants :

- syntaxe allégée pour la lisibilité, la maintenance et une courbe d'apprentissage réduite
- scénarios standards de recherche, de consultation et de saisie pris en charge avec la possibilité pour le développeur d'intervenir à chaque étape si nécessaire afin d'implémenter d'éventuelles règles de gestion. On peut imaginer une zone statut qui change automatiquement de valeur en fonction des valeurs renseignées dans d'autres zones.
- automatismes dans la gestion des écrans. La liaison entre l'interface utilisateur et les données prise en charge entièrement

- personnalisation des écrans, des données et des traitements par l'utilisateur final.
- pouvoir réaliser une application, dans le cas le plus extrême, sans écrire une seule ligne de code
- masquer les subtilités techniques de la base de données avec un éditeur de table, de requêtes et d'états simplifiés
- réaliser un programme

J'ai choisi pour la *release* #1 que le but visé est la création de l'application simple similaire à celle requise pour le projet Gestion de congés :

« Réaliser un programme de paramétrage avec la PDR »

L'identification des acteurs est dans notre cas un peu confus car il s'agit de l'équipe de développement elle-même. Il faut bien lire schéma d'IM en considérant que le développeur utilisera la version opérationnelle de la PDR pour réaliser une application appelée A.

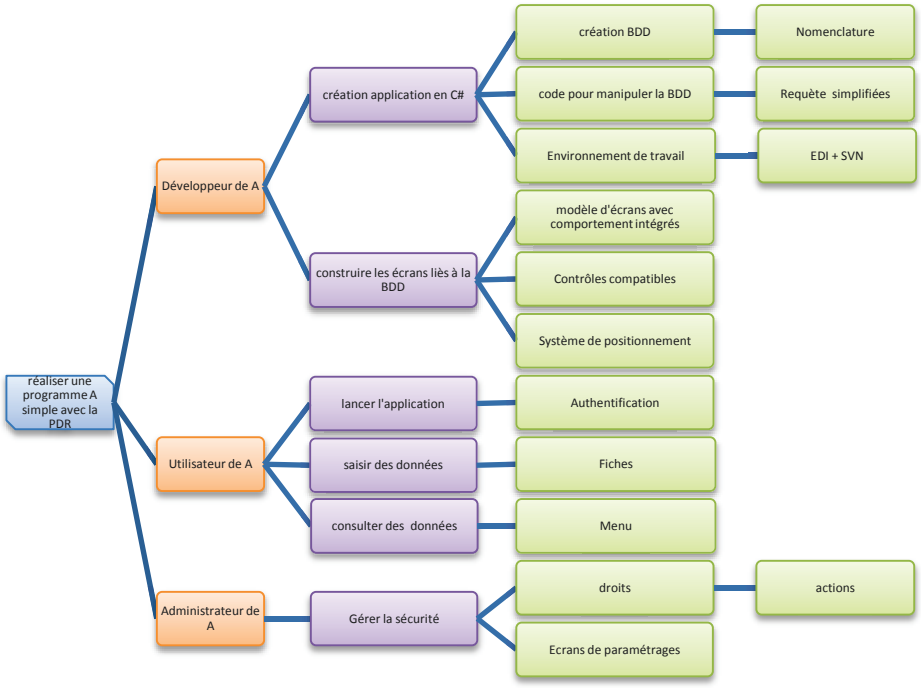


Figure 27 - Carte heuristique

On peut synthétiser les *features* nécessaires de la PDR

- Mise en place de l'environnement de travail
- Gestionnaire de base de données

- Gestionnaire de fenêtres
- Authentification et droits
- Système de configuration
- Ecrans de base (menu, fiche et écran de recherche)
- Contrôles avec système de positionnement simple

L'ensemble de ces fonctionnalités fera partie intégrante de la première version de la boîte à outils.

3.4.2. Version initiale

La *release #1* doit fournir les mécaniques opérationnelles de base. Cela comprend la base de données, les classes utilitaires pour la mise en œuvre simplifiée pour le développeur.

Le premier objectif est de rendre accessible au développeur les fonctionnalités par du code le plus simple possible. L'objectif suivant est de créer des tables pour paramétrer l'utilisation de ces fonctionnalités afin de supprimer le plus de code possible. Enfin il faudra construire des éditeurs graphiques qui alimenteront les tables de configuration. En procédant dans cet ordre, il ne restera au développeur qu'à écrire le code spécifique à son application. De plus, les fonctionnalités jugées non essentielles pour les premières versions des éditeurs graphiques seront disponibles par code.

3.4.3. Maquettage et ergonomie

Avant de commencer le projet, nous avons réalisé une maquette d'une application standard afin d'imaginer une ergonomie et les modalités de navigation entre les différents écrans. Nous avons retenu comme critère la présence de listes et de fiches qui sont des éléments incontournables dans les applications de gestion. Les premières offrent une bonne capacité de recherche et les secondes un détail élevé d'informations. Nous voulions également que la consultation simultanée de différentes fiches ou listes soit possible. Cela permet également de consulter des données sans interrompre une saisie.

Pour répondre à ces différents critères nous avons opté pour un modèle d'application utilisant des onglets. La Figure 28 montre l'application qui s'ouvre sur un premier onglet dont la fermeture est impossible, servant ainsi de menu principal.

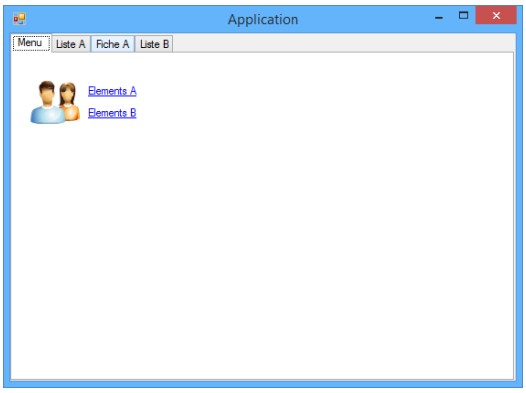


Figure 28 - Maquette de l'écran de menu

La navigation entre les écrans s'effectue au moyen de lien hypertexte comme pour des pages internet. Ainsi chaque onglet contient soit un écran de type liste soit un écran de type fiche soit un écran spécifique.

Les cheminements possibles sont les suivants :

Menu -> Ecran de recherche -> fiche en consultation

Menu -> Ecran de recherche -> Fiche en mode création

Menu -> Ecran spécifique

Fiche -> Ecran Spécifique

La structuration d'un écran de recherche pourrait être présentée comme le montre la Figure 29 ci-dessous.

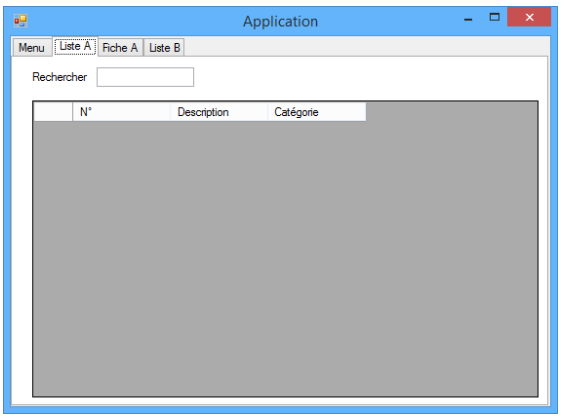


Figure 29 - Maquette d'un écran de type liste

A chaque changement du contenu de la zone de filtre, un filtrage s'opère sur la liste de résultat. Une des colonnes sera de type lien hypertexte afin d'ouvrir la fiche correspondante.

L'aspect commun des fiches nécessite d'identifier au premier regard le contenu de la fiche. C'est pourquoi une zone d'entête doit contenir les informations d'identification essentielles. La présentation en onglet permet de gérer la place disponible en favorisant la disposition par thème. Le recours aux barres de défilement doit être évité pour accélérer la navigation.

La modification est déclenchée explicitement et déverrouille les zones de saisie.

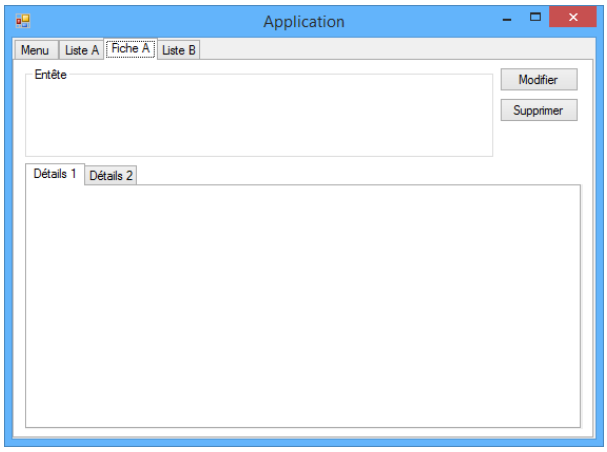


Figure 30 - Maquette d'un écran de type fiche

3.5. Mise en œuvre

J'ai travaillé en binôme avec Kévin Sébie pour la réalisation de la première *release*. Il effectuait chez AIS un stage de 10 semaines dans le cadre de sa 2ème année d'étude en DUT d'informatique.

Cette release a pour principal objectif la mise en place des fondations techniques de la bibliothèque d'outils.

3.5.1. Mêlée 1

3.5.1.1. Nomenclatures pour les bases de données

Avant de commencer à travailler avec une base de données, il convient de définir les règles à adopter pour nommer les éléments tels que les tables et les champs. En l'absence

de formalisme il devient très rapidement difficile de retrouver les informations alors même que le but de la base de données est de faciliter leur exploitation. La maintenance des applications est également impactée car les noms utilisés dans la base de données vont inévitablement apparaitre à de nombreuses reprises comme lors de l'écriture des requêtes ou lors de la liaison de données avec l'interface utilisateur. Quelle que soit la nomenclature adoptée, un utilisateur trouvera cela toujours trop complexe. En effet, un développeur ou un concepteur prendra en compte des aspects techniques comme les notions de tables, de vues, de clé primaire ou étrangère alors que l'utilisateur ne s'intéressera qu'à l'identification de la donnée en termes de contenu.

Pour répondre à ces 2 objectifs divergents, j'ai opté pour un système de vues générées automatiquement par un éditeur de table simplifié. Cette approche donnera à l'utilisateur l'opportunité de travailler avec des libellés plutôt que des noms de tables ou de champs. L'éditeur assurera également le respect de la nomenclature des contraintes suivantes :

- Clé primaire : PK_Aaa
- Clé étrangère : FK_Aaa_Bbb

La réalisation de cet éditeur de schéma de base de données sera abordée dans la *release #2* qui a pour objectif l'ajout des fonctionnalités pour l'utilisateur final.

La convention adoptée pour les noms des tables et des champs est la suivante :

Tous les noms sont composés de blocs de 3 lettres dont le premier caractère est en majuscule afin de faciliter la relecture. Par exemple « anamorpou » est visuellement plus confus que « AnaMorPou ». Chaque bloc peut être vu comme une abréviation d'un mot. Dans l'exemple il s'agit de : Analyse Mortalité et Pourcentage.

Pour supprimer les ambiguïtés qui pourraient subvenir lors des jointures, un préfixe est ajouté systématiquement aux noms des champs.

Tableau 3 - Nomenclature des tables et des champs

Nombre de blocs dans le nom de la table	Préfixe	Exemple	Exemple Préfixe
1	Le nom de la table	Ana	Ana
2	1ère lettre du Bloc 1 et 1ère et 2ème lettre du bloc 2	AnaLig	Ali
3	1ère lettre de chaque bloc	AnaMnuCat	Amc
4 ou plus	1ère lettre de chaque bloc avec si possible une combinaison qui n'est pas déjà utilisée. Si ce n'est pas possible on augmente la longueur du bloc pour assurer l'unicité.	AnaMnuCatLig	Aml ou Amcl

Le champ nommé uniquement du préfixe sera la clé primaire. Toutes les clés primaires sont non significatives, c'est-à-dire qu'elles sont définies comme des compteurs automatiques de type **bigint**. Des tables seront utilisées par la plateforme elle-même. Pour les distinguer clairement des tables utilisateurs, le caractère « _ » sera utilisé.

Une clé étrangère sera composée de la juxtaposition des préfixes.

Dans la Figure 31 - Exemple de convention de nommage des tables, on peut voir la table « Analyse », « Analyse Lignes » et une table système « Déclarations ».

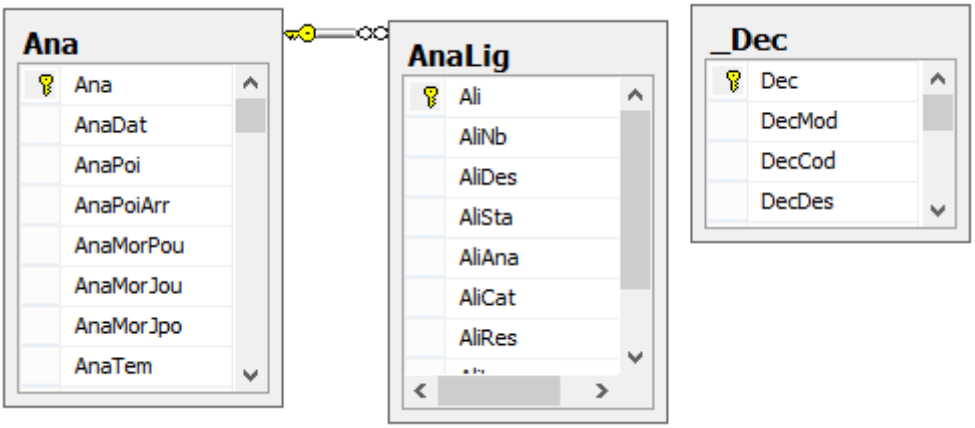


Figure 31 - Exemple de convention de nommage des tables

Une requête utilisant les 2 tables Analyses prendra la forme suivante :

```
Select Ana, AnaDat, AliNb, AliSta
From Ana
inner join AnaLig on Ana = AliAna
Where AndDat => '1/1/2014'
```

Type de données supportées

Il existe de nombreux formats de données disponibles dans *SQL Server*[7] mais dans le cadre d'une application de gestion, seuls quelques-uns sont nécessaires*. ADO.Net* dispose d'un pilote spécifique pour SQL Server dans l'espace de nom *Data.Sqlclient*. Le tableau suivant donne la correspondance entre les types de données *C#* et *SQLServer* :

Tableau 4 - Correspondance des types de données

Type SQL	Type .Net
int	Int32 ou int
bigint	Int64 ou long
nvarchar (taille) (unicode)	String
datetime	DateTime
float	Double

La taille des chaînes de caractères influe grandement sur l'espace occupé par une table. Les règles suivantes sont adoptées : les tailles maximales de chaînes sont à 50 caractères par défaut, les chaînes plus longues sont de 250 et les chaînes très longues multi lignes sont de 2500.

3.5.1.2. Problématiques de l'exploitation de la base de données

L'écriture de la bibliothèque d'outils, tout comme le code que le développeur devra écrire pour personnaliser son application, nécessitera de nombreuses lectures ou mises à jour de la base de données. Ces actions doivent être les plus simples possible. Les étapes requises pour utiliser une base de données sont peu nombreuses mais elles peuvent rapidement produire un nombre de lignes de code important.

Avec ADO.Net on dispose de 2 approches pour travailler avec la base de données : le mode connecté et le mode déconnecté.

Dans le mode connecté on procède simplement par :

- ouverture de la connexion

[7] Voir Annexe 1 – Type de données SQL Server 2008 R2

- exploitation des données en lecture et écriture
- fermeture de la connexion.

Ce mode permet de spécifier lors d'une lecture que l'on souhaite bloquer les données pour une prochaine mise à jour. Ce système de verrou combiné avec des transactions permet d'éviter les accès concurrentiels sur des traitements complexes. Cependant pour un usage plus basique, cela réduira les performances du serveur. En effet le moteur de base de données aura tendance à verrouiller des blocs de lignes connexes plutôt que la ligne exacte. De plus la vérification des verrous ralentira les lectures pour les autres utilisateurs.

J'ai donc choisi d'utiliser un accès à la base en mode déconnecté afin de ne pas maintenir de verrou côté serveur durant les périodes où les fiches seront en mode modification. Cette technique consiste à établir une connexion, exécuter le chargement de données puis fermer immédiatement la connexion. C'est alors au client de détecter les conflits qui peuvent survenir si plusieurs utilisateurs essaient de modifier simultanément les mêmes données.

Chargement de données en mode déconnecté

Les classes suivantes doivent être utilisées

- SqlConnection
- SqlCommand
- DataTable
- SqlDataAdapter

Le diagramme de classe simplifiée de la Figure 32 met en évidence les propriétés et les méthodes utiles. Le code complet d'un chargement de données est présenté dans l'Annexe 5.

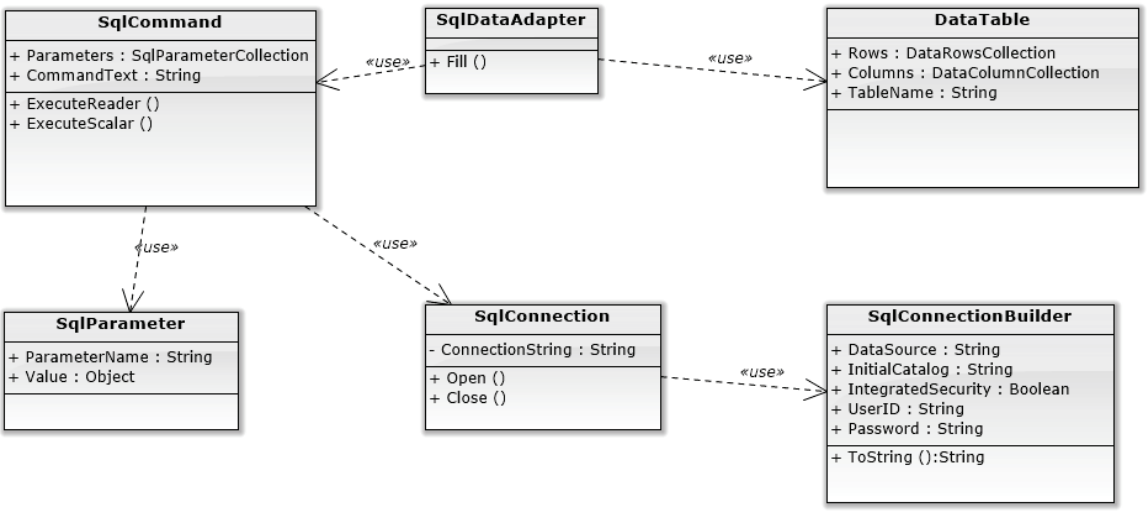


Figure 32 - Diagramme des classes utiles pour un chargement de données

Une fois les données stockées en mémoire dans le DataTable, il est possible de les modifier très facilement par l'intermédiaire des collections *Rows* et *Columns*. Les valeurs correspondantes aux champs dans ce modèle objet sont typées avec une classe générique[8] ce qui facilite les affectations mais impose de forcer une conversion lors de la lecture. Afin de faciliter cette opération la méthode générique *Field* permet de spécifier le type de donnée voulue. La notion de valeur *NULL* est implémentée en C# avec les types *nullables*. Il suffit d'ajouter le suffixe « ? » à un nom de type lors de sa déclaration pour bénéficier de propriétés supplémentaires *HasValue* et *Value*.

Les extraits de code suivants présentent la syntaxe à utiliser pour exploiter un type de donnée représentant une date qui supporte la notion de type nullable.

```
DateTime? d = null;
if (d.HasValue)
    Console.WriteLine(d.Value.ToString());
else
    Console.WriteLine("Date non définie");
```

d.Value est de type *DateTime* ce qui permet au compilateur d'effectuer tous les contrôles nécessaires pour la compilation du code.

[8] Le type *object* est le parent de toutes les classes en C#. Il permet donc de maintenir une référence vers n'importe quelle instance de classe.

```
DT.Rows[0]["Dat"] = DateTime.Now ;
DateTime ? d = DT.Rows[0].Field<DateTime ?>("Dat") ;
```

Enregistrement des modifications

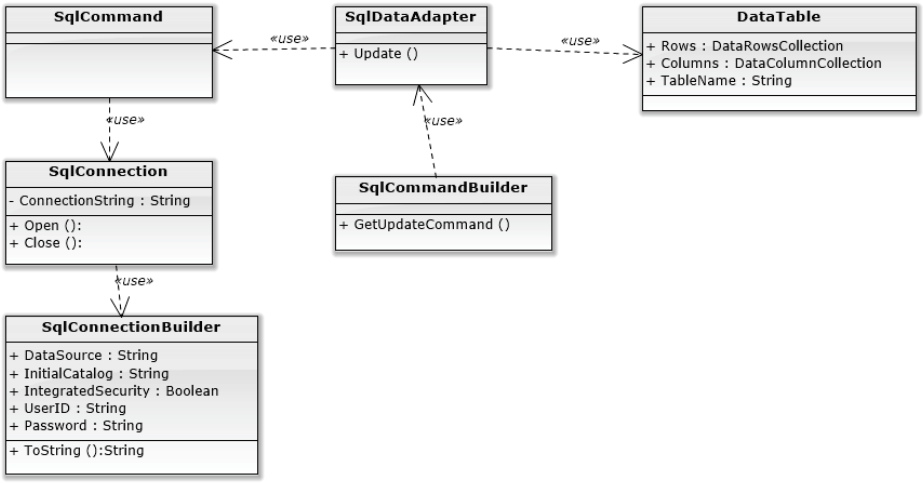


Figure 33 - Diagramme des classes utiles pour l'enregistrement de données

L'enregistrement des modifications effectuées dans le *DataTable* requiert les mêmes classes que pour la lecture ainsi que l'utilisation d'un *SqlCommandBuilder*. Il permet de générer automatiquement la requête SQL pour la mise à jour en fonction des informations fournies par le *SqlCommand* qui a servi pour l'extraction de données. La commande SQL ainsi générée est capable de détecter les accès concurrentiels car elle exploite les données avant modification que le *DataTable* a conservée. On peut noter que ce mécanisme fonctionne si le résultat du chargement concerne une seule table et que la clé primaire est parmi les colonnes sélectionnées.

La requête générée prend la forme suivante dans le cas d'une ligne modifiée comprenant 3 colonnes :

- ColPk : la colonne d'identification
- Col2 : une colonne dont la valeur est obligatoire
- Col3 : une colonne dont la valeur n'est pas obligatoire

```
Update {NomTable} Set Col1=@p1, Col2=@p2
WHERE ColPk=@p3 AND (Col1=@p4) AND ((@p5 = 1 AND Col2 IS NULL) OR Col2=@p6)
```

Les variables @p1 et @p2 sont les valeurs actuelles des champs Col1 et Col2 pour la ligne. La variable @p3 contient la valeur de la clé primaire. Les variables @p4 et @p6 sont les anciennes valeurs de Col1 et Col2. La variable @p5 a pour valeur 1 si @p6 a pour valeur NULL. On peut constater que la requête impactera la ligne dans la base de données si elle n'a pas changé depuis le chargement dans le *DataTable*. La méthode *Update* du *SqlDataAdapter* générera une exception si l'instruction SQL n'impacte aucune ligne.

Pour limiter au maximum la problématique de l'accès concurrentiel en mode déconnecté, il faut essayer de recharger les données avant de commencer à les modifier. On peut également les recharger lorsqu'un conflit est détecté. Puis on applique les modifications en comparant ce qui avait changé comme si finalement on avait commencé la modification après le dernier enregistrement dans la base de données.

Finalement j'ai opté pour un système de communication entre toutes les instances de l'application qui permet de synchroniser les rechargements et signale les entrées en modification d'un formulaire. Ce système de notification sera abordé dans un autre sprint dédié aux fonctionnalités du mode réseau.

Le problème des transactions en mode déconnecté

Pour réaliser un traitement qui impacte plusieurs tables, il faut recourir à des transactions afin de garantir la cohérence des données. Une fermeture de connexion qui n'est pas précédée d'une validation avec la commande COMMIT équivaut à une annulation. Pour pouvoir utiliser des *DataTables* dans le cadre d'une transaction, le *DataAdapter* ouvre automatiquement la connexion associée si nécessaire mais il ne la ferme pas. Cela laisse l'opportunité au développeur de gérer l'état de la connexion à la base de données.

La solution du gestionnaire DataBaseManager

Pour simplifier l'écriture du code, un gestionnaire de requête a été imaginé. Il permet d'exécuter des requêtes paramétrées.

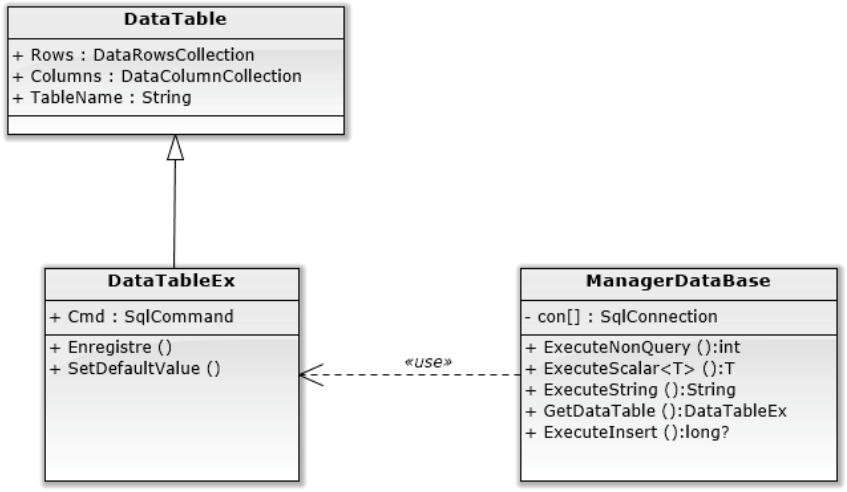


Figure 34 - Digramme de classes du gestionnaire de base de données

La première responsabilité du gestionnaire est la gestion de la connexion. Celle-ci est ouverte et fermée automatiquement. Avant chaque exécution de requête le gestionnaire vérifie l'état de la connexion. Si elle doit être ouverte alors elle sera refermée juste après l'exécution de la commande SQL. Pour éviter au programmeur de choisir quand ouvrir la connexion, une classe *Transaction* effectue cette opération dans les mêmes conditions que le gestionnaire de requête. Dans la pratique cela permet d'empiler les transactions et de ne pas se préoccuper lorsque l'on écrit du code si celui-ci sera utilisé dans un contexte transactionnel. Pour pallier toute éventualité, notamment aux erreurs imprévues, j'ai opté pour l'usage suivant de la classe *Transaction* :

Instanciation de l'objet *Transaction* dans une directive *using* qui forcera l'appel à la méthode *Dispose* par le runtime .Net. La méthode *Dispose* effectue une annulation des modifications si la méthode *commit* n'a pas été appelée.

```
using(Transaction t = new Transaction()) // Ouverture de la connexion
{
    // traitement
    using(Transaction t2 = new Transaction())
    {
        // Traitement
        t2.commit(); // sans effet
    }
    t.commit(); // Fermeture de la connexion
}
```

Les 5 fonctions suivantes couvrent les besoins d'exécution de commandes SQL :

- **ExecuteNonQuery** : cette méthode permet d'exécuter n'importe quel type d'instruction SQL à part les instructions de sélection de données. Le cas le plus courant est une commande de suppression de lignes. Le nombre de lignes impactées est retourné.
- **ExecuteScalar<T>** : permet la sélection d'une seule valeur et retourne la donnée convertie dans le type T fourni en paramètre. On peut citer par exemple une requête qui applique une opération de comptage pour déterminer le nombre de lignes qui correspondent à des critères particuliers.
- **ExecuteString** : c'est une variante de *ExecuteScalar*<T>. En effet la classe string ne dispose pas d'un constructeur sans paramètre contrairement à tous les types de base comme *int*, *long* ou *double*. Ce choix a été fait pour éviter de typer la fonction *ExecuteScalar* en *object* ce qui aurait forcé à un changement de type explicite.
- **ExecuteInsert** : permet l'exécution d'une insertion de ligne dans une table sans se préoccuper de la syntaxe SQL correspondante.
- **GetDataTable** : c'est la fonction principale qui permet d'obtenir un DataTable contenant le résultat d'une sélection de données.

Chacune de ces fonctions utilise les mêmes arguments. La première est une chaîne de caractères contenant l'instruction SQL. Dans cette chaîne, il ne faut pas insérer des données qui pourraient rendre la syntaxe incorrecte. C'est souvent le cas de chaînes de caractères contenant des apostrophes. Pour éviter tous les problèmes il convient d'utiliser des variables sous la forme @nomdevariable. Le deuxième paramètre est une classe représentant un dictionnaire d'objets correspondant aux valeurs des paramètres indexés par nom de variable. L'exemple de code suivant illustre l'utilisation de la fonction *GetDataTable* :

```
DataTableEx DT = ManagerDataBase.GetDataTable(@"select * from Ana
    where Ana=@id and AnaDes like @des", new ParametresSQL("id", 1234, "des", "%l'année%"));
```

3.5.1.3. Application minimale

La plateforme de développement est matérialisée par un *assembly* contenant toutes les classes nécessaires à l'exécution de l'application. Il n'est pas possible de démarrer une

application .Net autrement que par une fonction d'entrée statique située dans le fichier EXE. La seule opération à réaliser est l'instanciation de la classe *AppliAIS*.

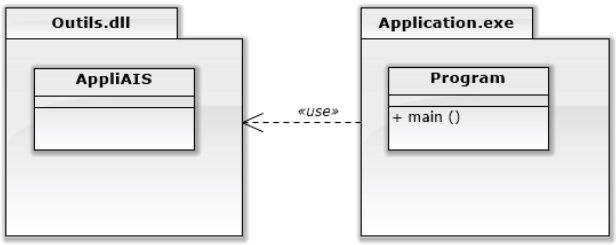


Figure 35 - Composants d'une application

On peut créer des applications avec interface graphique ou des applications de type console ou service Windows. J'ai choisi d'utiliser la technologie **WinForms** pour la partie interface utilisateur. Ce n'est pas la plus récente mais il existe des bibliothèques libres de droit de très bonne qualité qui répondront parfaitement aux besoins. J'ai utilisé les composants suivants :

- **WeifenLuo DockPanelSuite**[9] : Permet d'insérer des fenêtres filles dans une fenêtre parent avec un système d'onglet. Il est également possible d'accrocher les fenêtres sur les bords.
- **AvalonEdit**[10] : éditeur de texte avec coloration syntaxique utilisé pour le projet SharpDevelop. Il permettra la saisie des scripts pour la personnalisation de l'application une fois en production.
- **SourceGrid**[11] : un composant de type grille avec beaucoup de possibilités de paramétrage. La grille est parfaitement adaptée pour l'édition d'un *DataTable*.
- **DevExpress.XtraEditors**[12] : composants commerciaux qui ont bénéficié d'une promotion très intéressante : une gratuité à vie même pour une utilisation commerciale.

La séquence de démarrage de l'application est la suivante :

[9] http://sourceforge.net/projects/dockpanelsuite/

[10] http://www.nuget.org/packages/AvalonEdit/

[11] http://sourcegrid.codeplex.com/

[12] https://www.devexpress.com/products/net/controls/winforms/editors/

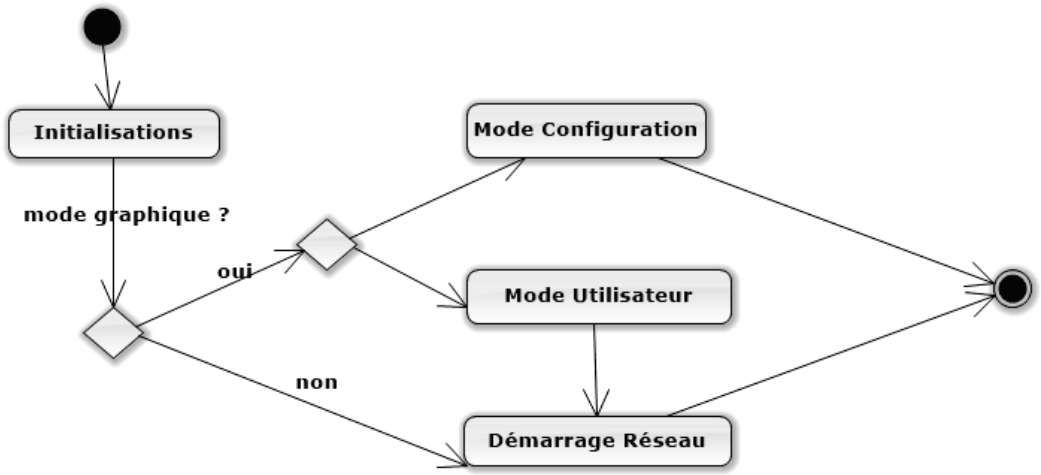


Figure 36 - Initialisation d'une application

Initialisations

Cette phase initiale doit charger les paramètres d'accès à la base de données qui ne doivent pas être fixés dans le programme. La plateforme .Net met à disposition un système de fichiers de configuration avec des fichiers XML mais j'ai opté pour le stockage dans la base de registre Windows des informations. La base de registre est plus simple à manipuler avec l'éditeur intégré à Windows et les programmes d'installation prennent tous en charge ce système.

L'application doit déterminer un nom de clé (AppName). La classe *AppliAIS* ira chercher en priorité dans l'emplacement :

HKEY_CURRENT_USER\Software\AIS\AppName\Configuration

Pour éviter de dupliquer la configuration pour chaque utilisateur de la même machine, un administrateur de l'ordinateur peut stocker les informations à l'emplacement suivant :

HKEY_LOCAL_MACHINE\Software\AIS\AppName\Configuration

Tableau 5 - Paramètres d'une application

Clé	Description	Type	Valeur(s)
Serveur	Instance SQL	Chaîne	
DataBase	Nom de la base de données	Chaîne	Machine\instance Machine, port
AuthWindows	Authentification Active Directory	Chaîne	True / False
User	Utilisateur authentification sql	Chaîne	
Password	Mot de passe authentification sql	Chaîne	
IPServer	Adresse du server réseau	Chaîne	
URL	URI du service Web de génération d'états. Ce paramètre est nécessaire à la génération d'état à distance[13]	Chaîne	

On peut voir sur la Figure 37 ci-dessous un exemple de la configuration de l'application *TempsCER*. La base de données **TempsCER** est disponible sur le server **srv1-ais** (port TCP 1472) avec une authentification intégrée à *SQL Server* avec le compte **sa**.

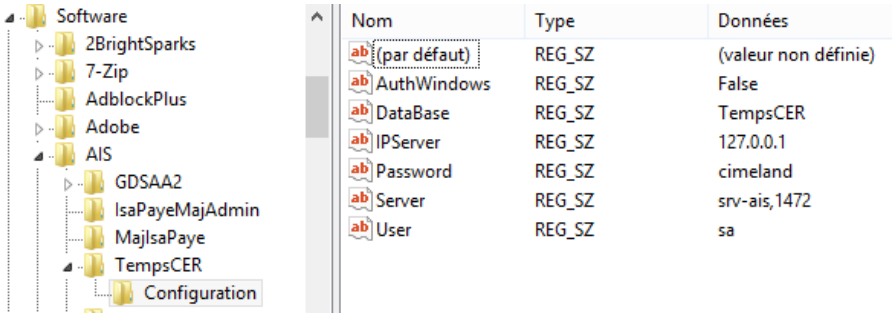


Figure 37 - Exemple de configuration dans la base de registre

La phase d'initialisation permet également de mettre en œuvre des éléments techniques présentés dans l'Annexe 6 comme l'interception des exceptions ou des fonctions de débogage optimisées.

La classe *AppliAIS* doit donc être initialisée avec au moins 3 paramètres qui sont :

- le titre de la fenêtre principale
- les arguments de ligne de commande
- le nom de la clé de registre

```
static void Main(string[] args)
{
    AppliAIS app = new AppliAIS("Titre", args, "cleRegistre");
}
```

[13] La génération d'état à distance est nécessaire pour contourner un bug présent dans Terminal Server lorsqu'un client n'a pas une résolution au ratio 4:3 et que le serveur est antérieur à 2008 R2

Gestion des fenêtres

La technologie WinForms permet de réaliser une application MDI[14] très simplement. Dans ce type d'application la fenêtre principale peut contenir en son sein plusieurs fenêtres filles. La classe *Form* représente une fenêtre et dispose des propriétés suivantes :

- **IsMdiContainer** : cette propriété de type booléen transforme la fenêtre en parent si la valeur est à vrai. Cela affecte la couleur de fond et active la prise en charge du défilement du contenu.
- **MdiParent** : doit contenir une référence de fenêtre parent pour intégrer la fenêtre à l'intérieur du parent. La disposition est restreinte à la zone intérieure du parent.

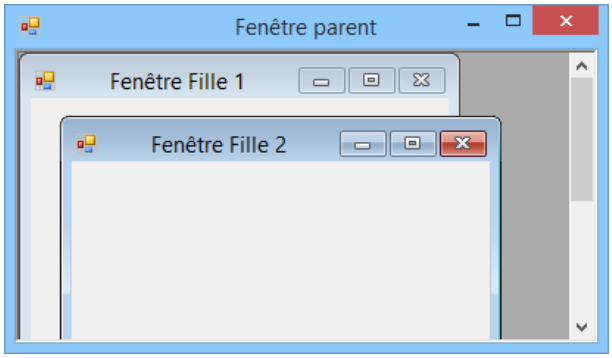


Figure 38 - Modèle d'application MDI

La présentation retenue dans la maquette est assez proche de ce principe. Les fenêtres filles sont disposées dans des zones internes d'onglets. La bibliothèque **DockPanelSuite** fournit une gestion du positionnement des fenêtres filles qui correspond parfaitement à la disposition voulue. Afin de bénéficier des fonctionnalités de positionnement dans toutes les fenêtres de l'application, j'ai créé une classe Fiche qui hérite de *DockContent*.

[14] Multi-Document Interface.

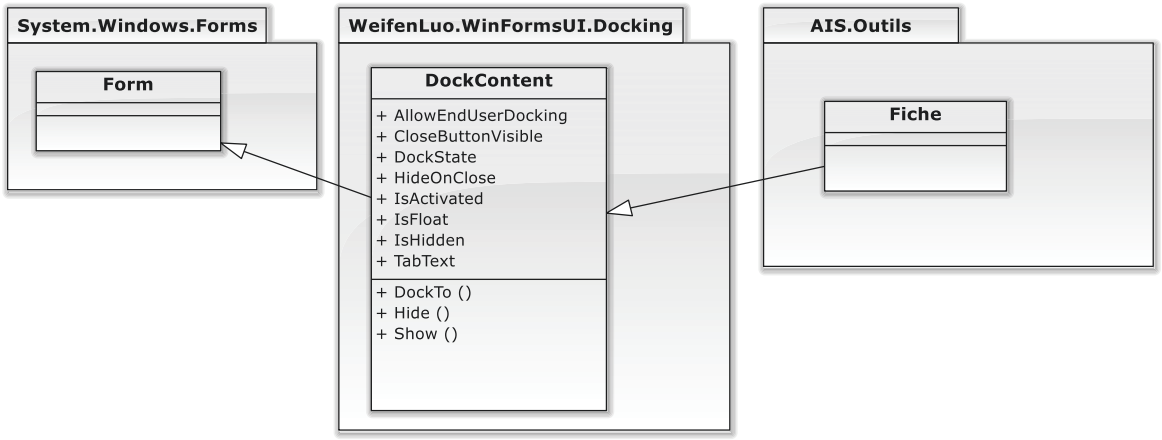


Figure 39 - Diagramme de classe simplifié d'une Fiche

Une fiche permettra d'éditer les champs correspondants à une ligne d'une table. Autrement dit, il peut y avoir autant d'instances différentes que de lignes présentes dans une table sans compter les instances destinées à ajouter de nouvelles lignes. On peut vouloir également des fenêtres qui doivent être en unique exemplaire. Comment savoir si une instance de fiche que l'on souhaite ouvrir n'est pas déjà ouverte ? Ce genre de question nous amène rapidement à l'idée qu'il faut un gestionnaire de fenêtre qui prenne en charge tous ces problèmes.

La classe *ManagerWindow* sera chargée de traiter toutes les demandes d'ouverture d'écrans. Pour y parvenir elle doit maintenir une liste des écrans ouverts avec pour chacun un élément d'identification.

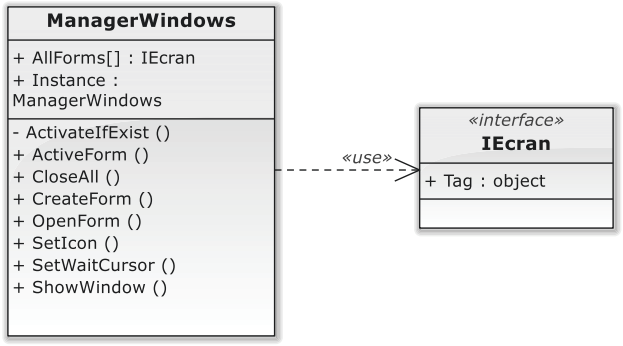


Figure 40 - Diagramme de classe du gestionnaire de fenêtres

- **AllForms** : tableau contenant la liste des références d'écrans ouverts. Suite à la fermeture de l'un d'entre eux le gestionnaire activera l'écran précédent dans la

liste. On obtient ainsi un effet logique lorsqu'un enchainement d'ouverture d'écran se produit et que l'on commence par en fermer un.

- **Instance** : propriété *static*[15] qui permet l'accès au gestionnaire. Sa valeur est affectée par le constructeur de la classe *ManagerWindows*. On garantit avec cette technique l'unicité du gestionnaire.
- **ActiveForm** : Sélectionne l'onglet pour un écran déjà ouvert.
- **CloseAll** : Fermeture de tous les écrans ouverts.
- **CreateForm** : création d'une nouvelle instance en fonction des informations d'identification.
- **OpenForm** : Recherche si l'identification de l'écran correspond à un écran déjà ouvert. Si c'est le cas l'onglet correspondant est sélectionné via la méthode ActiveForm. Sinon la méthode CreateForm est appelée.
- **SetIcon** : change l'icône de l'onglet.
- **SetWaitCursor** : affiche un icône sablier. En **WinForms** c'est la fenêtre principale qui doit activer cette fonctionnalité et le *ManagerWindows* est le seul à disposer d'une référence à cette fenêtre.
- **ShowWindow** : affiche la fenêtre fille grâce à la méthode DockContent.Activate() de l'écran. Ce qui implique que tous les écrans héritent de *DockContent*.

Pour identifier les écrans, le contenu de la propriété Tag est simplement une chaîne de caractères de la forme CODEECRAN_VALEUR. Pour des fiches permettant de modifier la table Analyse on utilisera des identifiants comme :

- « ANA_0 » : pour demander une fenêtre en mode ajout
- « ANA_1 » : pour demander l'ouverture de la fiche correspondante à la ligne de clé de valeur 1

Le menu principal utilise « MENU_ FIXE». Pour des écrans dont chaque ouverture doit donner lieu à une instance différente, on doit utiliser des GUID ce qui garantit l'unicité des identifiants. Ils se présentent sous la forme d'une série de lettres et de chiffres, ce qui donnera par exemple :

[15] Un élément static dans une classe est un élément commun à toutes les instances de cette classe.

ECRTJRSDIF_b04c1615-c042-44be-92b9-3c3893a3a449

Cette codification est strictement interne au gestionnaire de fenêtre et vise à retrouver rapidement une fenêtre déjà ouverte. Le développeur a besoin d'identifier les écrans mais également les actions associées aux écrans ou les actions personnalisées. Pour implémenter les différentes classes de la Figure 41 j'ai ajouté la table « Déclaration ». C'est une anticipation sur la possibilité de paramétrer les écrans et les actions sans modifier le code source de l'application.

Tableau 6 - Structure de la table _Dec

Champ	Type	Description
Dec	bigint	Clé primaire avec numérotation automatique
DecMod	uniqueidentifier	Guid correspondant au module
DecCod	nvarchar(50)	Code
DecDes	nvarchar(50)	Description pour le module de configuration
DecTyp	char(1)	E = Ecran A = Action
DecTab	nvarchar(50)	Nom de la table pour les écrans de recherche et les fiches
DecKey	nvarchar(50)	Nom de la clé primaire pour les écrans et les fiches
DecLnk	nvarchar(50)	Nom de la colonne qui servira de lien hypertexte dans les écrans de recherche
DecCls	nvarchar(250)	Nom de la classe de personnalisation du comportement d'un écran
DecActSel		Code de l'action à déclencher lors du clic de la colonne désignée par KeyLnk dans un écran de recherche. Il doit y avoir une action de type ouverture de fiche.

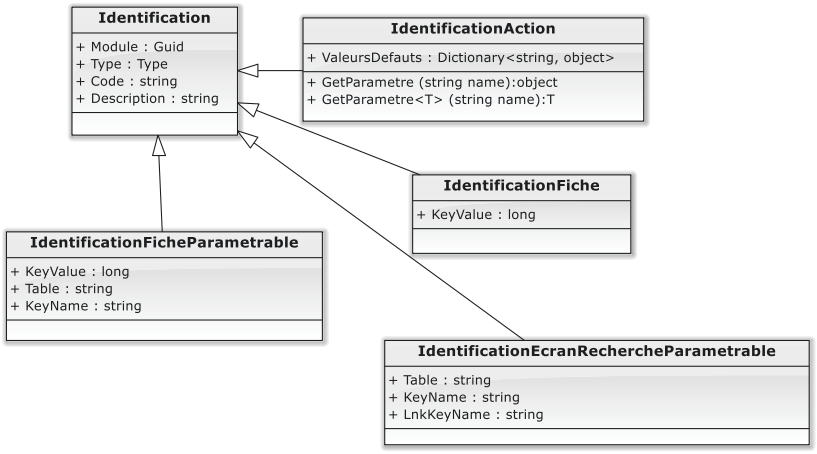


Figure 41 - Diagramme des classes Identification

Toutes les fonctions du gestionnaire de fenêtre utilisent des paramètres de type Identification (ou dérivés).

Modes de démarrage et modules

La bibliothèque d'outils doit permettre à terme de modifier l'application elle-même. Cela signifie qu'elle doit disposer d'un mode de démarrage qui donne accès à des écrans de paramétrage. L'idée est de décomposer une application en modules ce qui permet d'intégrer plusieurs modes de démarrage dans la même application.

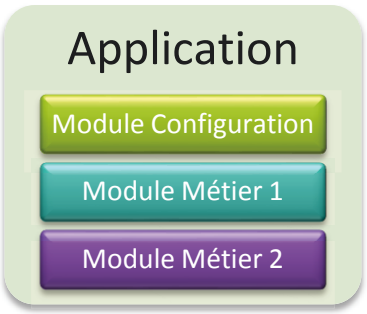


Figure 42 - Composition d'une application en modules

A chaque module sera associé un ensemble d'écrans et de traitements. Le comportement des modules est défini par la classe *Module*. La création d'un module nécessite la création d'une classe qui hérite de Module. Il faut surcharger les propriétés virtuelles **ID** et **Description**. Pour définir les écrans et les actions disponibles, le développeur peut utiliser les méthodes protégées **DeclareAction()**, **DeclareEcran()**, **DeclareEcranParametrable()** et **DeclareEcranRechercheParametrable()**.

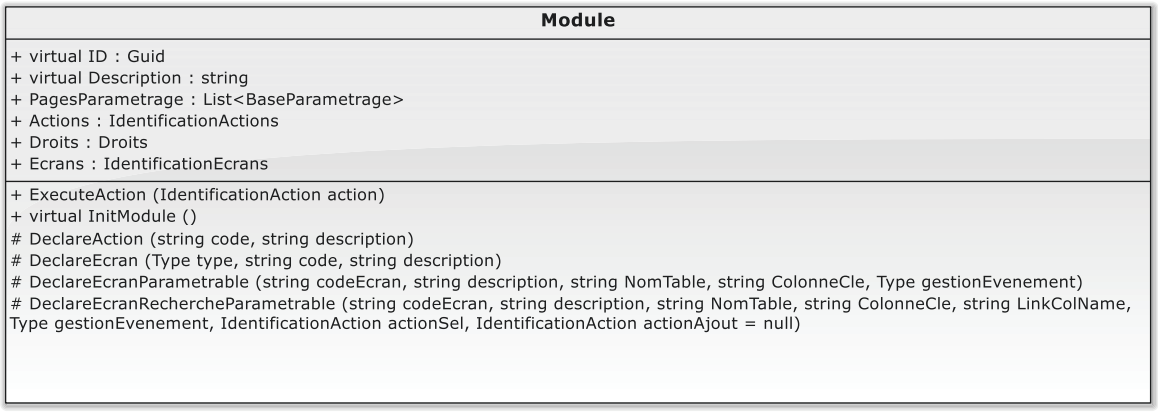


Figure 43 - Diagramme de classe Module

Mode Configuration

L'activation de ce mode se fait par l'utilisation d'un paramètre de la ligne de commande (/config) ou par le maintien de la touche shift pendant le chargement du programme.

Dans ce mode, l'application s'ouvre sur un écran composé de 2 parties séparées verticalement. La première est un *TreeView* qui présente les différents éléments paramétrables de l'application. La sélection d'un élément entraine l'apparition dans la partie de droite soit des éléments de niveaux inférieurs soit du détail de l'élément. Les panneaux restent chargés en mémoire pour permettre une navigation entre les éléments du *TreeView*. L'application des modifications est déclenchée par un bouton d'enregistrement situé en haut de la fenêtre.

Pour la mise en œuvre de ce fonctionnement j'ai créé les classes présentées sur la Figure 44.

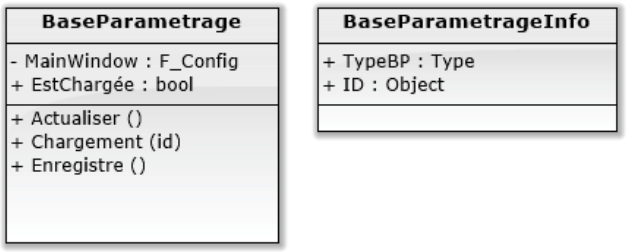


Figure 44 - Classes pour l'écran de configuration

Chaque panneau est une classe qui hérite de *BaseParametrage*. Il s'agit d'un composant de type *UserControl* qui met à disposition 3 méthodes virtuelles à surcharger pour implémenter la logique de l'écran de paramétrage. On trouve les méthodes suivantes :

Chargement : cette méthode est appelée lors de la création du panneau afin d'initialiser les données correspondantes.

Actualiser : cette méthode est appelée pour effectuer un rechargement des données après un enregistrement. En effet, les données peuvent avoir été altérées suite à l'enregistrement effectué par d'autres panneaux.

Enregistre : déclenche l'enregistrement des modifications. C'est à chaque panneau de décider s'il y a des actions à déclencher.

Le pilotage par l'écran de configuration permet de simplifier le développement de chaque panneau qui doit uniquement se concentrer sur ses informations spécifiques. Pour ne pas ralentir le chargement du mode configuration, les panneaux sont chargés uniquement lorsqu'une entrée du *TreeView* est sélectionnée. Pour arriver à ce mode de

fonctionnement, lors de l'initialisation du *TreeView*, les nœuds sont associés à une instance de *BaseParametrageInfo* afin de décrire quel est l'écran à instancier lors de la première sélection du nœud. La référence stockée dans la propriété Tag du nœud sera remplacée par celle du panneau correspondant au moment de sa création.

A ce stade l'arborescence ne contient que l'entrée correspondante au panneau qui permet d'éditer les informations d'accès à la base de données. Au fur et à mesure de l'ajout de fonctionnalités, des entrées supplémentaires seront ajoutées.

Lors de la première connexion seul l'outil de sélection de base de données est accessible puisque toutes les informations sont stockées dans la base de données.

Mode Utilisateur

Le mode de démarrage utilisateur consiste à ouvrir l'écran principal et le menu si l'authentification de l'utilisateur réussit.

Mode réseau

Le mode réseau se différencie du mode utilisateur par le démarrage d'une connexion à un processus serveur destiné à assurer la communication des notifications entre les instances de l'application sur le réseau local.

3.5.2. Mêlée 2

3.5.2.1. Authentification

L'authentification d'un utilisateur est une fonctionnalité de base pour toute application de gestion. Cette pratique vise à introduire les notions de non répudiation et d'imputabilité. La première empêche l'utilisateur de nier avoir commis une action non autorisée alors qu'elle a été détectée par le système. La seconde permet simplement de déterminer de façon sûre quelles sont les actions licites ou non pour cet utilisateur.

Il existe de nombreuses méthodes d'authentification mais on peut les classer parmi les niveaux suivants :

- **simple** : un seul élément est exploité. Il s'agit par exemple du mot de passe personnel de l'utilisateur ou d'un élément secret connu par l'ensemble des utilisateurs.
- **forte** : plusieurs éléments sont demandés. Ils sont plus complexes que pour les authentifications simples. Il peut s'agir de certificats numériques ou de protocoles avec un cryptage fort.
- **unique** : utilisation d'une seule authentification (de préférence de type fort) pour accéder à plusieurs applications informatiques. On peut citer l'authentification NTLM qui permet à un utilisateur Windows de s'identifier une seule fois à l'ouverture de sa session utilisateur et ainsi d'être authentifié auprès de toutes les machines d'un réseau Windows appartenant au même domaine de sécurité.

Dans le cadre de la bibliothèque d'outils, il faut déterminer quelles sont les informations nécessaires au paramétrage du système d'authentification. Il faut une table utilisateur (_Uti) à mettre en relation avec une table qui indiquera quels seront les modules, les écrans et les actions accessibles.

Puisque l'on ne peut pas présumer de l'environnement de sécurité dans lequel l'application sera exécutée, il faut prévoir un mode d'identification simple, pour les cas où la sécurité n'est pas un élément très important, et un mode de sécurité fort. En choisissant l'authentification intégrée à Windows pour le mode fort on peut définir les champs nécessaires dans la table des utilisateurs.

Champ	Description	Type de données	Obligatoire
Uti	N° interne non significatif	bigint	Oui
UtiNom	Nom de l'utilisateur	nvarchar(50)	Oui
UtiPas	Mot de passe si authentification simple. On stocke ici un hash MD5 pour ne pas enregistrer le mot de passe en clair	nvarchar(50)	Non
UtiNT	Nom de l'utilisateur précédé du nom de domaine pour l'authentification forte.	nvarchar(50)	Non

L'authentification par le nom d'utilisateur Windows peut fonctionner même dans le cas où les machines ne font pas partie d'un domaine de sécurité unique. En effet, si

l'ordinateur n'est pas membre d'un domaine, l'utilisateur est reconnu sous la forme NOM_MACHINE\Nom_Utilisateur.

On aura besoin plus tard de gérer la sécurité. Et pour éviter un paramétrage individuel des droits on peut prévoir la notion de groupe d'utilisateurs. Un nom et une description suffisent pour définir un groupe. Un utilisateur pouvant appartenir à plusieurs groupes, on peut stocker cette configuration avec les 3 tables suivantes :

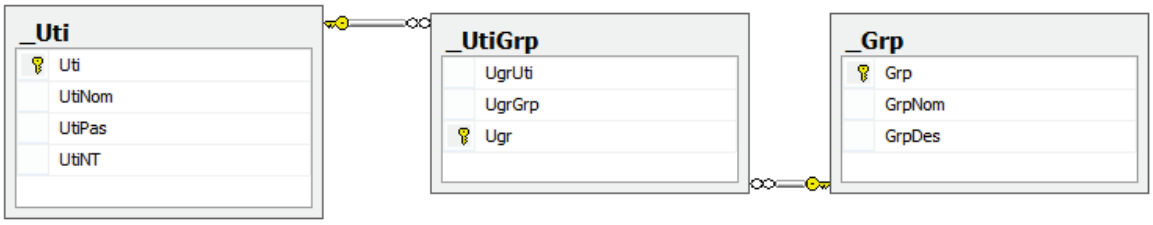


Figure 45 - Tables de configuration des utilisateurs et des groupes

La mise en œuvre de la configuration de l'authentification se résume à très peu de cas d'utilisation.

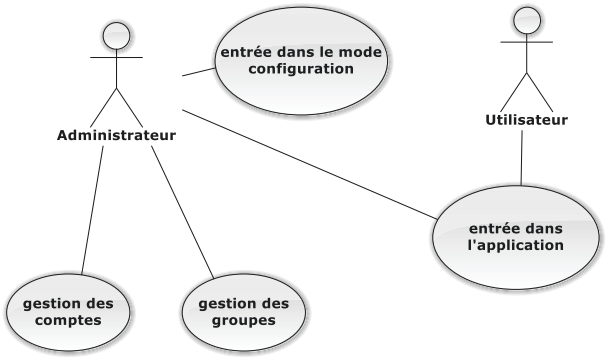


Figure 46 - Cas d'utilisation du mode Configuration

- Un administrateur a besoin de rentrer dans le mode de configuration.
- Un administrateur a besoin de créer, modifier ou supprimer des groupes. Les groupes sont des regroupements d'utilisateurs. Il doit pouvoir modifier les utilisateurs affectés à chaque groupe.
- Un administrateur a besoin de créer, modifier ou supprimer des comptes utilisateurs.
- Un utilisateur, tout comme un administrateur, a besoin de rentrer dans l'application en mode utilisateur.

Pour sécuriser entièrement l'application il faudra trouver un moyen d'autoriser uniquement l'administrateur à entrer dans le mode paramétrage. Il devra indiquer quels seront les comptes Windows valides pour entrer dans l'application en mode utilisateur.

Dans le module Configuration, l'accès aux panneaux de paramétrage des utilisateurs et des groupes se fera par des *TreeNodes* sur la partie gauche conformément au design choisi.



Figure 47 - Gestion des utilisateurs dans le menu de configuration

Configuration des Utilisateurs

Le panneau est constitué d'un *TreeView* avec au premier niveau les utilisateurs. Un deuxième niveau indique quels sont les groupes dont l'utilisateur est membre.

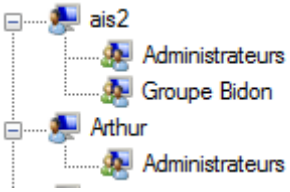


Figure 48 - Affichage des utilisateurs et des groupes

Toutes les actions se font par un menu contextuel. Ainsi un clic droit sur un élément du premier niveau donne les actions suivantes dans le menu contextuel :

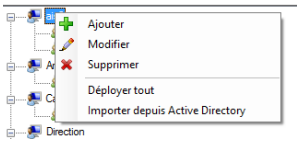


Figure 49 - Menu contextuel de gestion des utilisateurs

Les choix **Ajouter** et **Modifier** déclenchent l'ouverture d'une fenêtre d'édition de l'utilisateur. On y trouve dans un premier onglet des zones de saisie des champs relatifs à l'utilisateur. En cas d'utilisation de l'authentification Windows, la zone *Mot de passe* est désactivée. Dans le cas contraire, c'est la zone *Authentification Windows* qui n'est pas disponible.

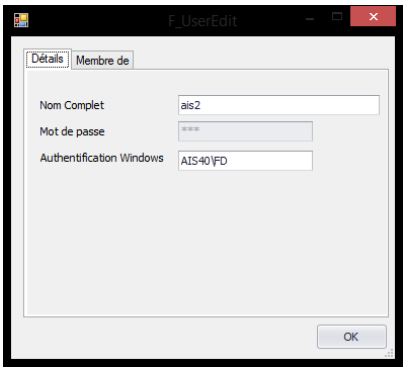


Figure 50 - Ecran de saisie d'un utilisateur

Le deuxième onglet permet d'affecter l'utilisateur aux différents groupes existants en cochant les groupes correspondants.

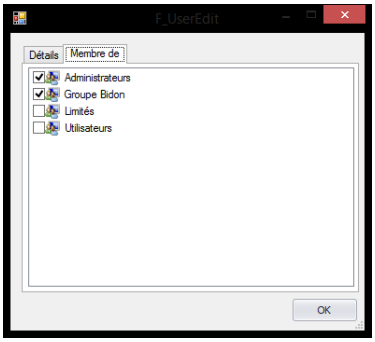


Figure 51 - Ecran d'affectation de groupe

Le choix **Supprimer** permet de supprimer l'utilisateur. L'opération échouera si l'utilisateur est référencé dans d'autres tables comme celle qui définit les droits. En revanche la table d'association avec les groupes est nettoyée automatiquement. On peut noter ici l'utilisation du raccourci clavier avec la touche **Suppr** pour rendre l'opération plus simple.

Le choix **Déployer Tout** permet d'afficher les 2 niveaux pour l'ensemble de l'arborescence. Cette option est destinée à améliorer la vision globale du paramétrage.

Un clic droit sur le deuxième niveau retire les choix Modifier/Supprimer et Ajouter/ Enlever du groupe.

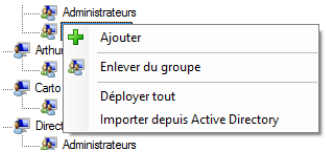


Figure 52 - Menu contextuel de gestion de groupe (écran utilisateur)

Le choix **Importer depuis Active Directory** permet d'ouvrir une fenêtre utilitaire. Celle-ci permet d'importer les utilisateurs définis au niveau du serveur du domaine Windows. Cette fonctionnalité est un gain de temps pour la mise en place de l'application dans le cas où il existe des dizaines d'utilisateurs. Le code nécessaire à cette opération est présenté dans l'Annexe 3.

Configuration des Groupes

La gestion des groupes est identique à celles des utilisateurs. On retrouve dans le panneau un *TreeView* avec au premier niveau les groupes et en dessous les utilisateurs.

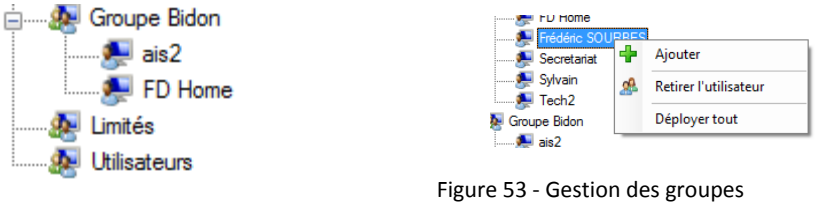


Figure 53 - Gestion des groupes

La fenêtre d'édition est également similaire à celle destinée à l'édition des utilisateurs.

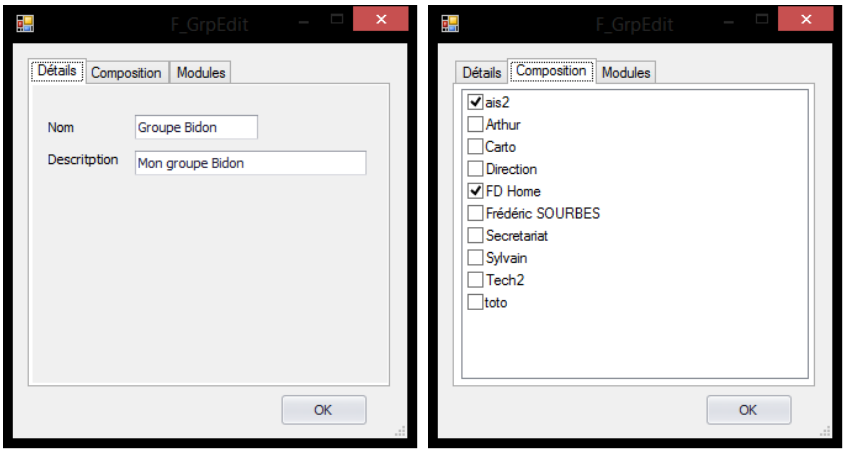


Figure 54 - Ecran de saisie de groupe

3.5.2.2. Identifications, droits et modules

Une application réalisée avec la bibliothèque d'outils comportera des éléments avec des actions standardisées. On peut imaginer facilement les actions. Par exemple l'élément Fiche dispose des actions Ouvrir, Modifier et Supprimer. Un écran de recherche dispose uniquement de l'action Ouvrir et un module dispose de l'action Chargement. En plus de ces actions prédéfinies, il faut laisser la possibilité de déclarer des actions à caractère métier. On peut penser à l'autorisation de déclencher un traitement.

Il se pose maintenant la question de l'identification et de la déclaration des différents éléments et actions. La première réflexion à ce sujet a été de lister les différents droits possibles. Le tableau suivant correspond à la liste des droits standards avec les objets sur lesquels ils sont applicables.

Tableau 7 - Droits standards enregistrés

Nom du droit	S'applique à	Description
Chargement	Module	Correspond à l'accès au module
Ajouter	Fiche	Correspond à l'ouverture d'une fiche en mode ajout
Modifier		Correspond à l'entrée en mode modification sur une fiche
Supprimer		Correspond à la suppression d'une fiche
Exécuter	Action	Correspond à l'exécution d'une action

Pour ne pas complexifier la gestion des droits nous sommes partis sur le principe : tout ce qui n'est pas interdit est autorisé. Ainsi seules les restrictions seront enregistrées et donc en cas de conflit, la restriction sera prioritaire.

Le tableau suivant indique quelles sont les actions standards par type d'écran nécessaires pour le gestionnaire de fenêtres.

Tableau 8 - Actions standards par type d'écran

Type	Exemple de code	Codification action	Codification suffixe	Exemple
Fiche	FANA	Ouverture	_OPEN	FANA_OPEN
		Insertion	_ADD	FANA_ADD
Ecran de recherche	LANA	Ouverture	_FIND	LANA_FIND

Après avoir identifié les éléments nécessaires à la description des écrans et des droits, les tables de paramétrages ont pu être créées. On dispose des tables suivantes en plus des tables _Grp, _Uti et _UtiGrp définies précédemment :

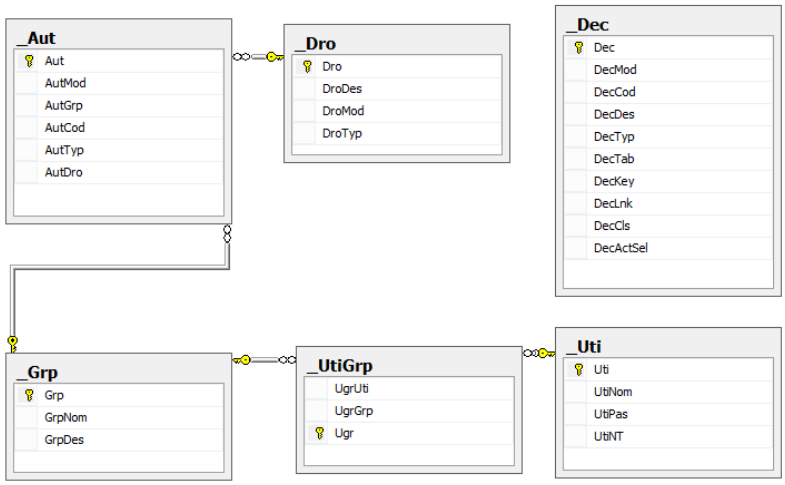


Figure 55 - Tables de déclaration des écrans et des autorisations

- **_Aut** : listes des Autorisations. En plus des clés étrangères vers les tables _Dro et _Grp, chaque ligne est associée à un code de module et un type (« A » pour action et « E » pour écran)
- **_Dro** : liste des Droits. Les droits standards n'ont pas de valeur de code de module.
- **_Dec** : déclarations

Il n'est pas prévu dans cette version des outils un éditeur d'écran. Les tables devront être modifiées par l'intermédiaire de l'éditeur de table dans l'outil Microsoft SQL Management Studio. Seule la partie de définition des droits sera implémentée dans le module de configuration. Les écrans déclarés dans la table (ou par code) apparaitront sous forme d'entrée dans la partie Sécurité présente dans le paramétrage d'un module. L'autorisation sera définie simplement en cochant ou décochant un groupe en dessous de l'élément concerné. La Figure 56 montre l'exemple des droits Ajouter et Modifier qui sont refusés à tous les groupes sauf le groupe administrateur pour l'écran « Configuration » appartenant au module « Pisciculture ».

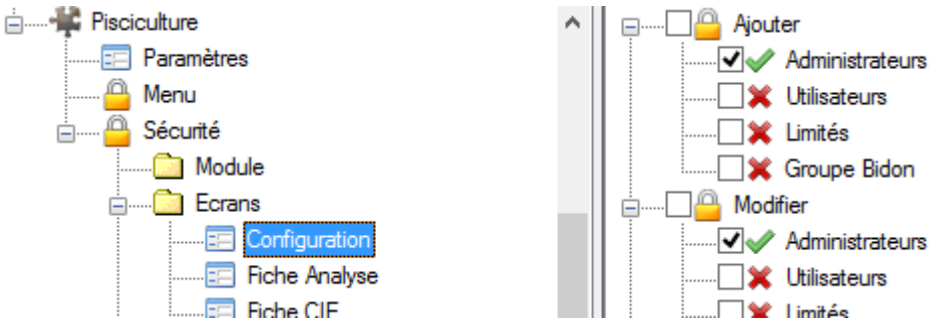


Figure 56 - Exemple de définition d'autorisations

Le modèle objet correspondant aux tables de paramétrage est réalisé selon le diagramme de la Figure 57.

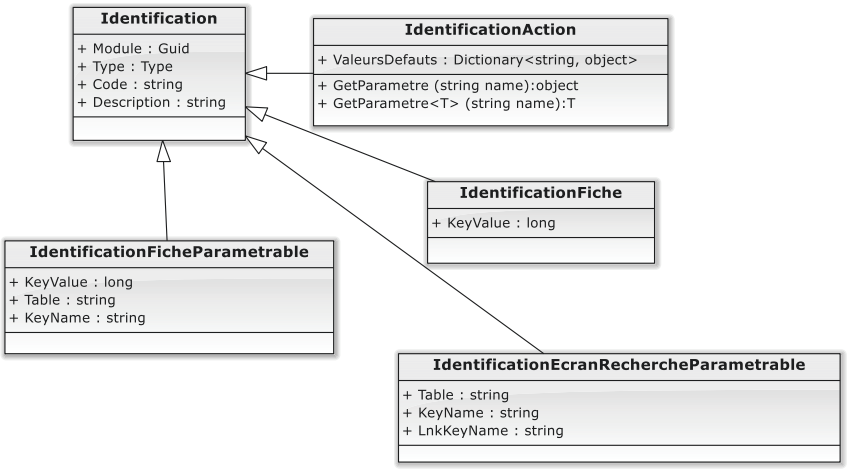


Figure 57 - Diagramme de classes des identifications d'écrans

3.5.2.3. Menu Principal

Le menu principal a pour objectif de permettre l'ouverture des écrans ou le déclenchement de traitements. Dans les 2 cas il s'agit d'exécuter une action suite au clic sur un lien hypertexte. La structure du menu a été figée sous la forme de blocs pouvant contenir un nombre variable d'actions. Chaque bloc dispose d'un titre et d'une image. Chaque action est également associée à un libellé et une image. La disposition générale est sous forme de grille. Le résultat souhaité est de la forme



Figure 58 - Schéma de l'écran de menu

La conception du modèle objet est très simple. Une classe *Menu* contient une collection de *BlocMenu*. Chaque BlocMenu contient une collection de BlocMenuAction.

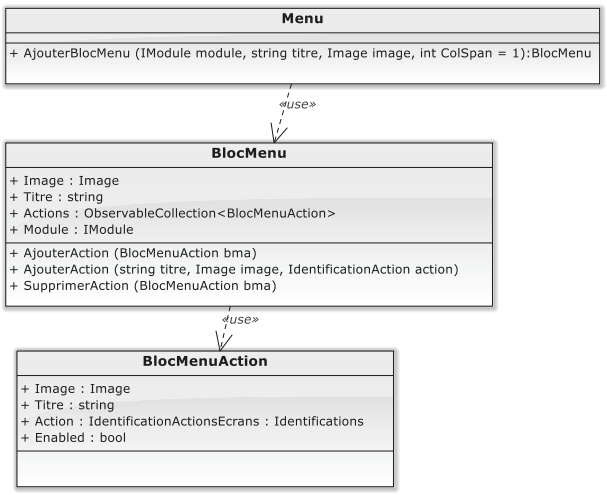


Figure 59 - Diagramme de classes de gestion du menu principal

Les classes intègrent le code nécessaire pour traduire le menu dans une fenêtre de type *DockContent*. Contrairement aux autres fenêtres, il n'est pas possible de fermer le menu principal.

3.5.3. Mélée 3

Le troisième sprint est orienté vers le choix et la mise en place de la partie interface utilisateur. Microsoft met à disposition du programmeur avec chacune de ses technologies d'interface utilisateur un ensemble de composants basiques. Mais les contrôles utilisateur ayant un comportement avancé sont laissés aux éditeurs tiers. On trouve dans le pack de contrôles DevExpress des zones de texte, cases à cocher, listes déroulantes et tout ce qui permet de réaliser une fiche avec un aspect professionnel.

3.5.3.1. Grille

Dans une application de gestion de données, le contrôle grille doit être performant et disposer d'une grande capacité de personnalisation. J'ai opté pour la célèbre grille open source appelée SourceGrid entièrement écrite en C#. J'ai dû effectuer des modifications du code source afin d'intégrer les éditeurs du pack de contrôles DevExpress comme éditeurs de champs. Dans la bibliothèque outils, tous les contrôles utilisateurs doivent supporter un mode consultation et un mode modification. J'ai donc créé un nouveau contrôle utilisateur. Il intègre une grille afin d'inclure des fonctionnalités complémentaires et de permettre l'exploitation de la grille avec un minimum de lignes de code pour le développeur.

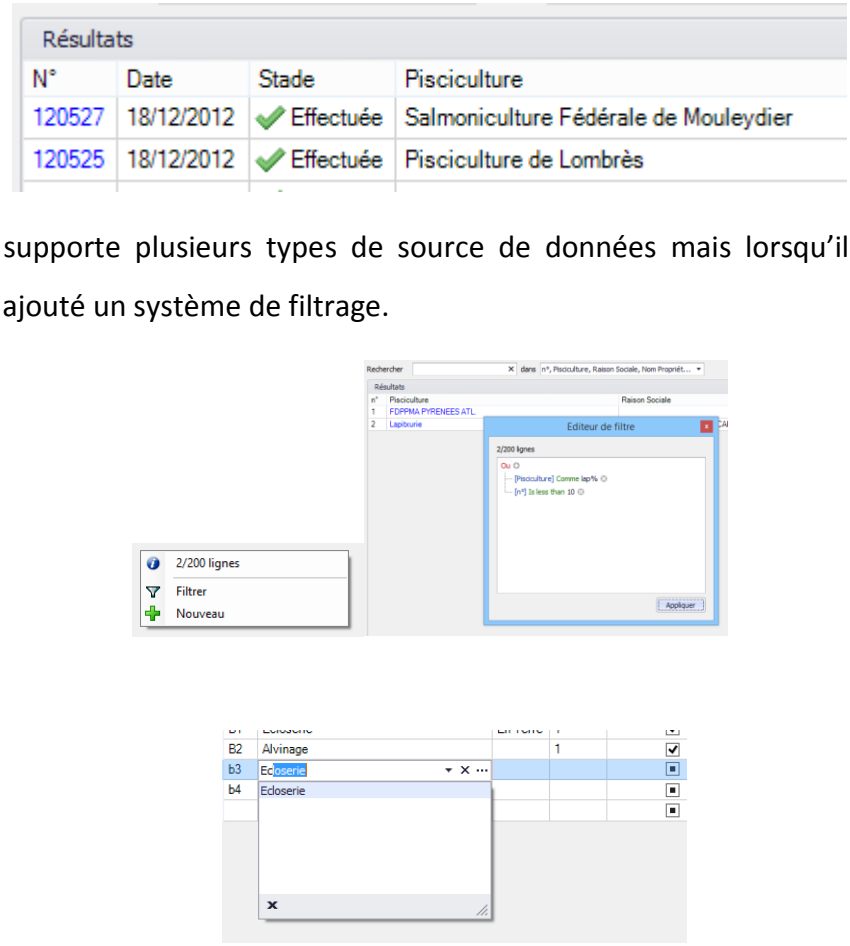


Figure 60 - Contrôle Grille personnalisé

Le passage en mode modification entraine l'apparition d'une ligne de saisie en bas de la grille ainsi qu'une colonne à gauche pour bien identifier l'état de la ligne courante. Une icône de crayon signifie que la ligne est en cours de modification. Il est possible d'activer individuellement l'ajout, la modification ou la suppression de ligne.

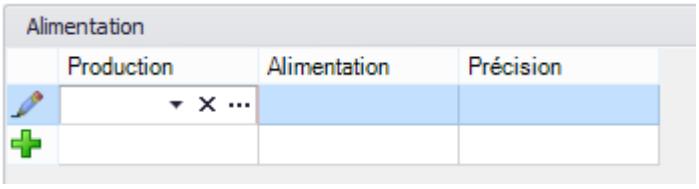


Figure 61 - Mode édition du contrôle Grille

3.5.3.2. Ecran de recherche

L'écran de recherche est le premier écran paramétrable implémenté. Il a été choisi car le paramétrage est limité. En effet, la partie graphique est fixe et le nombre d'éléments variables est faible. L'objectif est de fournir la source de données de la grille de recherche ainsi que l'action qui se produit lors du clic sur un élément de la colonne désignée comme de type lien hypertexte. La personnalisation des colonnes doit rester possible comme sur n'importe quelle grille. L'implémentation de l'écran doit donc permettre au développeur de fournir simplement les informations et masquer complètement les problèmes techniques comme par exemple la dégradation des performances avec de grands volumes de données. Pour répondre à ces critères, les classes suivantes sont mises en œuvre :

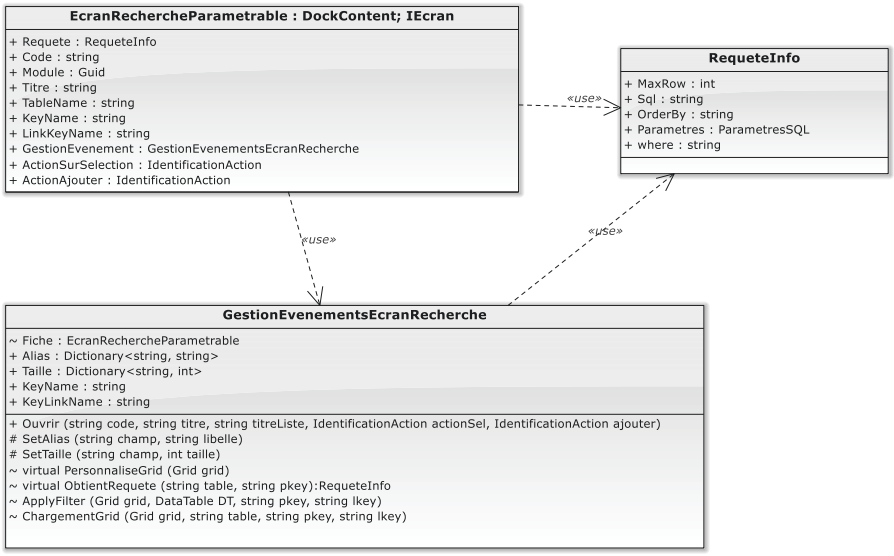


Figure 62 - Diagramme de classes de gestion d'écran de recherche

L'extrait de code suivant présente la séquence permettant d'ouvrir un écran de recherche suite au déclenchement d'une action de type *IdentificationEcran RechercheParametrable*.

```
public void ExecuteAction(IdentificationAction action)
{
    ManagerWindows.Instance.SetWaitCursor(true);
    foreach (IModule m in _modules)
        if (m.ID.Equals(action.Module))
        {
            // code supprimé  ( traitement autres action et contrôle des droits )
            if ( action.Code.EndsWith("_FIND"))
            {
                string codeEcran = action.Code.Split("_".ToCharArray())[0];
                if (m.Ecrans[codeEcran] != null)
                {
                    // creation de la classe
                    if (action.Type == null) // Pas de classe définie spécifiquement
                        action.Type = typeof(GestionEvenementsEcranRecherche);
                    GestionEvenementsEcranRecherche gest = (GestionEvenementsEcranRecherche)Activator.CreateInstance(action.Type);


                    gest.Ouvrir(codeEcran, m.Ecrans[codeEcran].Description, "Résultats",
                        ((IdentificationEcranRechercheParametrable)m.Ecrans[codeEcran]).ActionSurSelection,
                        ((IdentificationEcranRechercheParametrable)m.Ecrans[codeEcran]).ActionAjouter);
                    ManagerWindows.Instance.SetWaitCursor(false);
                    return;
                }
            }
            m.ExecuteAction(action);
            break;
        }
    ManagerWindows.Instance.SetWaitCursor(false);
}
```

Figure 63 - Extrait de code ExecuteAction() du gestionnaire de module

Le code de cette action se termine selon notre convention par le suffixe « _FIND » ce qui permet d'en déduire le code de l'écran correspondant. L'identification de l'écran, retrouvée grâce au code, peut être associée à une classe destinée à personnaliser son comportement. Pour un écran de recherche le type associé doit être une classe *GestionnaireEcranRecherche* ou dérivé. La création de l'instance se fait dynamiquement ce qui autorisera plus tard d'écrire les classes de personnalisation dans des scripts externes.

En l'absence de classe de personnalisation spécifique, la version de base est automatiquement créée. Elle se contente de sélectionner toutes les colonnes disponibles dans la table désignée par la propriété **TableName** de la classe *IdentificationEcranRecherche*. Les colonnes sont redimensionnées en fonction du contenu des lignes. Par défaut, un maximum de 200 lignes est retourné par la requête de sélection. Pour obtenir un résultat plus précis, il faut effectuer une recherche en saisissant dans la zone de recherche. A chaque modification de cette zone une première recherche est effectuée localement grâce au composant de filtrage interne à la grille. Puis une recherche dans la base est lancée en parallèle dans un autre Thread[16]. Ainsi le filtrage peut s'effectuer sur un ensemble très important de données sans bloquer l'application. Une fois les lignes

[16] Un Thread est une séquence de code exécutée en parallèle de l'exécution du code principal d'une application. Il permet d'effectuer des traitements longs sans visuellement bloquer le programme.

déterminées par la requête sur le serveur (toujours limitées à 200) on décide quel ensemble de données est le plus pertinent. Si le filtre côté serveur renvoie plus de lignes que le filtrage local, c'est que des lignes non présentes dans le jeu initial correspondent au filtre. Pour obtenir un filtrage presque instantané en apparence sur des tables contenant plusieurs dizaines de millier de lignes, il suffit de diminuer le nombre de lignes retournées par requête.

Pour personnaliser l'écran de recherche il faut implémenter une classe qui hérite de *GestionEvenementsEcranRecherche* et surcharger les fonctions virtuelles. L'annexe 7 présente des exemples de classes de personnalisation d'écran de recherche.

3.5.3.3. Contrôles avec libellé intégré

Dans un écran, chaque de zone de saisie doit disposer d'un texte qui explique la sémantique de la donnée. La construction des écrans avec des outils traditionnels impose de positionner individuellement un contrôle label à côté de la zone de saisie. Cette dichotomie a tout son sens aux yeux d'un informaticien puisqu'il ne s'agit effectivement pas des mêmes contrôles. Mais pour un utilisateur le libellé est indissociable de la zone de saisie. Pour simplifier cet état de fait, j'ai utilisé un UserControl appelé **LabelControl** composé comme suit :

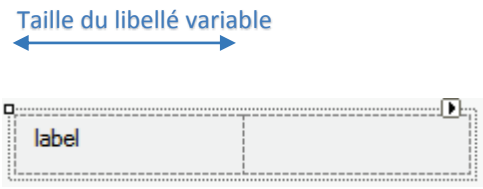


Figure 64 - Schéma d'une zone contrôle avec libellé

La classe **CustomControlWithLabel** générique permet une uniformisation des types de contrôles. On peut facilement créer des contrôles avec label avec un simple héritage et l'ajout de propriétés ou méthodes supplémentaires.

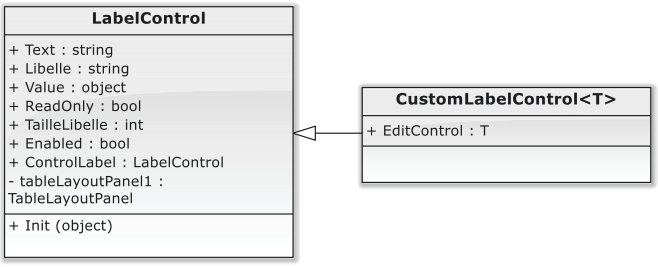


Figure 65 - Diagramme de classes pour les contrôles avec libellé intégré

Tableau 9 - Liste des classes avec libellé intégré

Classe	Type de base
LabelWithLabel	CustomLabelControl<LabelControl>
QuickGridWithLabel	CustomLabelControl<QuickGrid>
TextBoxWithLabel	CustomLabelControl<TextEdit>
UrlListWithLabel	CustomLabelControl<UrlList>
ButtonEditWithLabel	CustomLabelControl<ButtonEdit>
DateEditWithLabel	CustomLabelControl<DateEdit>
SpinEditWithLabel	CustomLabelControl<SpinEdit>
CheckEditWithLabel	CustomLabelControl<CheckEdit>
HyperLinkEditWithLabel	CustomLabelControl<HyperLinkEdit>
MemoEditWithLabel	CustomLabelControl<MemoEdit>
LookUpEditWithLabel	CustomLabelControl<LookUpEdit>
ImageComboBoxEditWithLabel	CustomLabelControl<ImageComboBoxEdit>
TimeEditWithLabel	CustomLabelControl<TimeEdit>
ComboBoxEditWithLabel	CustomLabelControl<ComboBoxEdit>
CheckedComboBoxEditWithLabel	CustomLabelControl<CheckedComboBoxEdit>
FileBrowserWithLabel	CustomLabelControl<HyperLinkEdit>

3.5.3.4. Fiche

La fiche permet la saisie de données. Elle sera constituée d'un ensemble de contrôles. Chacun d'eux sera connecté à un champ de l'unique ligne servant de source de données. Pour répondre aux critères définis par la maquette et que le programmeur n'ai pas de code à réécrire à chaque écran, il faut réaliser un composant avec les fonctionnalités suivantes :

- liaison de données entre les contrôles et la source de données
- opérations de base : chargement, enregistrement ou suppression
- gestion d'état (mode de visualisation ou modification)
- prise en compte des valeurs obligatoires et des valeurs par défauts (selon le schéma de la base de données)
- construction dynamique de l'écran selon une structure descriptive comprenant les champs et le positionnement des éléments.

Nous allons détailler comment chacune de ses fonctionnalités a été traitée.

3.5.3.4.1. Liaison de données

L'application des modifications dans la base de données est prise en charge par la classe *DataTableEx* réalisée précédemment. Cependant, avant de déclencher une demande d'enregistrement, il faut que les valeurs saisies par l'utilisateur soient répercutées dans le *DataTableEx*. Pour mettre en place une telle liaison, il faut utiliser une technique dite de « Data Binding » fournie par **WinForms**. Elle permet de synchroniser automatiquement la valeur d'une source et d'une cible. Plusieurs types de sources sont supportés mais dans notre cas il s'agit d'un DataTable. La cible est un contrôle utilisateur compatible. C'est le cas de toutes les classes héritant de la classe *Control*. Chacun d'eux dispose d'une collection d'objets *Binding* nommée **DataBindings**. L'objet Binding mémorise un lien vers un objet source grâce à sa propriété **DataSource**. Il contient également un nom de propriété pour la source et un nom de propriété de la cible. Enfin, l'objet *Binding* permet d'indiquer par sa propriété **Current** quel est l'objet spécifique de la source à relier dans le cas où elle contient plusieurs éléments.

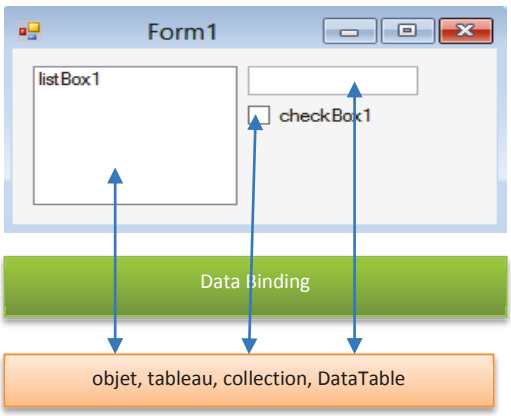


Figure 66 - Liaison de données

La liaison sera effectuée sur la propriété **Value** disponible sur les contrôles avec libellé intégré.

3.5.3.4.2. Opérations de base

Le contrôle utilisateur qui prend en charge l'ensemble du comportement d'une fiche ne nécessite que le nom de la table pour fonctionner et la valeur de la clé primaire de la

ligne concernée. Le type de la clé primaire est systématiquement *long* car les tables doivent respecter les contraintes définies en début de projet. Le fonctionnement imaginé lors de la réalisation de la maquette nous amène à implémenter la gestion d'état décrite sur la Figure 67.

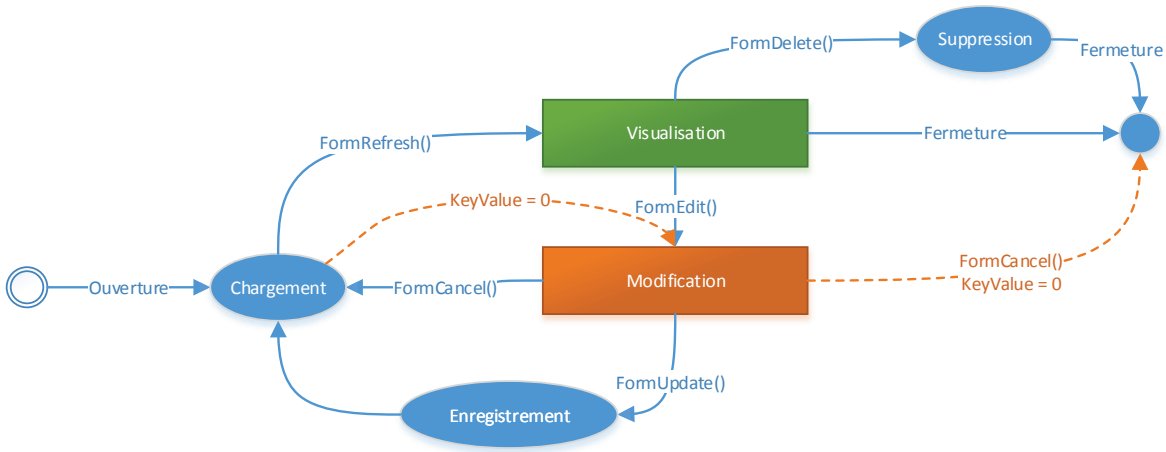
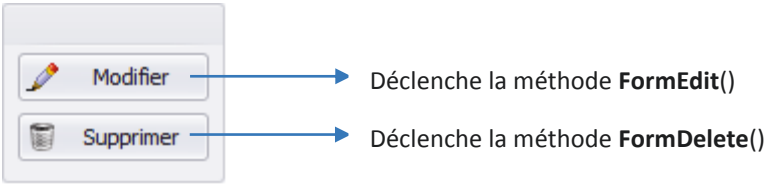


Figure 67 - Changement d'états d'une fiche

Les différents états de la fiche seront modifiés par un composant constitué d'un groupe de boutons. Selon le mode en cours, seules les options adéquates seront accessibles.

<u>En mode visualisation on aura les boutons suivants :</u>

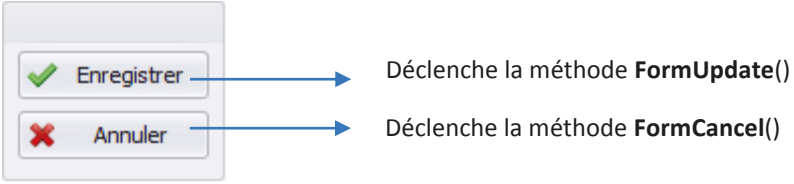


Déclenche la méthode **FormEdit**()

Déclenche la méthode **FormDelete**()

Tous les contrôles sont verrouillés dans ce mode. La propriété **ReadOnly** est passée à **true**. Si elle n'existe pas, le contrôle est désactivé via la propriété **Enabled**.

<u>En mode modification on aura :</u>



Déclenche la méthode **FormUpdate**()

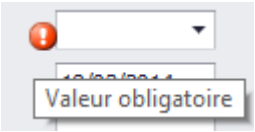
Déclenche la méthode **FormCancel**()

L'ouverture de l'écran déclenche la création des contrôles, le chargement initial de la source de données et la liaison entre eux. Le chargement est effectué par la méthode **FormRefresh**(). Elle exécute une requête très simple pour obtenir le *DataTableEx* contenant une seule ligne :

```
Select * from TableName where cle=@val
```

Le nom de la clé primaire est retrouvé automatiquement grâce aux métadonnées fournies par SQL Server. Si aucune ligne n'est retournée avec la requête, l'écran affecte 0 comme valeur de clé, ajoute une ligne vide à la source de données (pour permettre aux liaisons vers les contrôles de fonctionner correctement) et bascule l'écran en mode modification. Une annulation du mode modification conduira à la fermeture de l'écran.

L'enregistrement est géré par la méthode **FormUpdate**(). Avant d'envoyer les modifications vers la base de données, il y a une vérification des champs dont la valeur est obligatoire. En cas de valeur absente, la fiche est capable par l'intermédiaire des liaisons de retrouver le contrôle concerné et de le mettre en évidence avec une alerte visuelle comme le montre l'image suivante :



De plus, si le contrôle est de type libellé intégré, le libellé du contrôle est modifié dès le passage en mode modification pour apparaître en caractère gras. Cela évite à l'utilisateur de tenter un enregistrement pour savoir quelles sont les zones requises.

Si toutes les conditions sont réunies pour l'enregistrement, la méthode **FormUpdate**() débute une transaction. Ensuite elle provoque l'enregistrement. En cas d'insertion, la valeur de la clé est déterminée puis affectée à la propriété **KeyValue**. Un parcours de tous les contrôles détermine si certains d'entre eux doivent procéder à un enregistrement spécifique. C'est le cas par exemple d'une grille modifiable qui afficherait des informations d'une table connexe. Dans ce cas aussi, les métadonnées de SQL Server vont permettre de trouver la clé étrangère dans la table connexe et de mettre à jour automatiquement la clé étrangère dans les nouvelles lignes. Après un enregistrement réussi, il se produit un rechargement des données. En effet, il n'est pas exclu que les

données soient modifiées par des triggers[17]. Dans le cas où des modifications de la même ligne sont validées par d'autres utilisateurs pendant que l'on est en mode modification, il se produira une erreur d'accès concurrentiel lorsque l'on déclenchera l'enregistrement. Il faudra alors annuler et recommencer. Pour éviter cette situation, le gestionnaire de notifications développé plus tard sera utilisé.

3.5.3.4.3. Modèle déclaratif

Pour obtenir une construction de fiche simplifiée au maximum, le programmeur doit pouvoir décrire la fiche à partir des champs qu'il souhaite pouvoir modifier. Afin de ne pas imposer une composition figée des éléments, j'ai imaginé une structure hiérarchique de définition. La notion de composition étant un principe de base de **WinForms**, il sera très facile de traduire la structure de déclaration en composants de l'interface utilisateur. Le principe est de déterminer automatiquement les contrôles adaptés à chaque type de champs. Un ensemble de contrôles prédéterminé sera nécessaire pour améliorer la mise en page d'une fiche. On peut citer les tables, les groupes ou les onglets. Ces éléments conteneurs permettent de disposer les éléments sous forme de lignes et de colonnes. Pour positionner les éléments il suffit d'indiquer à quelle ligne ils appartiennent ainsi que leur hauteur et largeur s'ils doivent être étendus sur plusieurs lignes ou colonnes.

La structure de l'écran peut être définie par du code à l'exécution mais l'objectif est de stocker la définition dans des tables. Un éditeur d'écran réalisé plus tard n'aura plus qu'à alimenter ces tables.

[17] Les triggers sont des traitements à déclenchement automatique intégrés dans le moteur de base de données.

Tableau 10 - Structure de la table _EcrDef

Champ	Description	Type de donnée
Ede	Identifiant unique	bigint
EdeDes	Description ou libellé	nvarchar(250)
EdeTyp	Type d'élément	nvarchar(50)
EdeOrd	Ordre relatif au conteneur	Int
EdePar	Elément parent	bigint
EdeLig	Ligne relative au conteneur	int
EdeLigSpa	Nombre de lignes relatives au conteneur	int
EdeColSpa	Nombre de colonnes relatives au conteneur	int
EdeDec	Identifiant de l'écran	bigint
EdeImg	Nom de l'image	nvarchar(250)
EdeSizDes	Taille du libellé ou libellé par défaut pour un conteneur	int
EdeExpVis	Formule avec les noms de champs qui conditionne la visibilité de la zone	nvarchar(250)
EdeReqExp	Formule avec les noms de champs qui conditionne la valeur comme obligatoire	nvarchar(250)
EdeEnaExp	Formule avec les noms de champs qui conditionne si la zone est modifiable	nvarchar(250)
EdeFld	Nom du champ associé	nvarchar(50)
EdeCod	Code à usage du programmeur pour accéder à la zone si elle n'est pas associée à un champ.	nvarchar(50)
EdeFmt	Formule de formatage pour les nombres	nvarchar(50)
EdeDef	Valeur par défaut	nvarchar(50)
EdeLng	Longueur maximale	int
EdeEds	Code source de données pour une grille sous la forme « TABLE/Nom_table »	nvarchar(50)
EdeImgH	Hauteur de l'image	int
EdeImgW	Largeur de l'image	int
EdeAct	Code action sous la forme « ECRAN_OPEN »	nvarchar(50)
EdeAut	Indique si une grille est autonome. C'est-à-dire non soumise au mode d'édition courant d'une fiche.	bit
EdeCou	Couleur	nvarchar(50)
EdeMinH	Hauteur minimale. Aide pour la mise en forme d'une fiche afin d'éviter des grilles avec des hauteurs trop faibles.	int

Cette table est associée à la table de définition des colonnes pour les éléments de type conteneur.

Tableau 11 - Structure de la table de définition des colonnes _EcrCol

Champ	Description	Type de donnée
Eco	Identifiant	bigint
EcoGrp	Identifiant du conteneur	bigint
EcoCol	Numéro de la colonne	Int
EcoTai	Taille de la colonne. En % si la valeur est positive, en pixel si elle est négative	float

La Figure 68 présente une version simplifiée du diagramme des classes qui permettent de définir hiérarchiquement la structure d'un écran.



Figure 68 - Diagramme de classes des définitions d'éléments des écrans

3.5.3.4.4. Création effective des contrôles

La classe *FormBuilder* est chargée de la création des contrôles. Elle parcourt la hiérarchie en appelant la méthode **CreateControl**() de chaque élément écran. Un calcul automatique des tailles minimales de chaque libellé est effectué en fonction de la police d'affichage courante. Chaque contrôle associé à un champ est lié à un objet de contexte qui permettra de mettre en place la liaison de données. L'ensemble des contrôles est ensuite placé dans une zone associée aux données. Cette zone implémentée par la classe *DataBaseAera* correspond à une fiche. Elle peut cependant contenir un ensemble de sous zones dont chacune d'elle possède sa propre source de données et possède au moins une valeur commune avec sa zone parent.

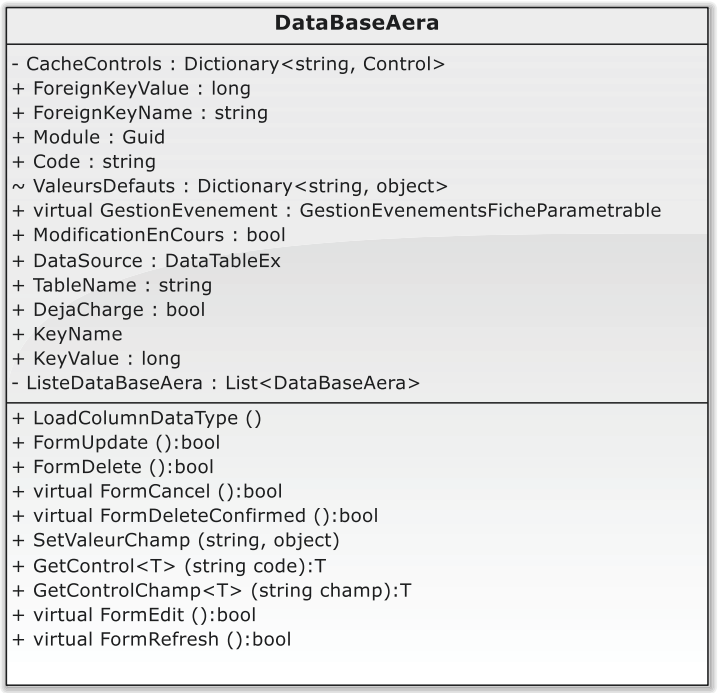


Figure 69 - Diagramme de classe DataBaseAera

Les différents types d'écrans disponibles sont présentés sur la Figure 70.

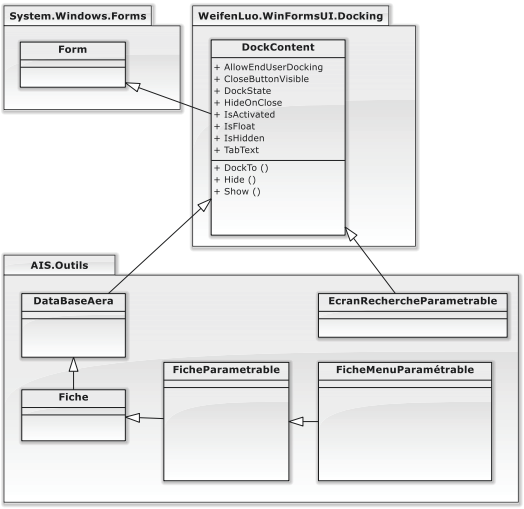


Figure 70 - Diagramme de classe général des Ecrans

3.5.4. Mélée 4

3.5.4.1. Mode réseau

L'objectif de ce mode réseau est de mettre en œuvre un système de communication entre les écrans d'une même application et ceux des autres instances de cette application sur la même machine ou sur les autres machines du réseau local. Il doit permettre de gérer les rafraichissements des données et prévenir les accès concurrentiels lors d'une modification de données. La synchronisation avec d'autres postes du réseau est optionnelle. Il faut donc que le fonctionnement du mode réseau soit transparent pour l'application. Le scénario de fonctionnement est présenté sur la figure suivante :

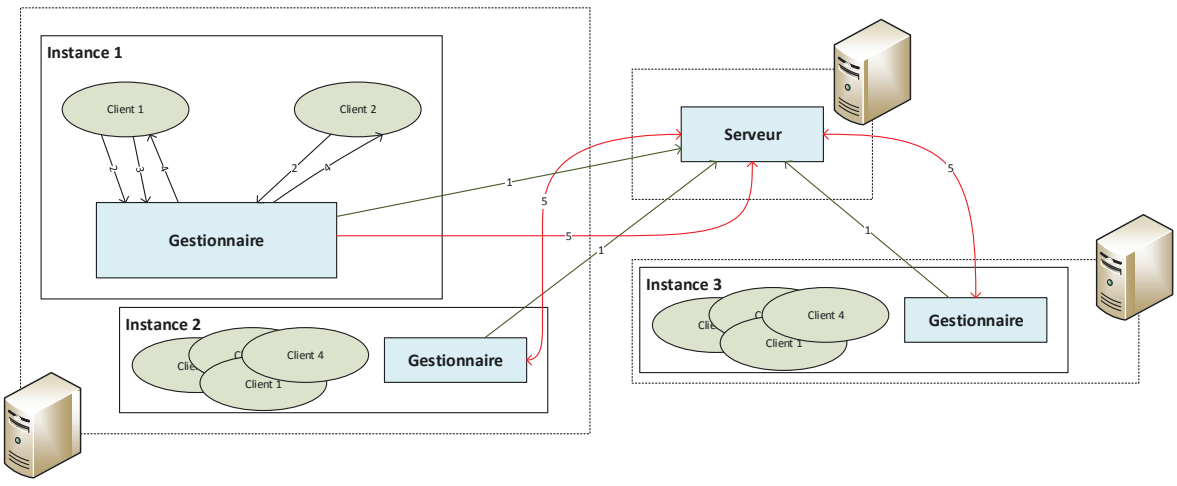


Figure 71 - Etapes de fonctionnement du système de notification

Etape 1 : Le gestionnaire de notification d'une nouvelle instance se connecte au serveur réseau si le mode de démarrage de l'application est réglé sur « mode réseau ». En cas d'échec, chaque nouvel évènement produira une nouvelle tentative de connexion au serveur. Cette approche évite le redémarrage des instances client en cas de perturbation du réseau temporaire ou de redémarrage du serveur réseau.

Etape 2 : Le client s'inscrit auprès du gestionnaire de son instance. Il précise pour quel évènement il souhaite être notifié. Le gestionnaire de notification maintient une liste des clients en fonction des évènements. Dans la majorité des cas, un évènement est identifié par un nom de table. Tous les autres cas concernent des messages système pour obtenir des informations du serveur réseau sur, par exemple, le nombre de connexions actives.

Etape 3 : Un client ou un traitement opère une modification qui nécessite l'actualisation des écrans. Un appel au gestionnaire permet de déclencher l'évènement correspondant. Cette étape sera automatisée par le gestionnaire de base de données puisqu'il est capable de déterminer à quel moment un enregistrement dans la table se produit.

Etape 4 : Chaque client inscrit pour l'évènement sera notifié. C'est au client de déterminer l'action à entreprendre en fonction des données complémentaires fournies avec la notification. Pour une table, il sera précisé la valeur de la clé primaire concernée. Dans cette étape, les écrans aussi pourront disposer d'un comportement automatisé par les classes de la bibliothèque d'outils.

Etape 5 : Si la connexion avec le serveur réseau est opérationnelle, l'évènement est envoyé au serveur qui le relayera à tous les autres gestionnaires inscrits. Les gestionnaires déclencheront les mêmes actions décrites à l'étape 3 et 4.

3.5.4.1.1. Codification des évènements

L'identification des types de messages est réalisée avec une simple codification d'une chaîne de caractères. Un message est composé nécessairement d'un type qui permettra de retrouver les clients inscrits qui sont en attente de réception de ce message. Des informations complémentaires peuvent être ajoutées. Pour cela on utilise le caractère de séparation « @ ». Par convention les types de messages relatifs aux tables de la base de

données sont préfixés par la chaîne « _TABLE_ ». Le tableau suivant donne la liste des messages utilisés avec leur format, leur condition de déclenchement ou de réception.

Tableau 12 - Messages utilisés dans les notifications

Message	Emetteur	Destinataire	Description
SRV@STOP	Serveur	Clients	Le serveur est en cours d'arrêt. Les clients basculent en mode déconnecté avec tentative de reconnexion à la prochaine émission de notification.
SRV@NOLICENSE	Serveur	Client	Le serveur indique au client que le nombre maximal de client simultané est atteint. Cette limite de clients est le seul mode de licence implémenté actuellement
SRV@QRYID	Serveur	Clients	Le serveur demande au client de calculer un identifiant unique de machine et retourne l'information au serveur. Ce message est implémenté en prévision d'un système de licence où chaque poste doit être activé individuellement
SRV@SETCON	Client	Serveur	Un client demande à recevoir les informations de diagnostic en temps réel du serveur.
CON@message	Serveur	Client	Le serveur envoie un message au client qui a activé le mode de diagnostic
_TABLE_Xxx	Clients	Clients	Un client envoie à tous les autres clients l'information que la table « Xxx » a été modifiée. En pratique la notification est envoyée par : • la classe *DataBaseAera* à la fin de la méthode **Enregistre**() • l'éditeur de table *TableEditor* • les listes déroulantes qui autorisent la modification du contenu émettent également des notifications.

3.5.4.1.2. Le Gestionnaire de notification

Le gestionnaire de notification est chargé de maintenir la liste des objets clients inscrits à un type d'évènement. Pour pouvoir rappeler les clients le moment venu, le gestionnaire a besoin de garder une référence vers l'instance du client et que celui-ci dispose d'une méthode de rappel. C'est l'objectif de l'interface *INotification*. Un objet doit implémenter cette interface pour pouvoir s'inscrire auprès du gestionnaire. Concrètement cela signifie que l'objet client doit disposer d'une méthode appelée Notification qui dispose d'un paramètre de type chaîne.

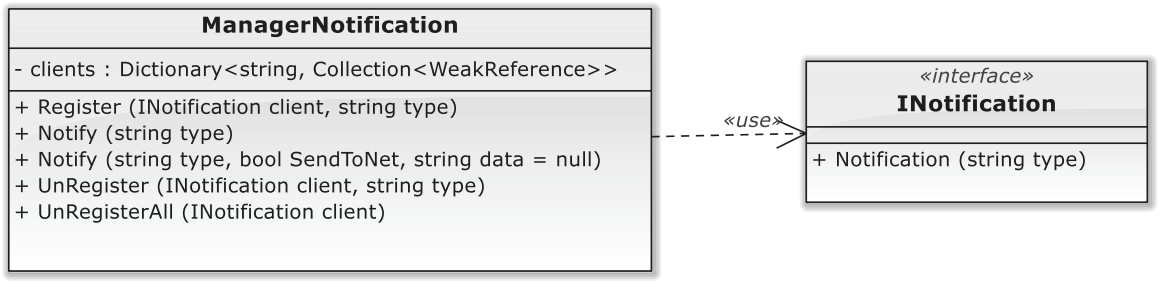


Figure 72 - Diagramme de classe du gestionnaire de notifications

L'environnement .Net dispose d'une gestion automatique de la mémoire. Il libère automatiquement la mémoire associée aux instances d'objets qui ne sont plus référencées. L'inscription auprès du gestionnaire de notification d'un écran implique nécessairement que l'écran avec tous les objets associés ne seront jamais libérés si le programmeur ne prend pas garde à la désinscription. Il faut trouver une solution automatisée à ce problème puisque d'une part les composants s'inscrivent automatiquement sans que le programmeur soit au courant et que d'autre part, en programmation objet orienté par évènement, on ne maitrise pas toujours l'enchainement des opérations. C'est la notion de WeakReference (référence faible) introduite avec la version 4.0 du Framework qui permet de solutionner ce problème. Le lien est en effet ignoré par l'algorithme de détection des dépendances avant la suppression des objets de la mémoire. Cela signifie qu'avant d'utiliser la référence lorsqu'un évènement se produit, il faut déterminer si l'objet client est toujours en mémoire. La classe WeakReference met à disposition, à cet effet, la propriété **IsAlive**.

3.5.4.1.3. Le serveur réseau de notification

Le serveur exploite des outils de la plateforme .Net appelés Windows Communication Foundation (WCF) destinés à simplifier le développement d'applications orientées services. C'est-à-dire que la totalité des mécanismes de communications entre un serveur et ses clients est prise en charge par la plateforme .Net. Certaines caractéristiques, comme la communication bidirectionnelle ou le maintien de la session de communication, sont définies dans un fichier de configuration.

Dans le cadre du serveur de notification, les différents gestionnaires communiquent avec le serveur comme avec n'importe quelle autre instance d'objet locale.

3.6. Bilan

La réalisation de ces 4 mêlées a été très intense. D'un point de vue technique, la complexité que l'on souhaite éviter au futur programmeur se traduit nécessairement par une augmentation de celle-ci dans la réalisation des outils. Il a fallu mettre en application toutes les recherches effectuées sur l'utilisation des composants. D'un point de vue organisationnel, la mise en œuvre d'une nouvelle technique de gestion de projet dans le cadre d'une équipe restreinte a été je pense bénéfique. Cela laisse le temps de bien comprendre l'intérêt de chaque étape du processus de la méthode SCRUM.

On constate que lors de la réalisation d'une fonctionnalité, les idées émergent assez facilement. La gestion du backlog permet d'ajouter continuellement des améliorations au fur et à mesure de leur découverte.

4. Le projet de GESTIONCP

4.1. Contexte et périmètre

Le CER40 souhaite le remplacement de la gestion papier des plannings des absences du personnel pour les différentes sociétés et sites du groupe. Elle se fait actuellement sous une forme classique par l'échange d'un formulaire papier entre un employé et son responsable hiérarchique. Les documents sont ensuite centralisés au service ressources humaines du site principal de Saint Pierre du Mont. Un décompte des jours disponibles est effectué à posteriori avec des outils de bureautique standards. Un changement au niveau de l'accord d'entreprise définit des règles très précises sur l'usage des jours de repos et de congés par type de poste. C'est pour vérifier la bonne application de ces règles et faciliter le suivi des demandes par les responsables que la mise en place d'un outil de gestion informatisé a été lancée. AIS a naturellement été désigné pour l'étude et la réalisation de la solution.

Le périmètre de ce projet se limite dans ce mémoire à la réalisation d'une application de paramétrage du logiciel de gestion de congés à partir de la première version de la boite à outils. Certaines étapes sont réalisées manuellement puisque une grande partie des outils graphiques n'est pas encore réalisée. Ils seront développés lors du projet **POISSONS V2**. Une partie de l'étude préalable est présentée afin d'identifier les paramètres qui doivent être pris en charge par l'application d'administration.

4.2. Etude préalable

4.2.1. 1 – Recommandations et philosophie du logiciel

Le document fourni par le CER40 est basé sur un modèle destiné à choisir un progiciel. De ce fait ce n'est pas vraiment un cahier des charges au sens strict du terme mais plutôt une liste de fonctionnalités générales, de contraintes ou d'aspects ergonomiques. Un extrait présenté ci-dessous montre les grandes lignes de la demande.

Les progiciels doivent pouvoir s'adapter aux modes de gestion des temps passés actuels et futurs du CERFRANCE LANDES en offrant la plus grande souplesse de paramétrage possible. Basé sur des concepts très généraux, le paramétrage permet de les adapter aux métiers du CER40.
En matière de sécurité, ils doivent permettre de gérer des droits d'accès, des habilitations par profil utilisateur, des habilitations par utilisateur.

Structure des données:
Découpage en grandes notions, telles que organisation, société, agence, service, équipe flux...
Toutes les données doivent être indépendantes des traitements.
L'interrogation des données par l'utilisateur doit être simple et lui procurer l'autonomie vis-à-vis du service informatique.

Gestion des informations:
L'information doit être unique, non-redondante et respecter les règles inhérentes au SGBDR, en l'occurrence SQL Server 2008. L'accès aux vues et à l'information doit se faire en temps réel et en fonction des droits de l'utilisateur (consultation, saisie et création, modification, suppression).
En terme d'archivage et de purge, nécessité de conserver en ligne 2 ans de données, d'effectuer des purges automatiques, d'archiver sur des supports hors ligne.

Processus de traitement:
Le principe d'unicité de la saisie est indispensable. Le progiciel doit permettre d'effectuer des traitements ou de la gestion, individualisés ou de masse.

Ergonomie:
Menus contextuels laissant apparaître les seules fonctions dont l'accès est autorisé à l'utilisateur.
Ergonomie générale de type WEB et mode d'application de type Serveur/Client (lourd), accès depuis un intranet, authentification utilisateur et mot de passe Windows.

4.2.2. Analyse

Des entrevues régulières avec Eric RIBBENS et une sélection d'utilisateurs représentatifs ont permis d'avancer étape par étape. Cette approche est parfaitement adaptée à la mise en œuvre d'un processus agile.

L'identification des acteurs a été rapide. L'utilisateur « standard » représente l'employé qui va saisir et consulter ses demandes de repos via une interface de type Web accessible par l'intranet de l'entreprise. Les personnes responsables des services seront chargées de valider (ou de refuser) les demandes depuis cette même application web. On appellera cet acteur « valideur ». Enfin il y a l' « administrateur » qui sera chargé du paramétrage du système. Il disposera d'une application de type client lourd réalisée avec la bibliothèque d'outils.

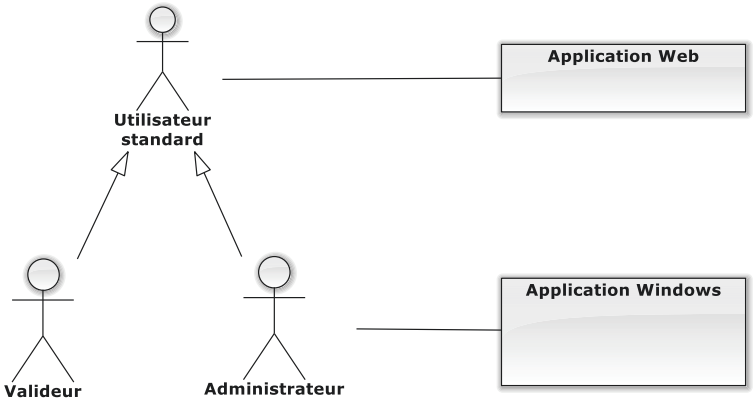


Figure 73 - Les acteurs du système de gestion des jours de repos

L'identification des cas d'utilisation pour chacun de ces acteurs est assez rapide également.

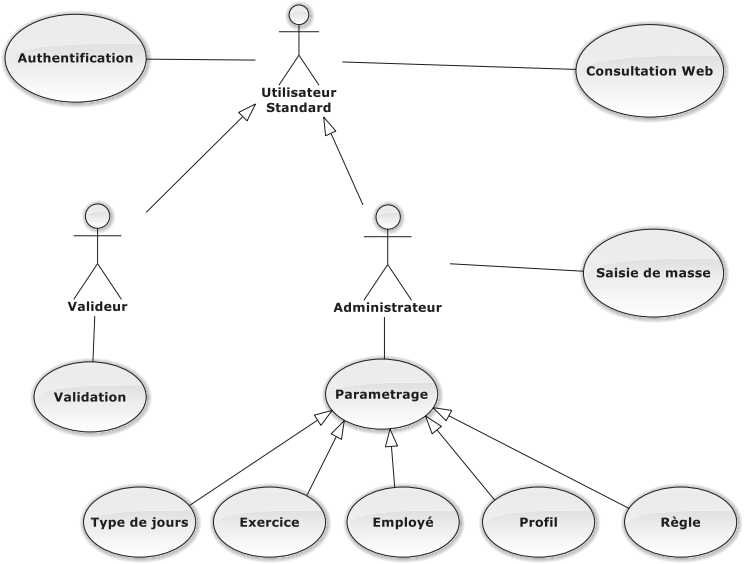


Figure 74 - Cas d'utilisations

L'**authentification** doit être automatique et transparente. C'est-à-dire qu'elle doit utiliser l'identité du compte utilisateur Windows. La plateforme de développement prévoit déjà cette possibilité. Il faudra s'assurer que le serveur web puisse également en bénéficier.

La **consultation Web** consiste pour un utilisateur standard à pouvoir visualiser son planning personnel et déposer des demandes de jours de repos en fonction des règles en vigueur. Cela comprend des limitations spécifiques au type de jour et du service de la personne. La visibilité en lecture seule du planning d'autres personnes peut également être octroyée par le biais d'un rôle.

La **validation** permet au responsable hiérarchique de modifier l'état d'une demande. Elle peut être acceptée ou refusée. La suppression reste du ressort du propriétaire de la demande.

La **saisie de masse** permet à l'administrateur de modifier rapidement le planning sans aucune restriction. Cette fonctionnalité est réservée au responsable des ressources humaines. Pour éviter tout litige, l'ensemble des manipulations du planning doit faire l'objet d'un enregistrement scrupuleux.

Le **paramétrage** comprend la définition de l'ensemble des paramètres utiles au bon fonctionnement du système.

4.3. Mise en œuvre

J'ai encadré lors de la réalisation de ce projet Fabrice CASTAGNETTI qui effectuait chez AIS un stage de 10 semaines dans le cadre de sa formation prodiguée par l'Association pour la Formation Professionnelle des Adultes (AFPA) de Pau.

4.3.1. Architecture de la solution

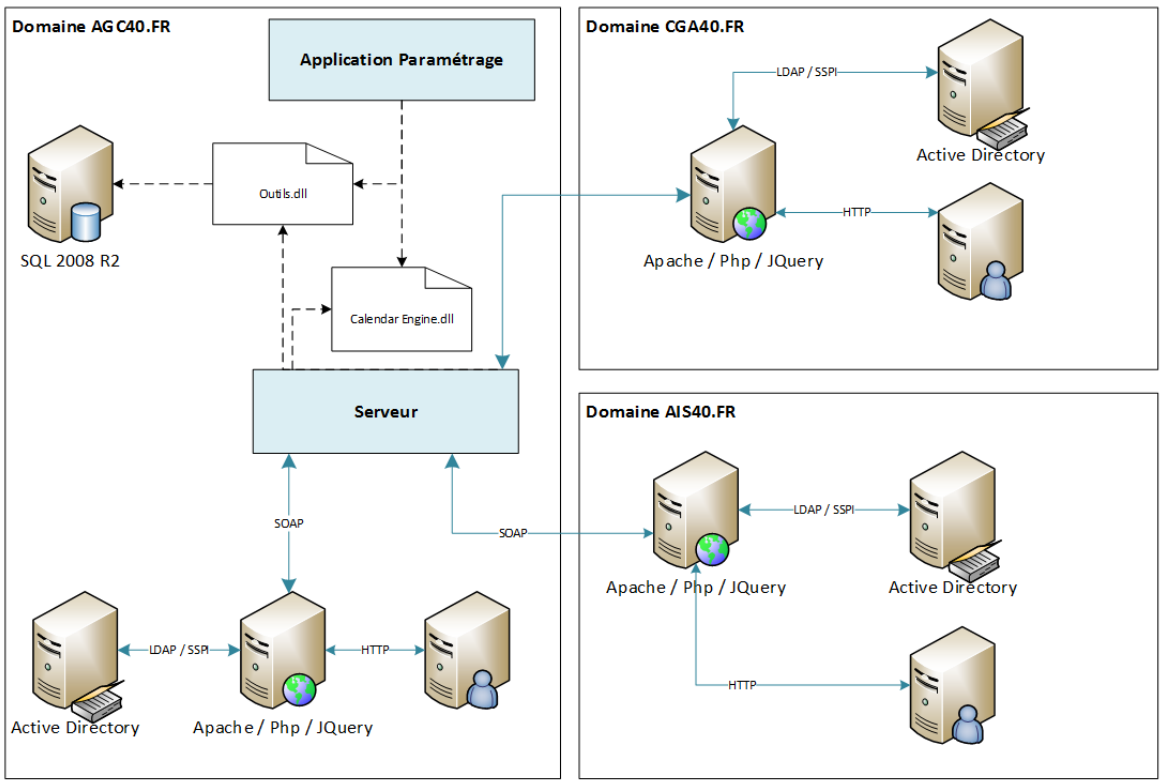


Figure 75 - Architecture technique de la solution de GESTIONCP

Les données sont stockées dans une base SQL Server 2008 R2. Un serveur sous forme de service Windows utilise la bibliothèque d'outils pour accéder à la base de données. Il est chargé de fournir via des services web (protocole SOAP[18]) les données ainsi que les traitements nécessaires aux serveurs **Apache** qui hébergent la partie web de la solution de gestion de congés. Les domaines de sécurité Windows gérés par le CER40 sont multiples. Il y a AGC40.FR, CGA40.FR et AIS40.FR. Pour rendre possible l'authentification automatique de l'utilisateur par le navigateur web **Firefox**, il est nécessaire que le serveur web se trouve également dans le domaine. Il a donc fallu prévoir 3 installations du serveur **Apache**.

L'application de paramétrage est réalisée entièrement par du code compatible avec la bibliothèque d'outils. Un *assembly* de génération de calendrier sous forme d'image est utilisé en commun avec le serveur. Cette approche garantit la centralisation des traitements. La partie web ne contient aucune règle de gestion métier. Elle se contente de mettre en forme les données fournies par le serveur. La partie interactive du calendrier est également prise en charge par le serveur, le navigateur web transmet les informations des clics de souris.

4.3.2. Conception de la base de données

L'utilitaire graphique de modification du schéma de la base de données n'est pas encore réalisé. La création des tables est donc effectuée avec l'outil SQL Management Studio et doit respecter la nomenclature pour être compatible avec les outils.

Tableau 13 - Structure de la table société (Soc)

Champ	Description	Type de donnée
Soc	Identifiant	bigint
SocDes	Nom de la société	nvarchar(50)

La table **Soc** contient la liste des sociétés du groupe CER40.

[18] Simple Object Access Protocole est un protocole permettant à 2 objets distants de communiquer par des messages standardisés par le World Wide Web Consortium.

Tableau 14 - Structure de la table site (Sit)

Champ	Description	Type de donnée
Sit	Identifiant	bigint
SitDes	Nom du site	nvarchar(250)
SitAdr	Adresse	nvarchar(2500)
SitCp	Code postal	nvarchar(50)
SitVil	Ville	nvarchar(50)
SitTel	Téléphone	nvarchar(50)
SitFax	Fax	nvarchar(50)
SitMob	Téléphone portable	nvarchar(50)
SitCon	Contact principal	nvarchar(250)
SitMel	Courrier électronique	nvarchar(50)
SitAct	Actif	bit
SitSoc	Société	bigint

La table **Sit** répertorie les emplacements géographiques qui composent les locaux d'une société.

Tableau 15 - Structure de la table service (Ser)

Champ	Description	Type de donnée
Ser	Identifiant	bigint
SerDes	Nom du service	nvarchar(50)
SerSit	Site	bigint

Dans chaque site, il existe plusieurs services.

Tableau 16 - Structure de la table jour (Day)

Champ	Description	Type de donnée
Day	Identifiant	bigint
DayDat	Date	datetime
DayMat	Matin	bit
DayApr	Après-midi	bit
DayLck	Verrouillage	bit
DaySit	Site	bigint
DaySoc	Société	bigint
DayJou	Type de jour	bigint

La table **Day** contient la description des jours particuliers que l'administrateur va pouvoir positionner à l'avance sur le calendrier. Ces jours peuvent être spécifiques à une société, un site ou communs à l'ensemble des employés. On peut citer par exemple les jours fériés ou les périodes de fermeture obligatoires d'un site. La notion de verrouillage détermine la capacité de l'employé à remplacer un jour positionné par l'administrateur.

Tableau 17 - Structure de la table type de jour (Jou)

Champ	Description	Type de donnée
Jou	Identifiant	bigint
JouDes	Nom abrégé du type de jour	nvarchar(50)
JouCouR	Indice de couleur rouge	int
JouCouG	Indice de couleur verte	int
JouCouB	Indice de couleur bleue	int
JouSpe	Jour spécial	bit
JouLib	Nom complet du type de jour	bit
JouFus	Fusion avec les cases de la même ligne possible	bit
JouAdm	Disponible uniquement pour l'administrateur	bit

La table **Jou** définit les types de jour que l'on peut positionner sur le calendrier. Un jour est défini comme « administrateur » si seul l'administrateur peut l'utiliser. Par exemple, les absences injustifiées, les congés sans solde entrent dans cette classification. La fusion détermine si, dans le cas d'un affichage du calendrier de plusieurs employés, des jours identiques sur une même ligne donnent lieu à un affichage groupé. Un jour « spécial » ne fait pas l'objet d'un décompte. Chez AIS, il existe le type de jour « Cesi » qui indique que la personne en contrat en alternance est au centre de formation du CESI.

Tableau 18 - Structure de la table exercice (Exo)

Champ	Description	Type de donnée
Exo	Identifiant	bigint
ExoDes	Libellé de l'exercice	nvarchar(50)
ExoSoc	Société	bigint
ExoDeb	Date de début	datetime
ExoFin	Date de fin	datetime

La table **Exo** contient la liste des périodes pour chaque société où la quantité de jours disponibles doit être dépensée. Ces dates de références sont utiles pour déterminer en fonction de la date quel est l'exercice correspondant. Il existe des tolérances pour dépenser certains types de jour qui donnent lieu à des chevauchements sur les exercices. Ils sont définis dans la table **Per**.

Tableau 19 - Structure de la table période (Per)

Champ	Description	Type de donnée
Per	Identifiant	bigint
PerDeb	Date de début	nvarchar(50)
PerFin	Date de fin	datetime
PerExo	Exercice	datetime
PerDes	Libellé de la période	nvarchar(50)
PerJou	Type de jour	bigint

Tableau 20 - Structure de la table évènement (Evt)

Champ	Description	Type de donnée
Evt	Identifiant	bigint
EvtDat	Date	datetime
EvtEmp	Employé	bigint
EvtMat	Matin	bit
EvtLck	Verrouillage	bit
EvtExo	Exercice	bigint
EvtJou	Type de jour	bigint

Chaque évènement est stocké sous forme de demi-journée dans la table **Evt**. Le verrouillage correspond ici au résultat de la validation d'une demande. La séquence de positionnement d'un événement est décrite sur la figure suivante :

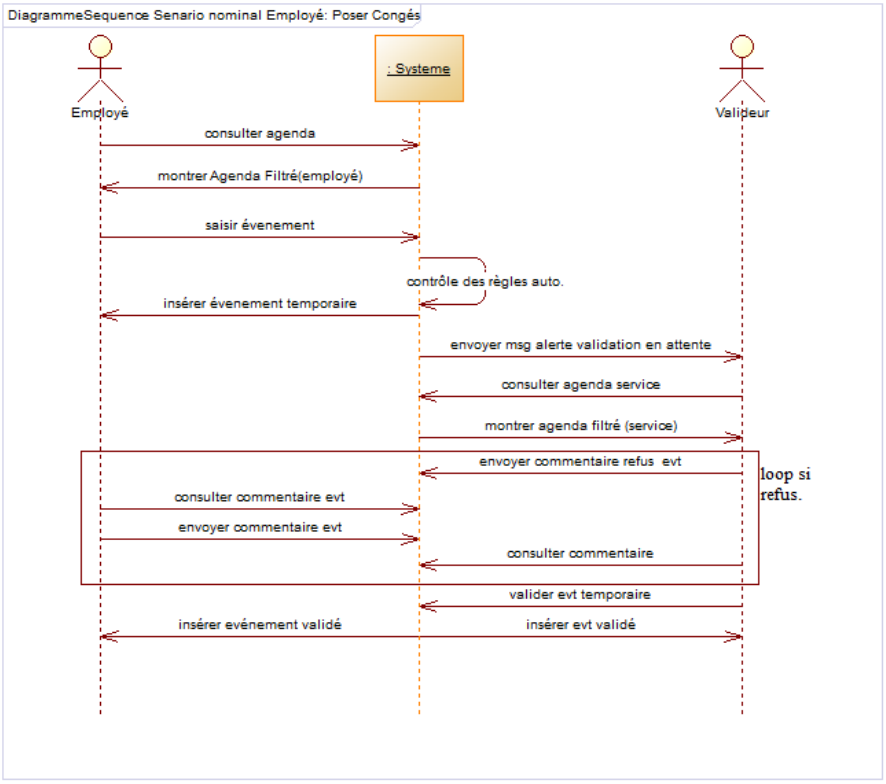


Figure 76 - Diagramme de séquence de dépense de jour de congé

Tableau 21 - Structure de la table profil (Prf)

Champ	Description	Type de donnée
Prf	Identifiant	bigint
PrfDes	Nom du profil	nvarchar(50)
PrfLun	Lundi	bit
PrfMar	Mardi	bit
PrfMer	Mercredi	bit
PrfJeu	Jeudi	bit
PrfVen	Vendredi	bit
PrfSam	Samedi	bit
PrfDim	Dimanche	bit

La table **Prf** décrit les profils des employés avec les journées travaillées dans une semaine type.

Tableau 22 - Structure de la table de liaison profil-jour (PrfJou)

Champ	Description	Type de donnée
Prj	Identifiant	bigint
PrjPrf	Profil	bigint
PrjJou	Type de jour	bigint
PrjQte	Quantité à dépenser	float

Pour chaque profil, la table **PrfJou** contient une quantité par type de jour qui est disponible à chaque début d'exercice. Ces informations permettent d'initialiser rapidement les quotas de chaque employé à la suite d'une création d'exercice.

Tableau 23 - Structure de la table règle (Reg)

Champ	Description	Type de donnée
Reg	Identifiant	bigint
RegDes	Nom de la règle	nvarchar(250)
RegCls	Nom de la classe d'implémentation	nvarchar(50)

La table **Reg** énumère les règles de gestion disponibles qui sont applicables lors de la création d'un évènement. Chaque règle correspond à une classe d'un objet implémentée dans l'application Serveur. L'activation d'une règle dépend de l'association avec un profil dans la table **PrfReg**.

Tableau 24 - Structure de la table de liaison profil-règle (PrfReg)

Champ	Description	Type de donnée
Prg	Identifiant	bigint
PrgPrf	Profil	bigint
PrgReg	Règle	bigint

Tableau 25 - Structure de la table employé (Emp)

Champ	Description	Type de donnée
Emp	Identifiant	bigint
EmpNom	Nom	nvarchar(50)
EmpPre	Prénom	nvarchar(50)
EmpMel	Courrier électronique	nvarchar(250)
EmpAct	Actif	bit
EmpSit	Site	bigint
EmpUti	Code Utilisateur	bigint
EmpPrf	Profil	bigint

La table **Emp** contient la liste des employés rattachés à leur site.

Tableau 26 - Structure de la table rôle (Rol)

Champ	Description	Type de donnée
Rol	Identifiant	bigint
RolDes	Nom du rôle	nvarchar(50)
RolSys	Indispensable pour le fonctionnement	bit

La table **Rol** contient actuellement uniquement les rôles « membre », « valideur » et « consultation ». Ces rôles permettent de définir dans la table **EmpSer** la nature de la relation entre un employé et un service. Avec cette approche on peut facilement trouver les personnes chargées de valider les demandes d'un employé dans un service. On peut également, en considérant que le « valideur de mon valideur est aussi mon valideur », déterminer la hiérarchie des personnes chargées de valider les demandes et déterminer qui peut remplacer un valideur absent.

Tableau 27 - Structure de la table de liaison employé-service (EmpSer)

Champ	Description	Type de donnée
Esr	Identifiant	bigint
EsrEmp	Employé	bigint
EsrSer	Service	bigint
EsrRol	Rôle	bigint

Tableau 28 - Structure de la table de liaison employé-type de jour (EmpJou)

Champ	Description	Type de donnée
Ejo	Identifiant	Bigint
EjoEmp	Employé	bigint
EjoExo	Exercice	bigint
EjoJou	Type de jour	bigint
EjoQuo	Quantité disponible	float

La table **EmpJou** permet de mémoriser pour chaque employé le quota par type de jour pour chaque exercice.

Tableau 29 - Structure de la table commentaire (Cmt)

Champ	Description	Type de donnée
Cmt	Identifiant	bigint
CmtSem	Semaine	int
CmtAn	Année	int
CmtEmp	Employé	bigint
CmtDat	Date	datetime
CmtDes	Texte du commentaire	nvarchar(2500)

La table **Cmt** est destinée à enregistrer, par numéro de semaine, les messages échangés par l'interface web entre un employé et son valideur.

La Figure 77 donne une représentation graphique du schéma relationnel de la base de données avec les différentes tables de paramétrage.

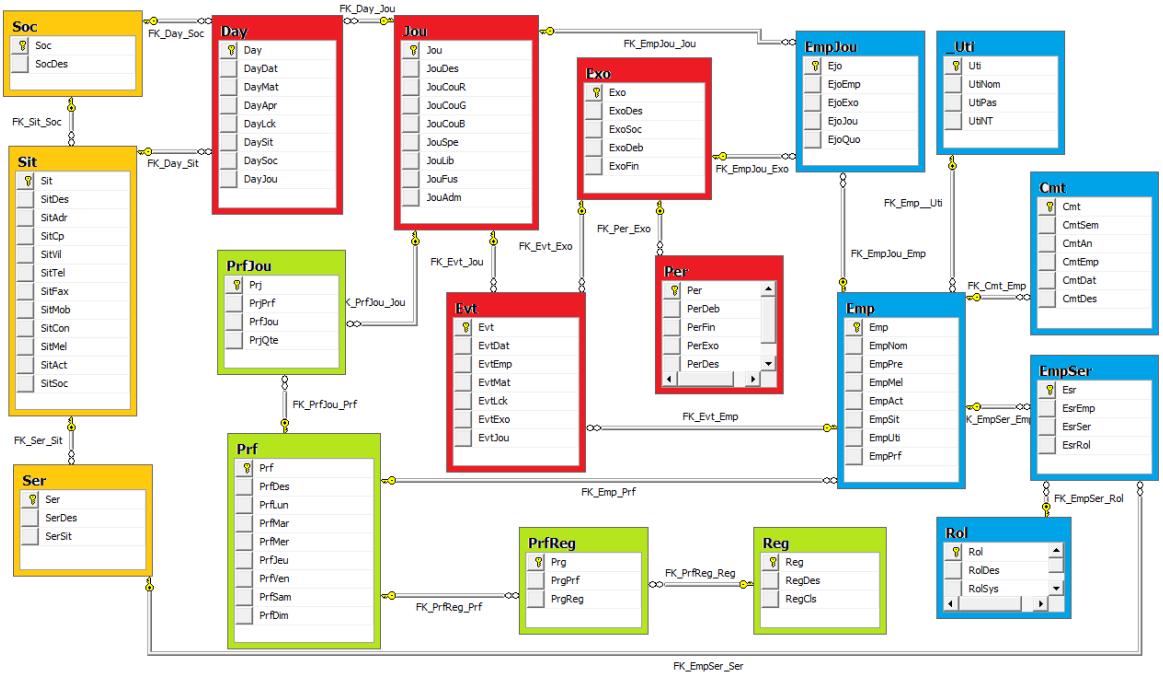


Figure 77 - Schéma relationnel de la base de GESTIONCP

4.3.3. Construction des écrans de paramétrage

Une édition des tables **_EdeDef** et **_EdeCol** par SQL Management Studio a permis de construire les écrans en moins d'une semaine. Un échantillon de ces écrans est présenté dans les figures suivantes :

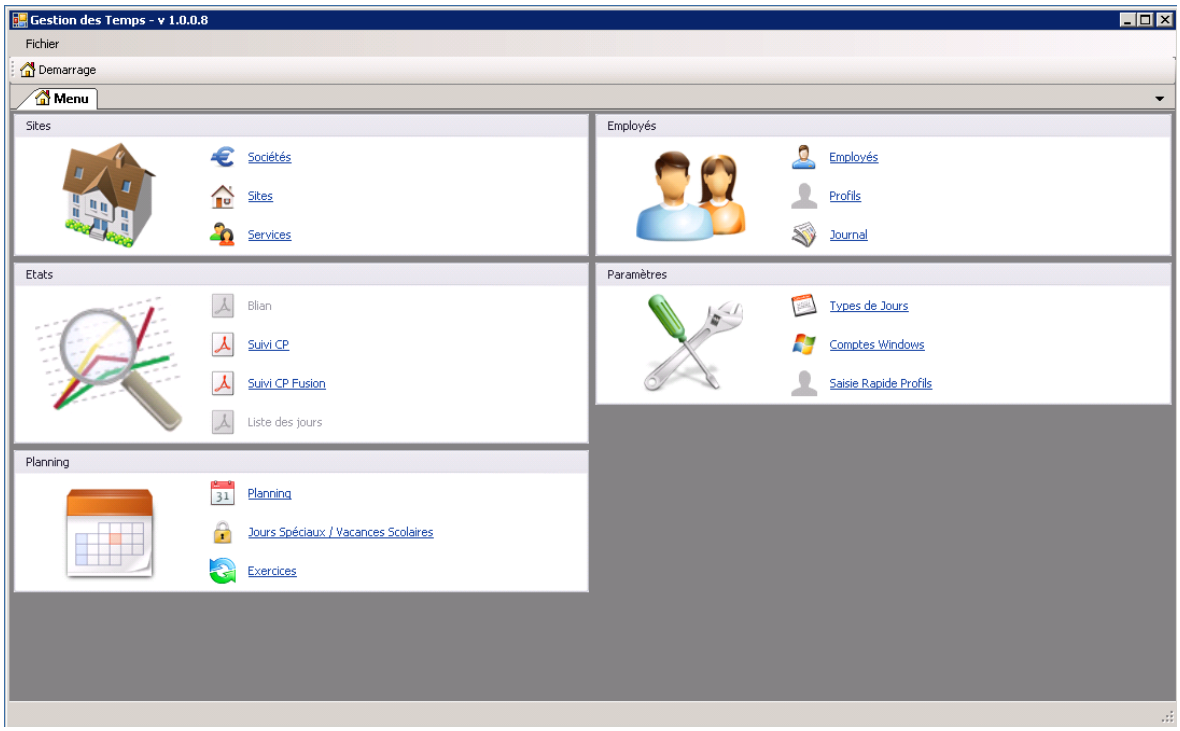


Figure 78 - Menu de l'application de paramétrage

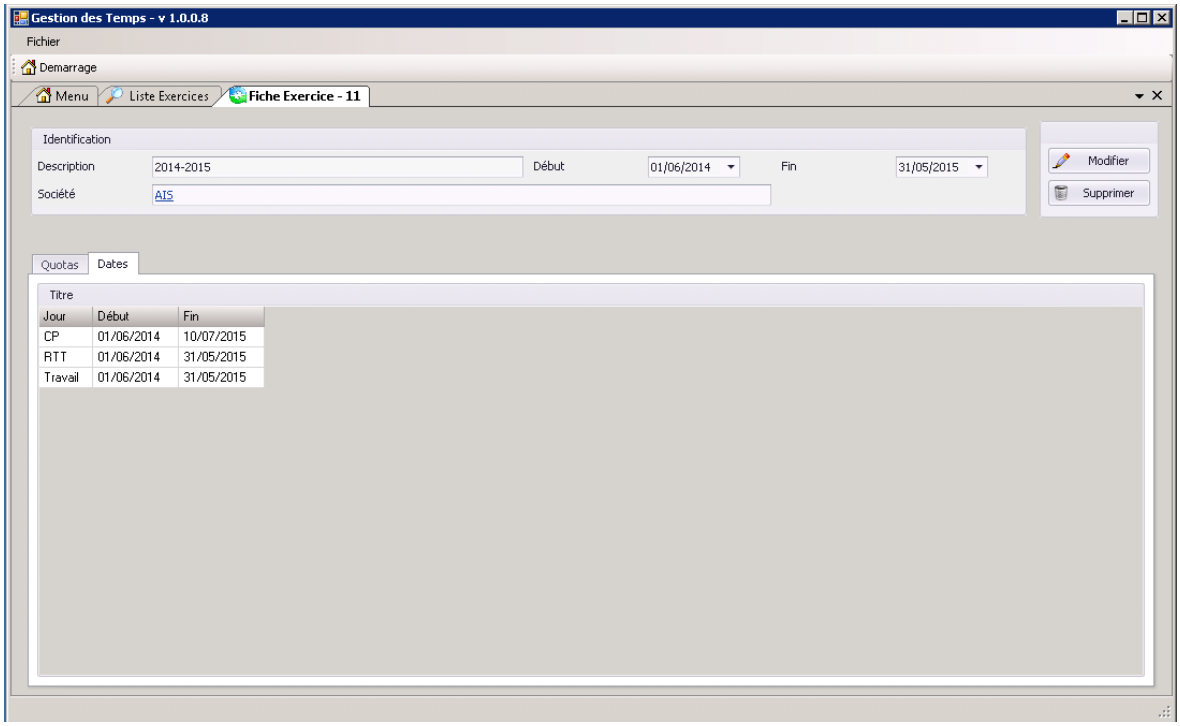


Figure 79 - Fiche exercice et périodes

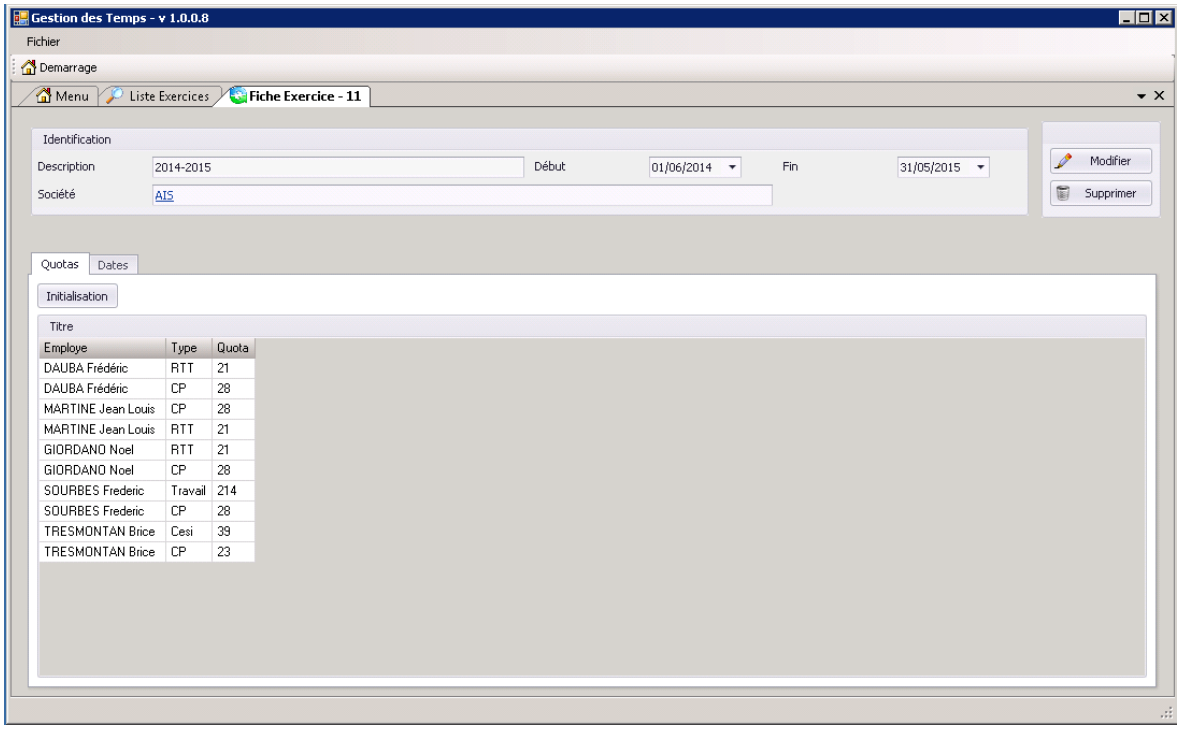


Figure 80 - Fiche exercice et quotas

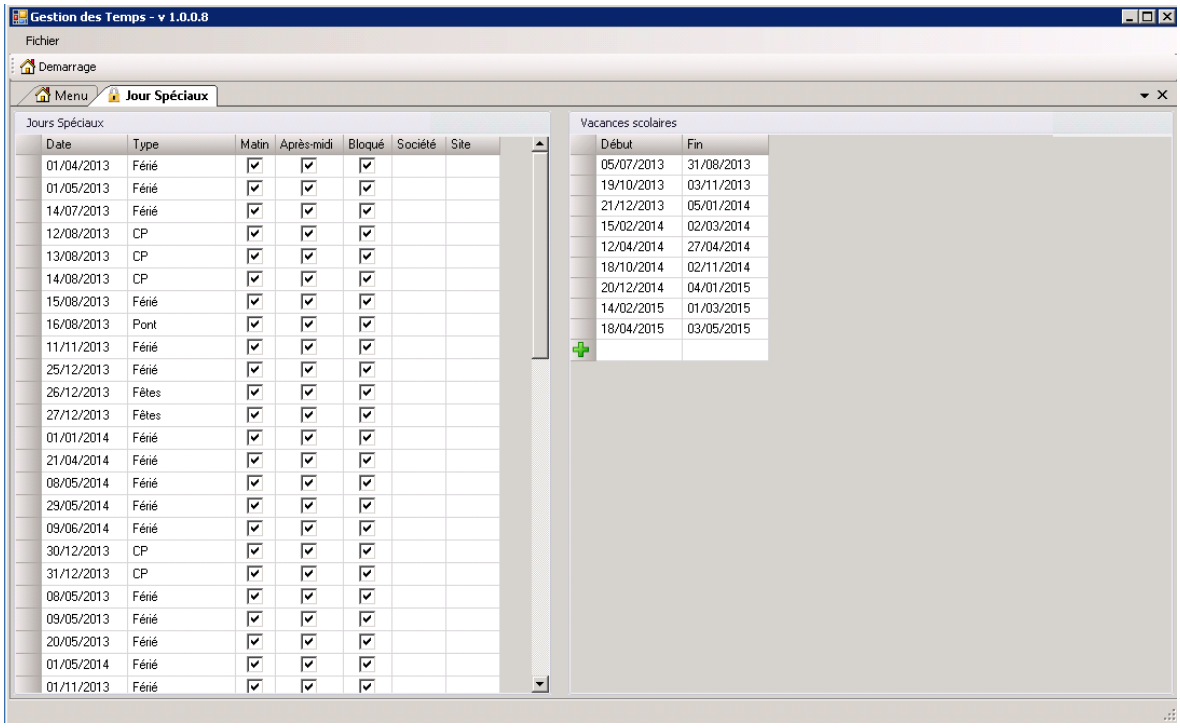


Figure 81 - Jours spéciaux

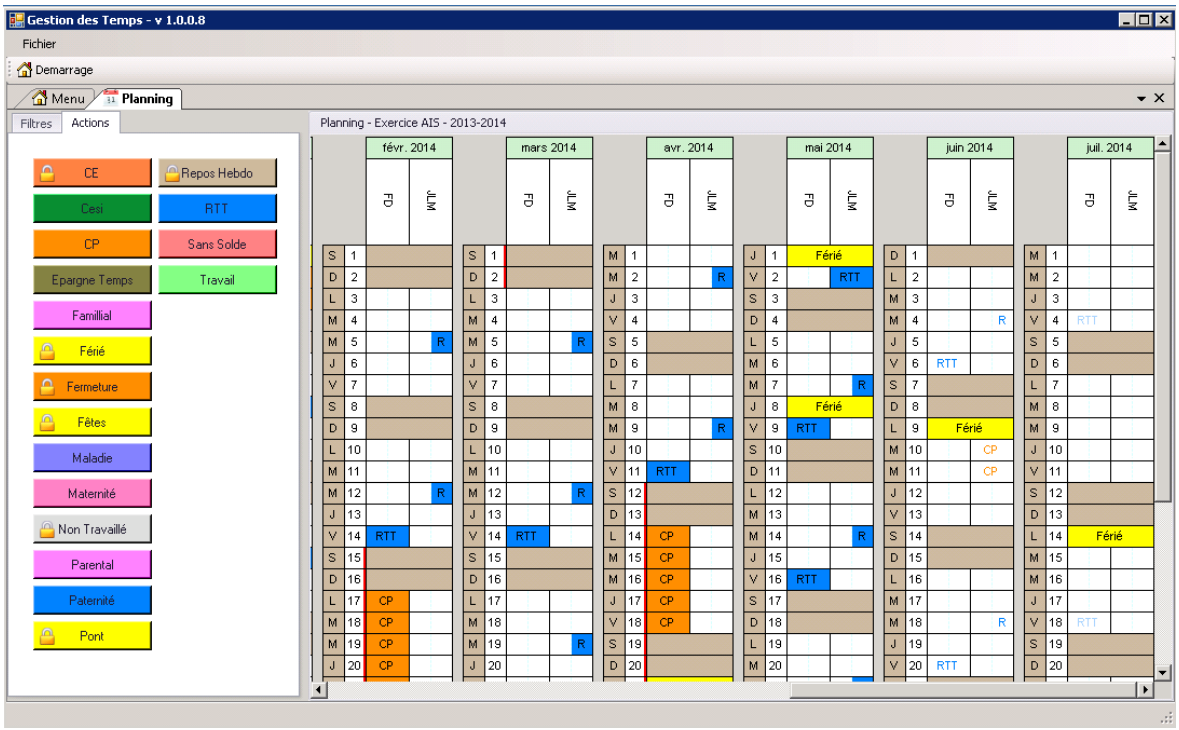


Figure 82 - Ecran de modification du calendrier pour l'administrateur

L'écran de modification du calendrier dans l'application de paramétrage est un exemple d'écran spécifique. Son contenu et sa programmation évènementielle sont entièrement réalisés par du code.

5. Le projet POISSONS V2

Ce projet est à l'origine du développement de la plateforme de développement rapide. La demande de possibilité de personnalisation de la nouvelle version du logiciel **POISSONS** repose sur le fonctionnement même du GDSAA dans les domaines sanitaire et environnemental.

J'ai travaillé avec Sylvain HAUDRECHY durant les 6 semaines de son stage chez AIS dans le cadre de sa formation au Centre des Etudes Supérieures Industrielles (CESI) de Pau.

5.1. Amélioration des outils

Dans le cadre de ce mémoire, seul le début de ce projet est abordé. De ce fait, nous allons voir les améliorations apportées à la bibliothèque d'outils afin de satisfaire les demandes de personnalisation de l'application.

5.1.1. Editeur de table de la base de données

L'éditeur de table doit permettre à un utilisateur de créer ou de modifier des tables en toute sécurité pour les données. Les règles de nomenclature nécessaires au bon fonctionnement des outils doivent être respectées et l'éditeur doit masquer au maximum toutes ces considérations techniques. Les noms de tables et de champs doivent être remplacés par des libellés. Ces libellés doivent également être disponibles pour la création de requêtes, par exemple pour la fusion de documents avec des logiciels de bureautique, ainsi que dans l'éditeur d'état.

L'accès à l'éditeur de tables est ajouté dans le *TreeView* du mode configuration comme le montre la Figure 83. En déroulant l'entrée « Tables » on obtient la liste des tables existantes ainsi que l'élément « Ajouter une table ». Cette liste est obtenue par l'intermédiaire de la bibliothèque d'objets SQL Management Objects (SMO) fournie avec le moteur de base de données. L'énumération des tables est une opération simple mais ici on utilise la possibilité de définir des propriétés personnalisées sur les éléments de la base SQL Server. Ainsi la propriété **Description** est utilisée pour stocker le libellé qui remplacera le

nom de la table lors de l'affichage. Si cette propriété n'est pas définie ou si elle ne possède pas de valeur, le nom réel de la table est affiché entouré par les caractères «[» et «]». La propriété **Modifiable** conditionne également l'apparition des tables dans la liste. Les tables particulières utilisées par la plateforme n'apparaissent pas et possèdent la valeur « Non » pour la propriété **Modifiable**. Ce sont les tables dont le nom commence par le caractère « _ ».

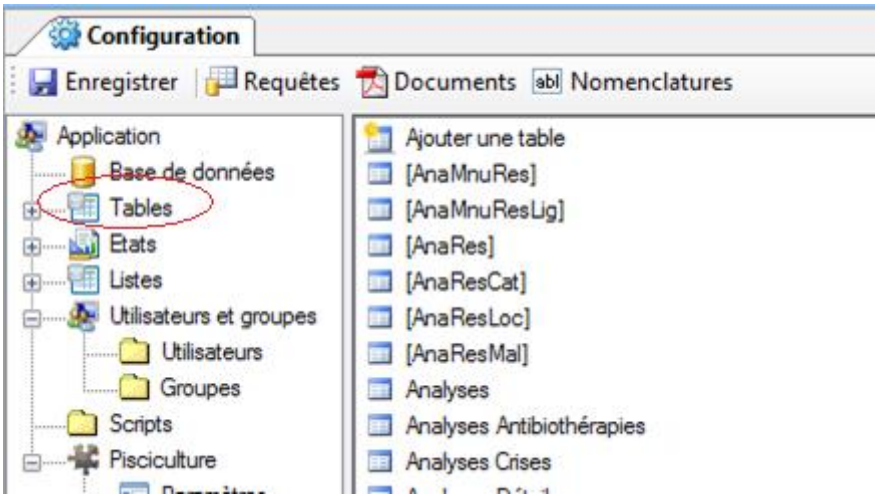


Figure 83 - Emplacement de l'éditeur de tables dans le module configuration

La sélection de l'entrée « Ajouter une table » permet d'activer le panneau de paramétrage présenté sur la Figure 84. La saisie de la description entraine un calcul pour déterminer le nom et le préfixe de la table à créer. L'unicité des préfixes utilisés dans les noms de champs est importante. Si le préfixe est déjà utilisé dans une autre table, un message indiquera la table concernée et il ne sera pas possible de créer une nouvelle table. Il est donc possible de modifier chacune des zones jusqu'à l'obtention d'un nom de table et de préfixe correct.

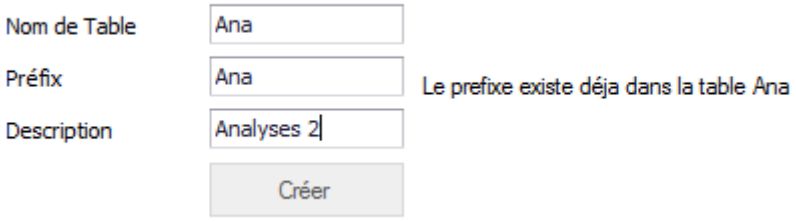


Figure 84 - Ecran d'ajout de table

Le bouton de création effectuera les opérations de création toujours par le biais de la bibliothèque SMO. Conformément aux conventions définies par la bibliothèque, une clé primaire de type long sera créée automatiquement et nommée comme le préfixe choisi. Après la création de la table, le panneau de modification de table est automatiquement ouvert.

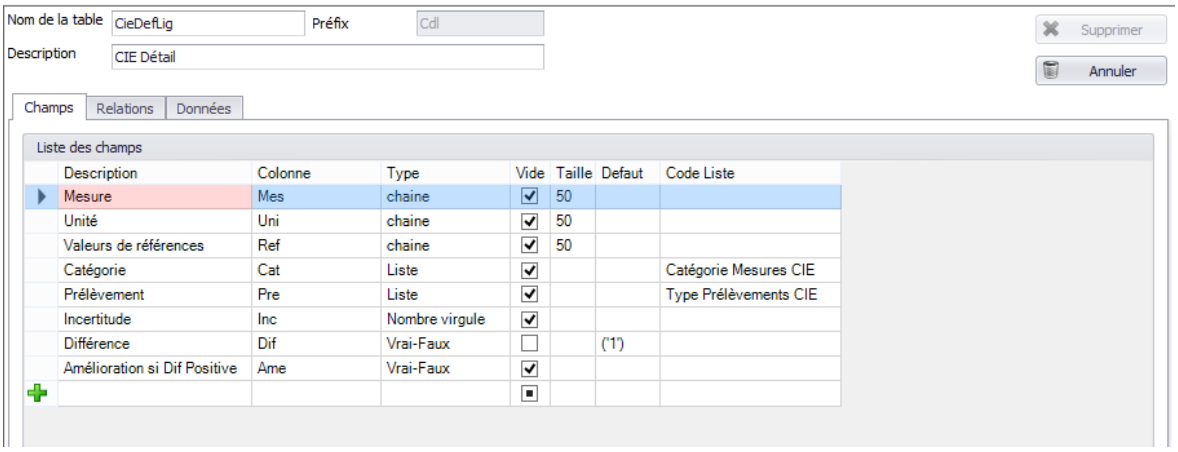


Figure 85 - Modification des champs d'une table

Un bouton permet de supprimer la table uniquement si elle ne contient pas de ligne. Le bouton **Annuler** recharge le schéma de table et annule ainsi les modifications non enregistrées. Il est possible de changer la description associée à la table afin de modifier le nom visible de la table. Les onglets Champs, Relations et Données permettent respectivement de modifier le schéma, les relations (contraintes de clé étrangères) avec les autres tables et enfin le contenu de la table.

L'onglet **Champs** permet l'édition du schéma. La saisie de la description entraine le calcul du nom du champ afin de respecter les conventions de nommage plus facilement. Le préfixe de champ n'est pas affiché. Le type de données est sélectionné par une liste déroulante contenant un nom francisé :

- Entier
- Entier Long
- Chaîne
- Nombre virgule
- Vrai-Faux
- Date
- Liste

Une case à cocher indique si le champ accepte de ne pas avoir de valeur ce qui aura un impact lors des vérifications avant l'enregistrement d'une fiche. Pour le type chaîne, on peut indiquer la taille maximale. Une valeur par défaut peut être spécifiée pour tous les autres types de champ. Un champ de type Liste sera traduit par un contrôle liste déroulante dans les écrans. Le champ est en réalité une clé étrangère vers une table dont la description est choisie parmi les listes disponibles (voir la partie 5.1.2. concernant la notion de liste).

Pour faciliter l'édition, l'écran empêche la suppression d'un champ s'il existe au moins une valeur pour ce champ dans les lignes existantes. De plus, les noms de champs sont mis en gras lorsqu'ils sont utilisés dans l'éditeur d'écran. C'est-à-dire qu'ils sont visibles sur l'écran au travers d'un contrôle de saisie ou qu'ils sont dans une formule conditionnant la visibilité ou le verrouillage d'un autre champ.

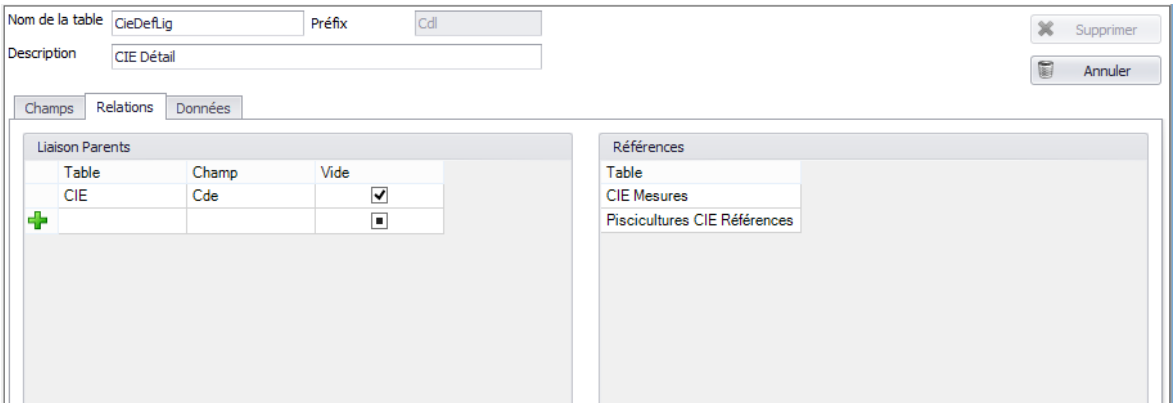
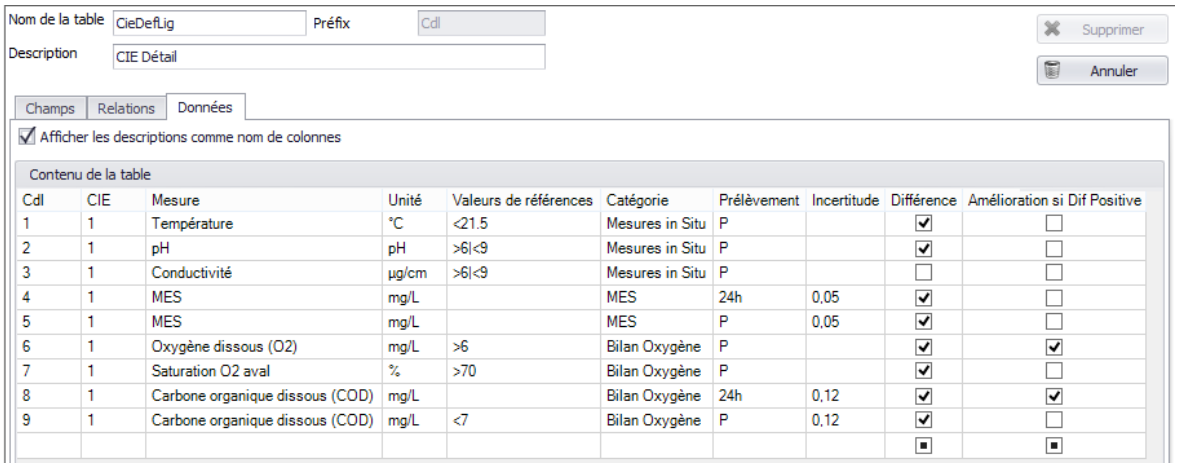


Figure 86 - Edition des relations d'une table

L'onglet **Relations** permet de sélectionner les tables connexes. Pour chacune des tables, un champ sera créé comme pour le type de champ Liste. Par défaut le nom du champ est « Auto » lors de l'ajout d'une nouvelle relation. Il est modifiable dans le cas où la même table doit être ajoutée plusieurs fois. Une liste des tables qui référencent la table en cours de modification est affichée.

Enfin l'onglet **Données** donne la possibilité d'éditer directement le contenu de la table en affichant les noms réels de champs ou les descriptions. L'enregistrement de cette grille d'édition est indépendante du reste de l'écran ce qui permet par exemple de vider la table et ensuite de supprimer la table.



Cdl	CIE	Mesure	Unité	Valeurs de références	Catégorie	Prélèvement	Incertitude	Différence	Amélioration si Dif Positive
1	1	Température	°C	<21.5	Mesures in Situ	P		☑	☐
2	1	pH	pH	>6|<9	Mesures in Situ	P		☑	☐
3	1	Conductivité	µg/cm	>6|<9	Mesures in Situ	P		☐	☐
4	1	MES	mg/L		MES	24h	0,05	☑	☐
5	1	MES	mg/L		MES	P	0,05	☑	☐
6	1	Oxygène dissous (O2)	mg/L	>6	Bilan Oxygène	P		☑	☑
7	1	Saturation O2 aval	%	>70	Bilan Oxygène	P		☑	☐
8	1	Carbone organique dissous (COD)	mg/L		Bilan Oxygène	24h	0,12	☑	☑
9	1	Carbone organique dissous (COD)	mg/L	<7	Bilan Oxygène	P	0,12	☑	☐
								■	■

Figure 87 - Modification directe du contenu d'une table

L'enregistrement du schéma d'une table est suivi de la création ou de la mise à jour d'une vue dont les noms des champs et de la vue sont ceux des libellés de description. Dans cette vue, les champs de type liste afficheront le libellé correspondant dans la table connexe et non la valeur du champ. Cette approche limite le nombre de tables à mettre en relation lors de la conception d'une requête pour obtenir des données exploitables dans un état.

5.1.2. Editeur de Listes

Les listes sont des tables avec un schéma défini par la bibliothèque de développement destinées à alimenter les listes déroulantes et faciliter les requêtes pour les états. La structure, comme le montre la Figure 88, est composée d'une clé primaire au format *uniqueidentifier*, d'une description de 50 caractères maximum et d'une valeur numérique pour ordonner la liste autrement que par l'ordre alphabétique.

Nom de la colonne	Type de données	Autoriser les valeurs Null
Par	uniqueidentifier	☐
ParDes	nvarchar(50)	☐
ParOrd	int	☑
ParOldVal	nvarchar(50)	☑

Figure 88 - Schéma d'une table de liste

Après la suppression ou l'ajout d'une nouvelle liste, la vue dbo.ParView est mise à jour. Elle est constituée par une requête d'union de toutes les tables correspondant à des listes. Ainsi il est possible d'écrire une fonction au niveau du moteur de bases de données qui va effectuer une requête sur cette vue. Il sera alors possible d'obtenir le libellé de

n'importe quel champ de type liste sans connaitre le nom de la table associée. Cette fonction nommée **dbo.ParLbl()** est utilisée également dans les vues générées automatiquement pour chaque table modifiable.

L'accès à l'éditeur de listes s'effectue comme pour l'éditeur de table dans le mode configuration. En dessous de l'entrée du *TreeView* se trouve une entrée dédiée à la création d'une nouvelle liste puis apparaissent par ordre alphabétique les listes existantes. Le libellé de celles qui sont utilisées dans une table est en gras.

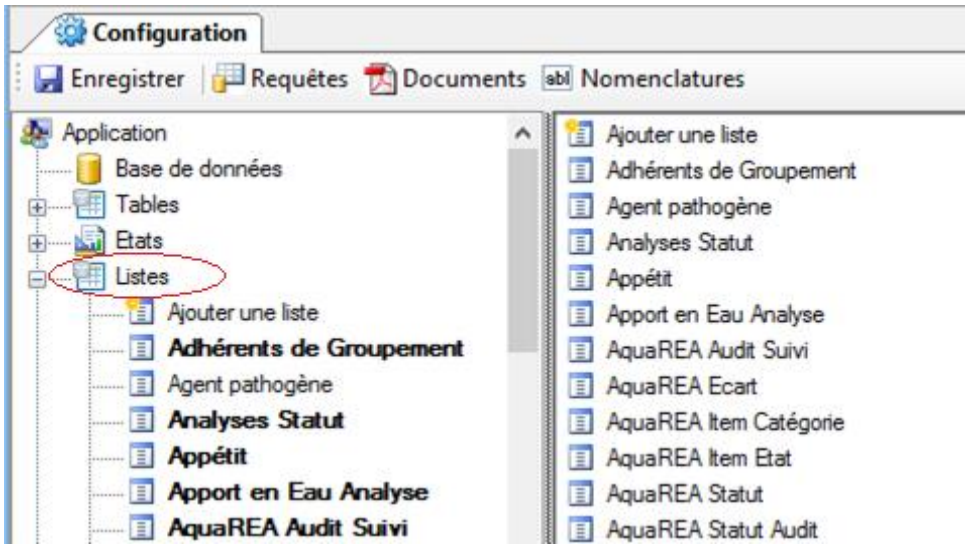


Figure 89 - Emplacement de l'éditeur de listes dans le module configuration

L'édition est similaire à celle mise en œuvre pour les tables. Deux cases à cocher supplémentaires sont cependant disponibles.

La première, nommée « libellés fixes » permet de verrouiller l'éditeur de liste. C'est utilisé dans le cas où le contenu de la liste ne doit pas changer. Cette option a été ajoutée pour sécuriser les scripts basés sur la valeur de ces libellés.

La deuxième option libellée « Ajout automatique » conditionne le comportement des contrôles de saisie. Lorsque cette option est activée, une valeur saisie qui ne figure pas dans la liste pourra être ajoutée automatiquement après un message de confirmation. Cette approche autorise la gestion des listes de moindre importance et ne nécessite pas l'intervention d'une personne autorisée à lancer le mode configuration.

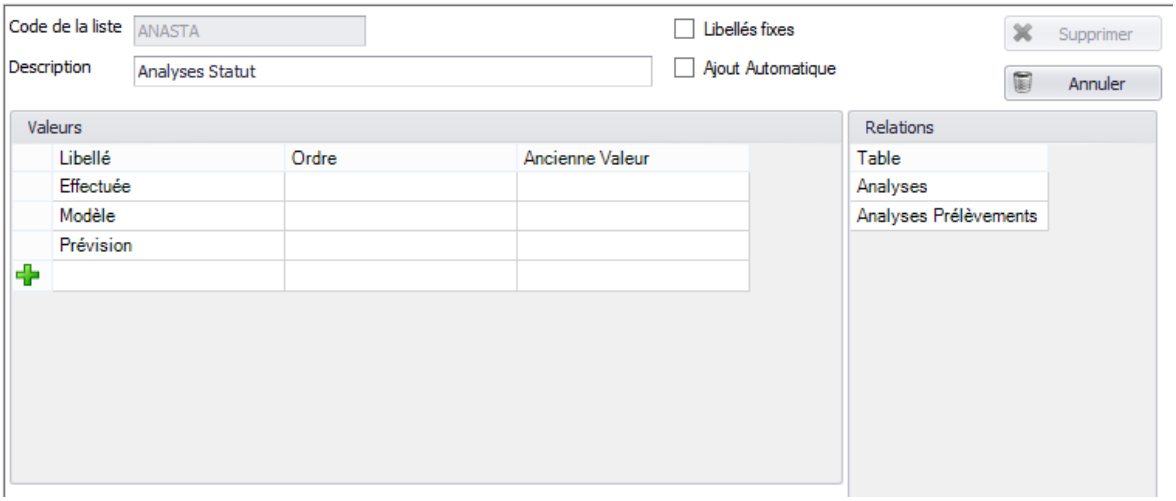


Figure 90 - Edition d'une liste

5.1.3. Editeur d'écran

L'éditeur d'écran a pour objectif de construire les menus, les écrans de recherche et les fiches de manière interactive. Les écrans sont spécifiques à un module si bien que pour ajouter ou modifier des écrans il faut aller en dessous du module concerné dans le mode configuration. La liste des écrans y est déjà présente pour la gestion des droits. L'accès à l'éditeur a simplement été ajouté par un menu contextuel comme présenté sur la Figure 91.

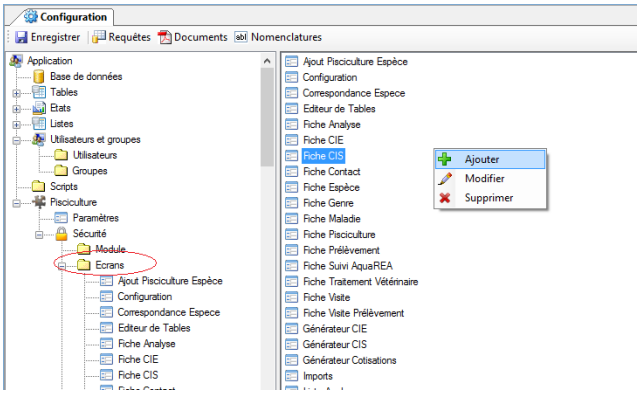


Figure 91 - Emplacement de l'éditeur d'écran dans le module configuration

L'éditeur se présente sous la forme d'une fiche puisqu'il a lui-même été réalisé avec la bibliothèque d'outils. Il est divisé en 4 parties visibles sur la Figure 92.

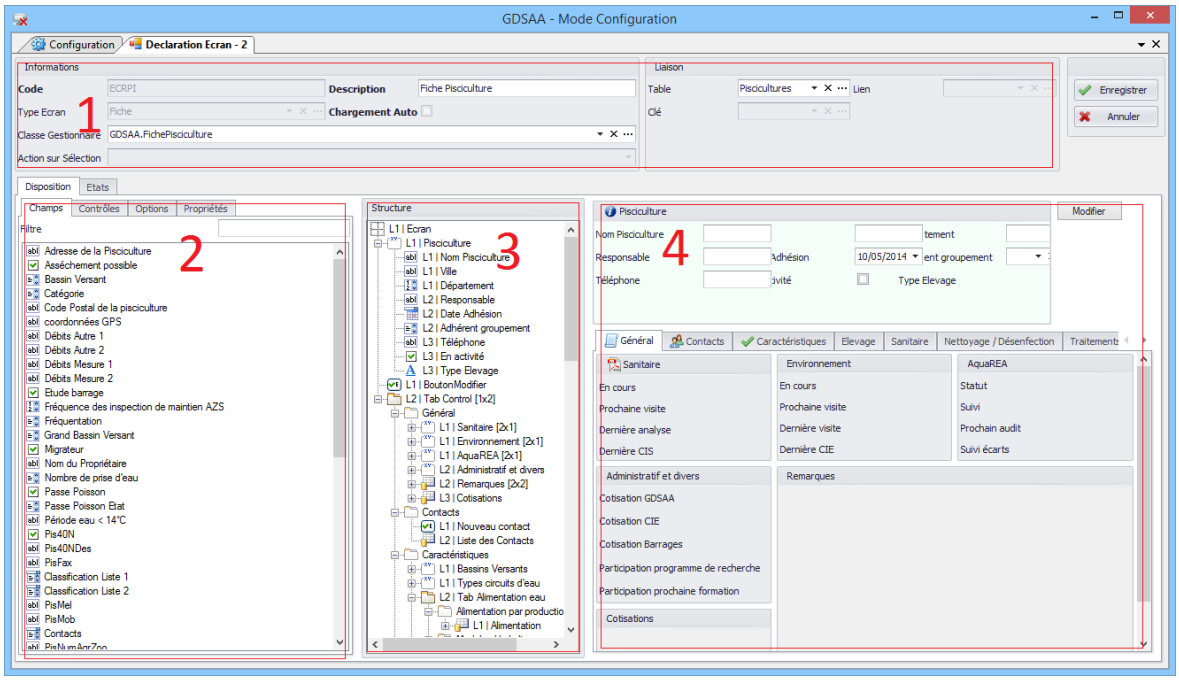


Figure 92 - Editeur d'écran

Partie 1 : L'en-tête comporte le code de l'écran nécessaire pour pouvoir s'y référer dans les scripts. La description est le nom visible de l'écran dans le mode configuration. Le type permet de choisir entre les différents modèles d'écran disponibles. Il y a la fiche, l'écran de recherche, le menu ou encore l'écran simple. La classe gestionnaire permet de spécifier quel type d'instance d'objet défini dans les scripts sera créé pour personnaliser le comportement de l'écran lors de l'exécution de l'application en mode utilisateur. Les autres propriétés dépendent du type d'écran sélectionné comme le montre le tableau suivant :

Tableau 30 - Zones disponibles par type dans l'éditeur d'écran

Type	Table	Lien	Chargement auto	Action sur sélection
Ecran de recherche	X	X		X
Fiche	X			
Menu			X	
Simple				

Partie 2 : Des onglets donnent accès aux éléments disponibles pour construire l'écran. Le premier affiche les champs de la table spécifiée dans l'en-tête qui ne sont pas encore utilisés dans l'écran. Une zone de filtre permet de réduire la liste pour retrouver plus facilement un champ. Il suffit ensuite de faire glisser le champ dans la structure de la partie 3. Le type de contrôle associé au champ est déterminé automatiquement selon son type. Le deuxième onglet nommé Contrôles laisse la possibilité d'ajouter des contrôles qui

ne sont pas liés à des champs. La liste des contrôles disponibles est présentée par la Figure 93.

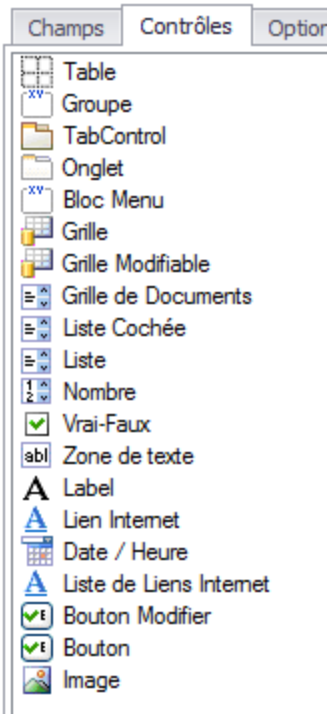


Figure 93 - Types de contrôles visuels disponibles

L'onglet Options autorise l'affichage de la partie 4 dans une fenêtre autonome.

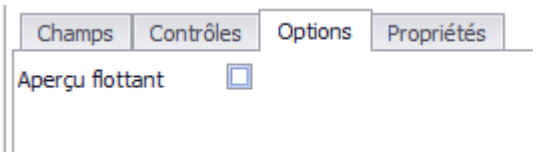


Figure 94 - Options d'affichage de l'aperçu dans l'éditeur d'écran

Enfin l'onglet Propriétés permet de modifier les propriétés de chacun des éléments (voir Figure 68) qui définissent la structure de l'écran en cours de modification. L'affichage est obtenu grâce au contrôle **PropertyGrid** de **WinForms**. Les propriétés publiques sont automatiquement affichées et associées au type d'éditeur correspondant. Des attributs peuvent être ajoutés aux propriétés des classes afin de personnaliser l'éditeur ou les descriptions affichées. Ainsi j'ai pu, par exemple, intégrer des boites de dialogue pour sélectionner la table associée à une grille.

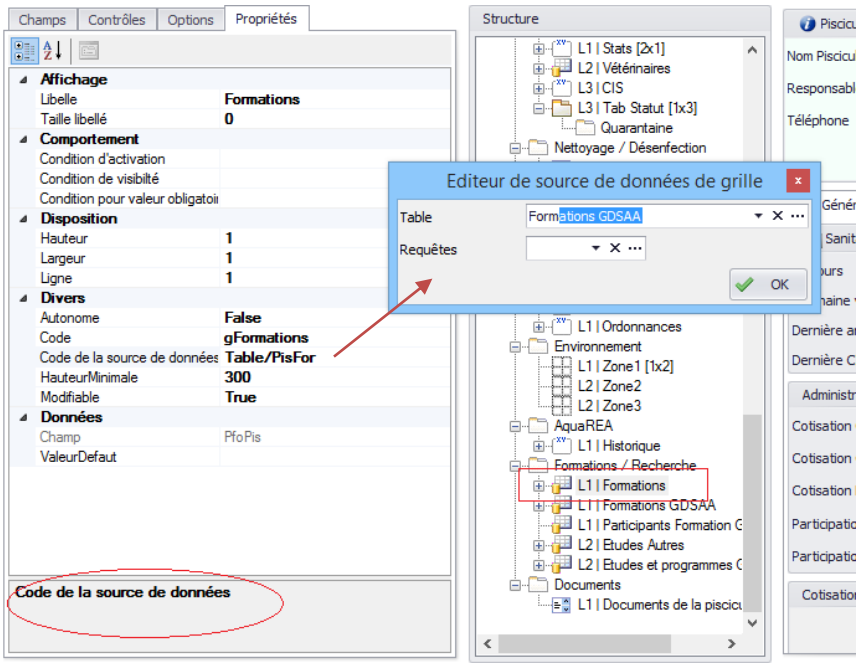


Figure 95 - Editeur d'élément écran

Partie 3 : La structure est définie de manière hiérarchique. Il est possible de déplacer librement les éléments depuis l'onglet champs ou de la structure elle-même. Des groupes entiers de contrôles peuvent être repositionnés vers un autre niveau hiérarchique. Le positionnement réel des contrôles est calculé comme suit :

Chaque élément de type conteneur dispose d'une zone découpée sous forme de table. Chaque élément possède une propriété **Ligne** qui indique à quelle ligne il doit être positionné dans son élément conteneur parent. Il n'y a pas de propriété colonne puisque cette donnée peut être déduite de la position de l'élément dans la structure hiérarchique. Pour affiner le positionnement, les propriétés **Largeur** et **Hauteur** permettent d'étendre un élément sur plusieurs lignes ou plusieurs colonnes. Des raccourcis clavier aident au positionnement. Les touches + et – du clavier vont incrémenter ou décrémenter la propriété **Ligne** et corriger si nécessaire celle des éléments impactés. Visuellement, le nom des éléments est préfixé par « Lx | » ou x est la valeur de la propriété **Ligne**. Un préfixe est également ajouté sous la forme « [Largeur x Hauteur] » si les valeurs sont différentes de 1.

Partie 4 : La zone d'aperçu affiche l'écran tel qu'il sera construit lors de l'exécution en mode utilisateur. La routine de construction de l'aperçu est la même que celle utilisée pour le

chargement réel de l'écran. A chaque modification de la structure ou de propriété d'un élément, l'aperçu est mis à jour. L'option d'aperçu flottant permet de visualiser l'écran avec les dimensions réelles afin de mieux prévoir l'agencement des zones.

5.1.4. Editeur de code pour les scripts

Dans les programmes paramétrables, les scripts sont généralement sous forme de fichiers textes dans un langage de programmation interprété au moment de l'exécution. La notion de scripts dans la PDR est différente. Le code source est effectivement enregistré sous forme de fichiers au format texte. La différence réside dans le fait que le code est compilé comme un programme C# traditionnel pour produire un assembly au format MSIL. Cette DLL sera alors chargée dynamiquement au prochain démarrage de l'application en mode utilisateur. Il est possible donc de mettre l'intégralité du code source sous forme de scripts. Seul doit rester dans le ficher EXE une classe contenant la fonction de démarrage **main**() qui démarre la PDR avec la classe *AppliAIS*. Nous allons voir comment l'éditeur de script permet de créer, modifier, vérifier, compiler, activer ou désactiver un script.

5.1.4.1. Edition des fichiers de code source

La gestion des fichiers de scripts se fait dans le mode configuration grâce à l'entrée de menu Scripts en dessous de celle de la gestion des utilisateurs et des groupes. Le panneau de gestion de la partie droite contient une liste des scripts existants. Les scripts sont simplement l'ensemble des fichiers présents dans le même dossier que le fichier exécutable de l'application dont l'extension est « .script » ou « .script_Inactive ».

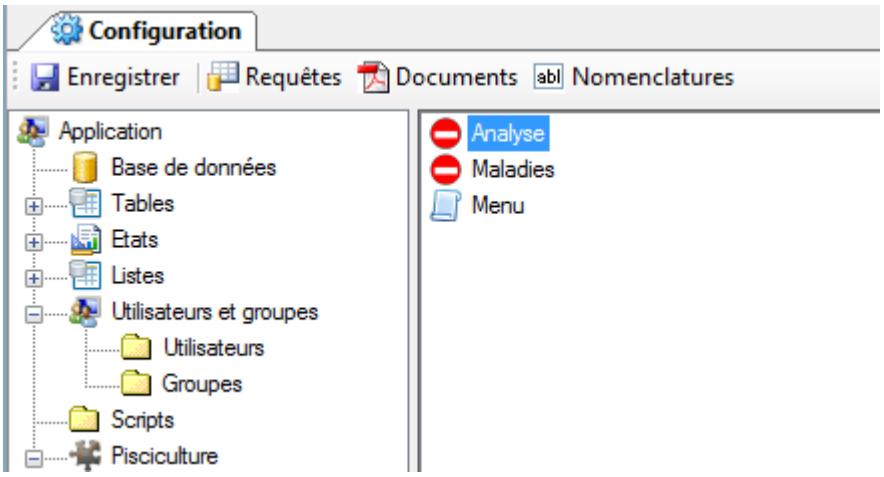


Figure 96 - Emplacement de l'éditeur de scripts dans le module configuration

Un menu contextuel propose les actions Ajouter, Modifier ou Supprimer. Le choix de l'option Ajouter ouvre une fenêtre qui demande le nom du fichier de script à créer ainsi qu'une description. Une fois le choix effectué, l'éditeur s'ouvre et charge le nouveau fichier qui ne contient que la description choisie sous forme de ligne de commentaire complétée par la date de création du fichier.

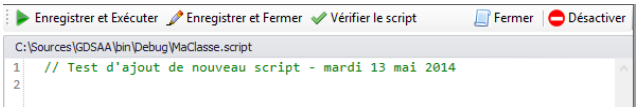


Figure 97 - Nouveau script dans l'éditeur de scripts

L'éditeur comprend une zone de saisie, avec la prise en charge de la coloration syntaxique spécifique de C#, et une barre d'outils avec les commandes suivantes :

- **Enregistrer et Exécuter** :
- **Enregistrer et Fermer**
- **Vérifier le script**
- **Fermer**
- **Désactiver**

La commande **Fermer** ferme l'éditeur sans enregistrer les modifications éventuelles puis affiche la liste des scripts existants.

La commande **Désactiver** est une sorte de case à cocher sous forme de bouton. Si elle est sélectionnée lors de l'enregistrement des modifications, le fichier sera enregistré et renommé avec l'extension « script_Inactive ». Il sera alors exclu lors de la compilation.

La commande **Vérifier le script** provoque une compilation en mémoire uniquement de l'ensemble des scripts actifs pour détecter des problèmes de syntaxe (voir 5.1.4.2).

La commande **Enregistrer et Exécuter** effectue l'enregistrement et la compilation puis recopie l'ensemble des fichiers de l'application dans un dossier de test afin d'exécuter l'application recopiée avec les modifications des scripts actives.

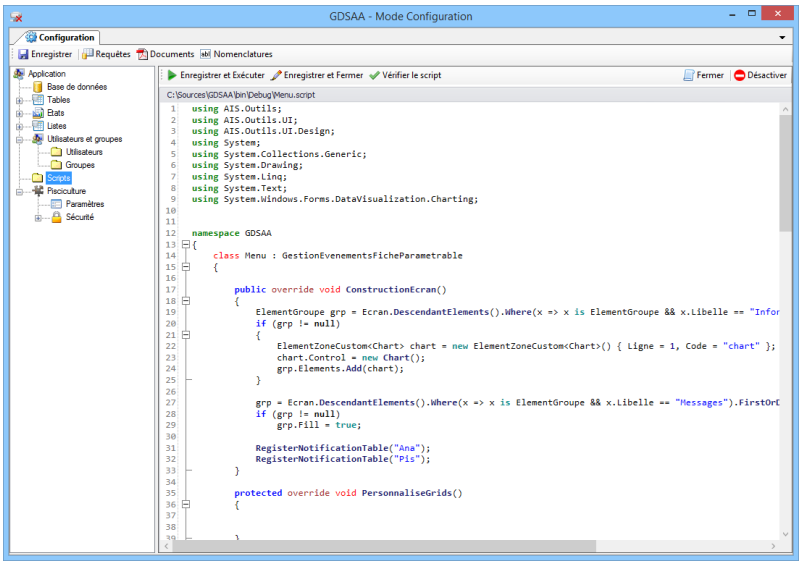


Figure 98 - Edition de scripts

5.1.4.2. Mise au point et génération du fichier binaire

Avant chaque enregistrement, une archive du script en cours d'édition est effectuée dans un sous dossier avec l'ajout d'un suffixe précisant la date et l'heure à la seconde près. L'éditeur prend en charge la fonction d'annulation de saisie classique mais il est ainsi possible de revenir en arrière même en cas de multiples sessions de modifications.

A la suite d'une compilation, l'éditeur met en évidence les lignes comportant une erreur de syntaxe. La description de l'erreur est fournie en positionnant le curseur d'édition sur la ligne en erreur.



Figure 99 - Affichage des erreurs de compilation dans les scripts

5.1.4.3. Chargement dynamique des classes

Quel que soit le nombre de scripts actifs au moment de la compilation, un texte comprenant la définition de la classe ScriptLoader est ajouté. Cette classe permettra à

l'application de provoquer le chargement de l'assembly au moment voulu. Le chargement des *assembly* se fait sur la plateforme .Net lors de la première instanciation d'une classe. Ce comportement laisse l'occasion de vérifier la présence d'un fichier scripts.dll plus récent et de le remplacer puisque l'ancien n'est pas encore chargé et donc verrouillé par le système d'exploitation. Cette approche ne fonctionne pas si plusieurs instances de l'application sont déjà en cours d'exécution. Un message indique alors qu'une mise à jour est en attente jusqu'au prochain démarrage et qu'il faut fermer si possible toutes les instances ouvertes.

5.2. Exemples d'écrans de l'application

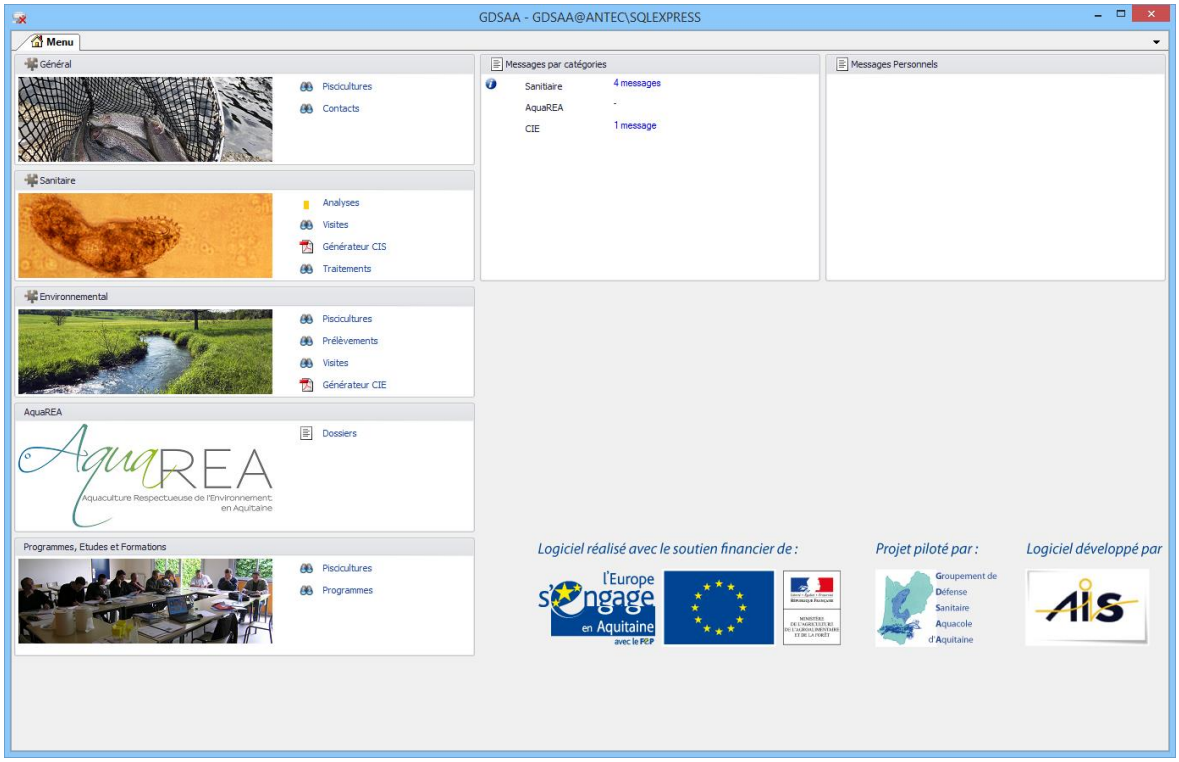


Figure 100 - Ecran de menu

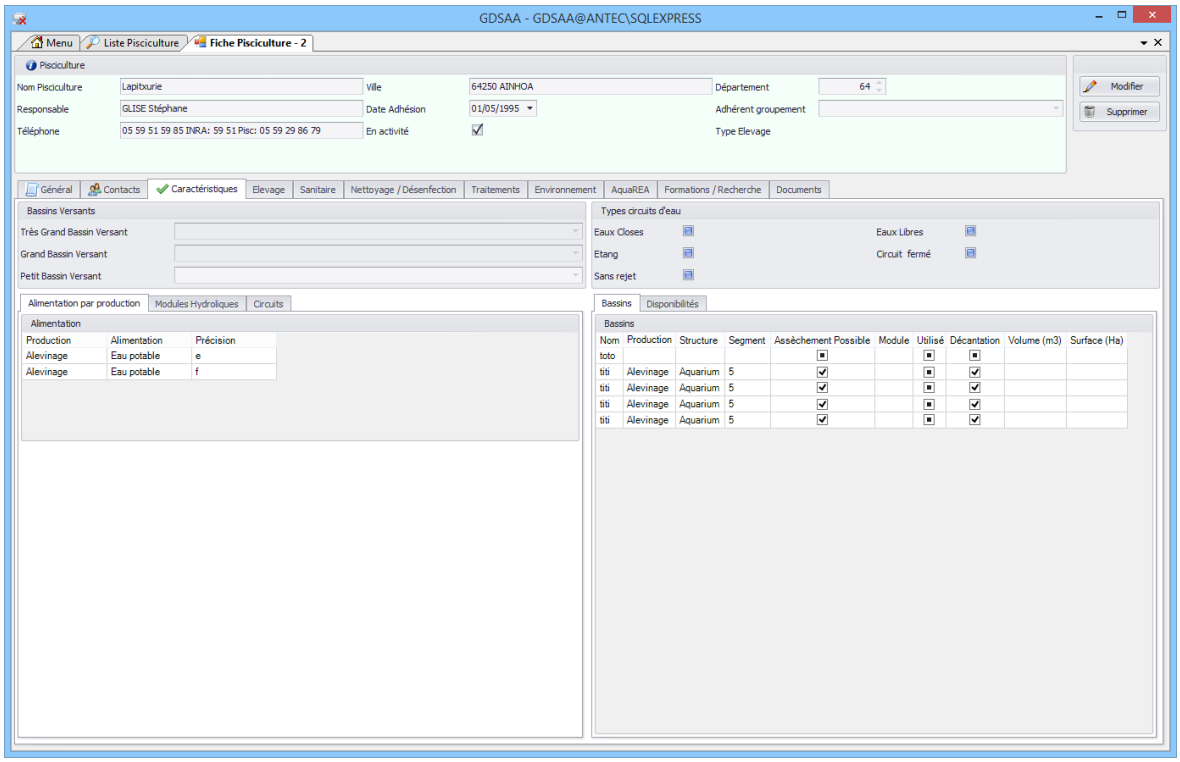


Figure 101 - La fiche pisciculture

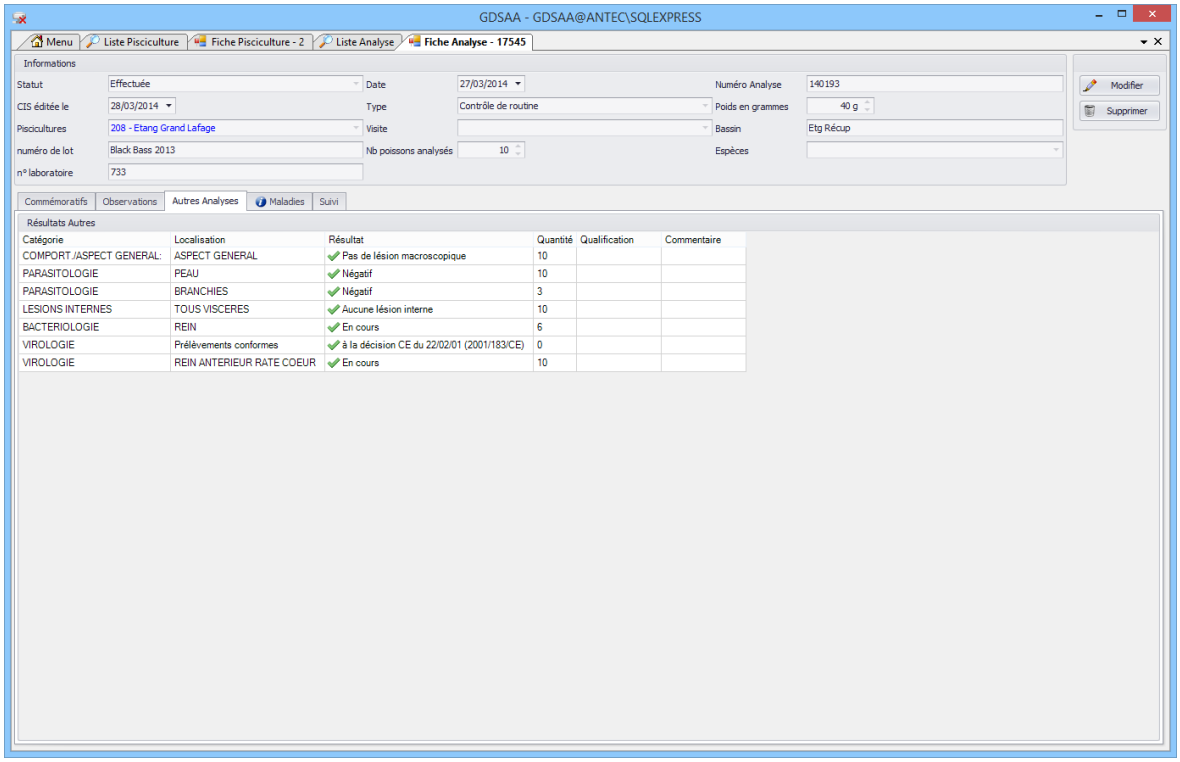


Figure 102 - La fiche analyse sanitaire

6. Conclusion

Ce mémoire devait répondre à de multiples problématiques de la société AIS dans le domaine du développement d'application informatique de gestion.

La première d'entre elles est l'amélioration des outils et des méthodes de réalisation de logiciel. Pour répondre à des besoins spécifiques, comme la capacité de l'utilisateur final à modifier une application en production, j'ai proposé le développement d'une plateforme de développement rapide en interne. Le travail a été effectué en 3 phases :

La première phase concerne la mise en place d'une méthode de gestion de projet. Le choix de la méthode SCRUM s'est avéré pertinent. Cette méthode est proche des habitudes puisque les développeurs avaient déjà une grande liberté. La sécurisation du code source avec un gestionnaire de code source est une amélioration importante pour la société. La conception et la réalisation de la plateforme de développement a permis de mettre en œuvre l'ensemble de mes connaissances acquises durant ma formation au CNAM et lors de mes emplois précédents.

La deuxième phase constitue une mise à l'épreuve des outils sur un projet interne qui constitue de ce fait un risque faible. L'objectif était de gagner suffisamment de temps sur la réalisation de la partie paramétrage afin de tenir les délais de livraison de l'application. Le déploiement a été effectué dans les temps.

La dernière phase est l'amélioration des outils pour apporter la capacité de manipulation des outils par l'utilisateur final. Cela correspond à un ensemble d'éditeurs graphiques pour manipuler le schéma de la base de données, le comportement et le contenu de l'application. La stratégie générale était de passer du temps sur des points techniques en attendant d'obtenir des informations de plus en plus précises sur les écrans et traitements à intégrer dans la deuxième version du logiciel **POISSONS**. Une fois les éditeurs opérationnels, des rendez-vous réguliers chez le client ont permis, par du simple paramétrage, de concrétiser plus facilement la vision du client de sa nouvelle application. Cela a grandement facilité le dialogue entre les différents intervenants.

A titre personnel, je retire une grande satisfaction d'avoir pu gérer en toute autonomie ces projets tant sur le plan technique que sur le plan organisationnel. Les solutions que j'ai proposées ont pu répondre aux besoins identifiés en collaboration avec

les clients tout au long des projets. J'ai également apprécié diriger et accompagner les différents stagiaires qui sont intervenus dans les projets. Il reste encore de nombreuses étapes à franchir avant la livraison de la nouvelle version de **POISSONS** mais je ne doute pas que ce projet arrivera à son terme.

Annexes

Annexe 1 Type de données SQL Server 119
Annexe 2 : Principes sous-jacents au manifeste 120
Annexe 3 : Code C# pour importer les utilisateurs depuis Active Directory 121
Annexe 4 : Configuration de IceScrum avec SQL Server 123
Annexe 5 : Exécution de requête SQL en C# 123
Annexe 6 : Initialisation d'une application avec support du débogage optionnel 125
Annexe 7 : Implémentation d'un écran de recherche personnalisé 126

Annexe 1 Type de données SQL Server

	Type	Taille	précision	Min	Max	Description	Nouveau 2000	Nouveau 2005	Nouveau 2008	Type déprécié	Type UDT .Net	Compressible ?	Indexable ?	Type Variable ?	Equiv. Type ANSI	Remarque
Nombres entiers																
bit	-	1	1	0	1	Booléen						X	X		BIT	Si plusieurs champs bit existent, SQL Server stocke jusqu'à 8 bits par octets
tinyint	-	1	1	0	255	Nombre entier non signé 8 bits							X		-	
smallint	-	2	1	-32 768	32 767	Nombre entier signé 16 bits						X	X		SMALLINT	
int	-	4	1	-2 147 483 648	2 147 483 647	Nombre entier signé 32 bits						X	X		INTEGER	
bigint	-	8	1	- 9 223 372 036 854 775 808	9 223 372 036 854 775 807	Nombre entier signé 64 bits	X					X	X		BIGINT	
Nombres décimaux																
decimal(p, s)	1 ≤ p < 10	5	1 x 10^{-s}	-99 999 999 999 999 999 999 999 999 999 999 999 999	99 999 999 999 999 999 999 999 999 999 999 999 999	Nombre à virgule de précision exacte							X		DECIMAL / NUMERIC	
numeric(p, s)	10 ≤ p < 20	9	1 x 10^{-s}													
	20 ≤ p < 29	13	1 x 10^{-s}													
	29 ≤ p ≤ 38	17	1 x 10^{-s}													
	vardecimal	2 à 20	1 x 10^{-s}									X		X		Cette compression existe depuis SQL Server 2005 Service Pack 2
smallmoney	-	4	0,0001	-214 748,3648	214 748,3647	Nombre à virgule de précision exacte						X	X		-	C'est un type int divisé par 10000
money	-	8	0,0001	- 922 337 203 685 477,5808	922 337 203 685 477,5807	Nombre à virgule de précision exacte						X	X		-	C'est un type bigint divisé par 10000
float(n)	1 ≤ n < 25	4	max 1 x 10^{38}	~ -3,4028234 x 10^{38}	~ 3,4028234 x 10^{38}	Nombre à virgule flottante approché						X			REAL / FLOAT	Type non précis, sa précision maximale est atteinte proche de 0, à déconseiller pour des applications financières
	25 ≤ n ≤ 53	8	max 1 x 10^{308}	~ -1,7976931348623157 x 10^{308}	~ 1,7976931348623157 x 10^{308}							X			DOUBLE PRECISION	Type non précis, sa précision maximale est atteinte proche de 0, à déconseiller pour des applications financières
real	-	4	max 1 x 10^{38}	~ -3,4028234 x 10^{38}	~ 3,4028234 x 10^{38}	Nombre à virgule flottante approché						X			REAL	Type non précis, sa précision maximale est atteinte proche de 0, à déconseiller pour des applications financières
Date et Heure																
smalldatetime	-	4	1 min	01/01/1900 00:00	06/06/2079 23:59	Date et Heure						X	X		TIMESTAMP	
datetime	-	8	3,33 ms	01/01/1753 00:00:00.000	31/12/9999 23:59:59.997	Date et Heure						X	X		TIMESTAMP	
date	-	3	1 jour	01/01/0001	31/12/9999	Date						X	X		DATE	
time(n)	0 ≤ n < 3	3	1 x 10^{-n} sec	00:00:00	23:59:59	Heure			X				X		TIME	Le type time sans taille est en fait un time(7)
	3 ≤ n < 5	4	1 x 10^{-n} sec													
	5 ≤ n ≤ 7	5	1 x 10^{-n} sec													
datetime2(n)	0 ≤ n < 3	6	1 x 10^{-n} sec	01/01/0001 00:00:00	31/12/9999 23:59:59	Date et Heure			X				X		TIMESTAMP	Le type datetime2 sans taille est en fait un datetime2(7)
	3 ≤ n < 5	7	1 x 10^{-n} sec													
	5 ≤ n ≤ 7	8	1 x 10^{-n} sec													
datetimeoffset(n)	0 ≤ n < 3	8	1 x 10^{-n} sec	01/01/0001 00:00:00 -X	31/12/9999 23:59:59 +X	Date, Heure et fuseau horaire			X				X		TIMESTAMP WITH TIME ZONE	Le type datetimeoffset sans taille est en fait un datetimeoffset(7)
	3 ≤ n < 5	9	1 x 10^{-n} sec													
	5 ≤ n ≤ 7	10	1 x 10^{-n} sec													
Chaînes de caractères																
char(n)	-	n	1 car.	n caractères	8000 caractères	Chaîne de caractère de longueur fixe n caractères							X		CHARACTER	Jeu de caractères définie par COLLATE, la taille est multipliée par 2 en cas d'utilisation d'un jeu de caractère le nécessitant
varchar(n)	-	2 + r	1 car.	0 caractères	8000 caractères	Chaîne de caractère de longueur variable							X	X	CHARACTER VARYING	Jeu de caractères définie par COLLATE, la taille est multipliée par 2 en cas d'utilisation d'un jeu de caractère le nécessitant
varchar(max)	-	2 + r	1 car.	0 caractères	2 milliard de car.	Chaîne de caractère de longueur variable long		X					X	X	CHARACTER LARGE OBJECT	Jeu de caractères définie par COLLATE, la taille est multipliée par 2 en cas d'utilisation d'un jeu de caractère le nécessitant
text	-	2 + r	1 car.	0 caractères	2 milliard de car.	Chaîne de caractère de longueur variable long				X				X	CHARACTER LARGE OBJECT	Jeu de caractères définie par COLLATE, la taille est multipliée par 2 en cas d'utilisation d'un jeu de caractère le nécessitant
nchar(n)	-	n x 2	1 car.	n caractères	4000 caractères	Chaîne de caractère de longueur fixe type unicode							X		NATIONAL CHARACTER	Les caractères sont stockés sur 2 ou 4 octets suivant le jeu de caractère auquel il appartient
nvarchar(n)	-	2 + r x 2	1 car.	0 caractères	4000 caractères	Chaîne de caractère de longueur variable type unicode							X	X	NATIONAL CHARACTER VARYING	Les caractères sont stockés sur 2 ou 4 octets suivant le jeu de caractère auquel il appartient
nvarchar(max)	-	2 + r x 2	1 car.	0 caractères	1 milliard de car.	Chaîne de caractère de longueur variable type unicode long		X					X	X	NATIONAL CHARACTER LARGE OBJECT	Les caractères sont stockés sur 2 ou 4 octets suivant le jeu de caractère auquel il appartient
ntext	-	2 + r x 2	1 car.	0 caractères	1 milliard de car.	Chaîne de caractère de longueur variable type unicode long				X				X	NATIONAL CHARACTER LARGE OBJECT	Les caractères sont stockés sur 2 ou 4 octets suivant le jeu de caractère auquel il appartient
Valeurs binaires																
binary(n)	-	n	1 octet	n octets	8000 octets	Binaire de longueur fixe							X		-	
varbinary(n)	-	2 + r	1 octet	0 octets	8000 octets	Binaire de longueur variable							X	X	BINARY VARYING	
varbinary(max)	-	2 + r	1 octet	0 octets	2 Go	Binaire de longueur variable long		X					X	X	BINARY LARGE OBJECT	
varbinary(max)	filestream	2 + r	1 octet	0 octets	> 2 Go	Binaire de longueur variable, stocké dans le système de fichier			X					X	BINARY LARGE OBJECT	Stockage du contenu sur le système de fichier et permet d'outrepasser les limites classiques des champs binaires.
image	-	2 + r	1 octet	0 octets	2 Go	Binaire de longueur variable long				X				X	BINARY LARGE OBJECT	
Divers																
sql_variant	-	max 8ko	N/A	N/A	N/A	Type variant pouvant être de n'importe quel type de moins de 8ko	X						X	X	-	Tous les types SQL excepté : text, ntext, image, varchar(max), nvarchar(max), varbinary(max), timestamp et UDT
timestamp	not null	8	1 octet	0x0000000000000000	0xFFFFFFFFFFFFFFFF	Binaire representant les modifications de l'enregistrement en s'incrémentant									ROWVERSION	Binaire de 8 octets incrémenté à chauge modification de l'enregistrement dans lequel il se trouve. Sa valeur est unique au sein de la base de données
	null	2 + r	1 octet	0x	0xFFFFFFFFFFFFFFFF											
uniqueidentifier	-	16	1 octet	00000000-0000-0000-0000-000000000000	FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFFFFFF	Type GUID	X						X		-	Généré par la fonction NEWID(). Depuis SQL Server 2005 la fonction NEWSEQUENTIALID() est également utilisable mais uniquement dans une valeur par défaut. La probabilité d'apparition d'un double est quasiment nulle.
hierarchyid	-	max 2Go	N/A	N/A	N/A	Type hierarchique permettant de représenter un arbre			X		X			X	-	Dispose de méthodes permettant la manipulation de l'arborescence
xml	-	2 + r x 2	1 car.	0 caractères	1 milliard de car.	Type XML		X					X	X	XML	Type nvarchar(max) validant que son contenu est bien de l'XML bien formé. Peut être indexé à l'aide d'index XML.
Spatial																
geography	-	max 2Go	N/A	N/A	N/A	Type spatial sur sphère					X		X	X	-	Peut être indexé à l'aide d'un index spatial
geometry	-	max 2Go	N/A	N/A	N/A	Type spatial sur plan					X		X	X	-	Peut être indexé à l'aide d'un index spatial
Types définis par l'utilisateur																
UDT (alias SQL)	-	max 8ko	N/A	N/A	N/A	Alias de type SQL, crée par sp_addtype ou CREATE TYPE				X					-	
UDT .net	-	max 2Go	N/A	N/A	N/A	Type .net contenu dans une assembly .net, crée par CREATE TYPE			X		X			X	-	Les .Net peuvent occuper plus de 8ko à partier de SQL Server 2008
UDT .net	-	max 8ko	N/A	N/A	N/A			X			X			X	-	

r représente le nombre de caractères ou d'octets utilisé par le champ

Source : http://blogs.codes-sources.com/christian

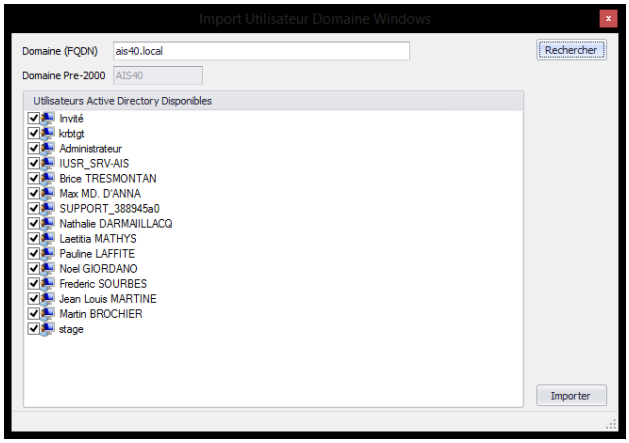
Annexe 2 : Principes sous-jacents au manifeste

(source : http://agilemanifesto.org/iso/fr/principles.html)

Nous suivons ces principes:

- Notre plus haute priorité est de satisfaire le client en livrant rapidement et régulièrement des fonctionnalités à grande valeur ajoutée.
- Accueillez positivement les changements de besoins, même tard dans le projet. Les processus Agiles exploitent le changement pour donner un avantage compétitif au client.
- Livrez fréquemment un logiciel opérationnel avec des cycles de quelques semaines à quelques mois et une préférence pour les plus courts.
- Les utilisateurs ou leurs représentants et les développeurs doivent travailler ensemble quotidiennement tout au long du projet.
- Réalisez les projets avec des personnes motivées. Fournissez-leur l'environnement et le soutien dont ils ont besoin et faites-leur confiance pour atteindre les objectifs fixés.
- La méthode la plus simple et la plus efficace pour transmettre de l'information à l'équipe de développement et à l'intérieur de celle-ci est le dialogue en face à face.
- Un logiciel opérationnel est la principale mesure d'avancement.
- Les processus Agiles encouragent un rythme de développement soutenable. Ensemble, les commanditaires, les développeurs et les utilisateurs devraient être capables de maintenir indéfiniment un rythme constant.
- Une attention continue à l'excellence technique et à une bonne conception renforce l'Agilité.
- La simplicité – c'est-à-dire l'art de minimiser la quantité de travail inutile – est essentielle.
- Les meilleures architectures, spécifications et conceptions émergent d'équipes autoorganisées.
- À intervalles réguliers, l'équipe réfléchit aux moyens de devenir plus efficace, puis règle et modifie son comportement en conséquence.

Annexe 3 : Code C# pour importer les utilisateurs depuis Active Directory



L'objectif de cet outil est de présenter la liste des utilisateurs qui sont dans Active Directory mais qui ne sont pas encore dans la table des utilisateurs. Pour fonctionner il faut déterminer dans un premier temps le nom interne du domaine Windows. On retrouve cette information en effectuant une requête dans la base de données Active Directory. Il faut utiliser les classes fournies par le framework .Net : *DirectoryEntry* et *DirectorySearcher*

```
private string GetDomaine(string FQDN)
{
    string ldapUrl = "LDAP://" + FQDN;
    try
    {
        ldapUrl = ldapUrl.Insert(7, "CN=Partitions,CN=Configuration,DC=").Replace(".", ",DC=");
        DirectoryEntry Ldap = new DirectoryEntry(ldapUrl);
        DirectorySearcher searcher = new DirectorySearcher(Ldap);
        searcher.SearchScope = SearchScope.Subtree;
        searcher.PropertiesToLoad.Add("cn");
        searcher.Filter = "nETBIOSName=*";
        SearchResultCollection Users = searcher.FindAll();
        foreach (SearchResult anUser in Users)
        {
            DirectoryEntry User = anUser.GetDirectoryEntry();
            return User.Properties["cn"].Value.ToString();
        }
    }
    catch { }
    return "DOMAINE?";
}
```

Une fois l'information obtenue, elle est affichée dans la zone de texte nommée *tDomaine*. Un clic sur le bouton **Rechercher** lance une autre requête dans la base de l'Active Directory afin de déterminer les comptes utilisateurs qui ne sont pas déjà dans notre table _Uti. Si au moins un résultat est trouvé, le bouton **Importer** est activé.

```
// Connexion à Active Directory avec le compte courant
string ldapUrl = "LDAP://" + tDomaine.Text;
try
{
    DirectoryEntry Ldap = new DirectoryEntry(ldapUrl);

    DirectorySearcher searcher = new DirectorySearcher(Ldap);
    searcher.Filter = "(&(objectClass=user)(objectCategory=Person))";
    SearchResultCollection Users = searcher.FindAll();
    treeView1.Nodes.Clear();

    List<string> actifs = new List<string>();
    DataTable DTA = ManagerDataBase.GetDataTable("select UtiNT from _Uti where UtiNT is not NULL");
    foreach (DataRow row in DTA.Rows)
        if (!actifs.Contains(row.Field<string>("UtiNT")))
            actifs.Add(row.Field<string>("UtiNT"));

    string DOMAINE = GetDomaine(tDomaine.Text);
    t2000.Text = DOMAINE;
    foreach (SearchResult anUser in Users)
    {
        DirectoryEntry User = anUser.GetDirectoryEntry();
        if (!actifs.Contains(DOMAINE + "\\" + User.Properties["sAMAccountName"].Value.ToString()))
        {
            TreeNode node = new TreeNode(User.Properties["cn"].Value.ToString());
            node.Tag = DOMAINE + "\\"  +User.Properties["sAMAccountName"].Value.ToString();
            node.Checked = true;
            treeView1.Nodes.Add(node);
        }

    }
    if (treeView1.Nodes.Count > 0)
        bImport.Enabled = true;
}
catch
{
    treeView1.Nodes.Clear();
    t2000.Text = "";
    MessageBox.Show("Impossible de trouver les informations dans Active Directory", "Recherche", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

L'ajout des utilisateurs sélectionnés est simplement ajouté dans la table comme le montre l'extrait de code suivant :

```
private void bImport_Click(object sender, EventArgs e)
{
    DataTableEx Uti = ManagerDataBase.GetDataTable("select * from _Uti where Uti Is Null");

    foreach (TreeNode node in treeView1.Nodes)
    {
        if (node.Checked)
        {
            DataRow row = Uti.NewRow();
            row["UtiNom"] = node.Text;
            row["UtiNT"] = node.Tag.ToString();
            Uti.Rows.Add(row);
        }
    }
    Uti.Enregistre();
    Close();
}
```

Annexe 4 : Configuration de IceScrum avec SQL Server

Télécharger le pilote SQL server pour java à l'adresse http://www.microsoft.com/en-us/download/details.aspx?displaylang=en&id=11774 et copier le fichier sqljdbc4.jar dans le dossier tomcat\lib du server IceScrum. Il faut ensuite modifier le fichier de configuration config.groovy pour indiquer le driver JDBC ainsi que le serveur, la base de données SQL et les identifiants de connexion.

```
grails.serverURL="http://srv1-ais:8080/icescrum"

icescrum.baseDir="C:/Documents and Settings/Administrateur/Local Settings/Application
Data/Kagilum/iceScrum Server"

icescrum.log.dir="C:/Documents and Settings/Administrateur/Local Settings/Application
Data/Kagilum/iceScrum Server/logs/"

dataSource.driverClassName = "com.microsoft.sqlserver.jdbc.SQLServerDriver"

dataSource.url = "jdbc:sqlserver://srv1-
ais:1472;databaseName=IceScrum;integratedSecurity=false;"

dataSource.username = "sa"

dataSource.password = "xxxxxxxx"

icescrum.project.import.enable = false

icescrum.project.export.enable = false

icescrum.auto_follow_productowner = true

icescrum.auto_follow_stakeholder  = true

icescrum.auto_follow_scrummaster  = true

icescrum.alerts.errors.to = "frederic.dauba@gmail.com"

icescrum.alerts.subject_prefix = "[icescrum]"

icescrum.alerts.enable = true

icescrum.alerts.default.from = "frederic.dauba@gmail.com"

icescrum.attachments.enable = true

grails.mail.host = "smtp.gmail.com"

grails.mail.port = 587

grails.mail.from = "frederic.dauba@gmail.com"

grails.mail.username = "frederic.dauba@gmail.com"

grails.mail.password = "xxxxxxxxxxxxx"

grails.mail.props = ["mail.smtp.auth": "true",

        "mail.smtp.socketFactory.port": "587",

        "mail.smtp.starttls.enable": "true",

        "mail.debug": "true"]

icescrum.debug.enable = false
```

Annexe 5 : Exécution de requête SQL en C#

L'extrait de code suivant présente l'utilisation des classes fournies par le framwork.Net pour effectuer un chargement de données en mode déconnecté.

La première étape consiste à définir les paramètres de la connexion dans la propriété ConnectionString de l'objet SqlConnection. La classe SqlConnectionStringBuilder nous aide à construire une chaîne avec la bonne syntaxe.

La deuxième étape est la définition de la requête avec ses éventuels paramètres. Dans l'exemple, on utilise le paramètre « id » pour fournir la valeur de filtre sur le champ « Ana ». La requête est initialisée avec une référence vers la connexion.

Enfin la classe SqlDataAdapter déclenche l'exécution de la requête et stocke le résultat dans une table mémoire matérialisée par la classe DataTable. L'ouverture et la fermeture de la connexion sont gérées automatiquement par le SqlDataAdapter.

```
SqlConnection con = new SqlConnection();
SqlConnectionStringBuilder builder = new SqlConnectionStringBuilder();
builder.DataSource = "ANTEC\\SQLEXPRESS";
builder.IntegratedSecurity = true;
builder.InitialCatalog = "GDSAA";

con.ConnectionString = builder.ToString();

SqlCommand cmd = new SqlCommand("select * from Ana where Ana = @id", con);
cmd.Parameters.Add(new SqlParameter("id", 1234));

SqlDataAdapter DA = new SqlDataAdapter(cmd);
DataTable DT = new DataTable();

DA.Fill(DT);
```

Annexe 6 : Initialisation d'une application avec support du débogage optionnel

L'attribut **Conditionnal** permet de spécifier que la méthode est définie uniquement si le symbole « DEBUG » est présent au moment de la compilation. Si ce n'est pas le cas, tous les appels à cette méthode sont ignorés.

```
[Conditional("DEBUG")]
private void DebugEnableFisrtChanceExceptionWatch()
{
    AppDomain.CurrentDomain.FirstChanceException += CurrentDomain_FirstChanceException;
}

[Conditional("DEBUG")]
public static void DebugConsole(string message)
{
    Console.WriteLine(message);
}

void CurrentDomain_FirstChanceException(object sender, System.Runtime.ExceptionServices.FirstChanceExceptionEventArgs e)
{
    if (e.Exception.InnerException == null)
        AppliAIS.DebugConsole("[DEBUG] " + AppDomain.CurrentDomain.FriendlyName + " - "
            + e.Exception.Message + "\r\n" + e.Exception.StackTrace);
    else
        AppliAIS.DebugConsole("[DEBUG] " + AppDomain.CurrentDomain.FriendlyName + " - "
            + e.Exception.Message + "\r\n" + e.Exception.InnerException.Message + "\r\n" + e.Exception.StackTrace);
}
```

Une application qui utilise des formulaires graphiques ne possède pas de fenêtre de sortie textuelle (Console MS-DOS). Il est possible de la créer afin de visualiser les messages d'erreurs comme le montre l'extrait de code suivant :

```
public AppliAIS(string titre, string[] args, string regKey, bool authWindows = false, bool NoStartup = false,
{
    VersionDataBase = 1;
    DebugEnableFisrtChanceExceptionWatch();

    if (!NoStartup &&  System.Windows.Input.Keyboard.Modifiers == System.Windows.Input.ModifierKeys.Control)
    {
        try
        {
            Win32.AllocConsole();
            Console.BackgroundColor = ConsoleColor.DarkBlue;
            Console.ForegroundColor = ConsoleColor.White;
            Console.SetWindowSize(Console.LargestWindowWidth / 2, Console.LargestWindowHeight / 2);
            Console.Clear();
        }
        catch { }
    }
```

Annexe 7 : Implémentation d'un écran de recherche personnalisé

La personnalisation d'un écran de recherche nécessite au minimum la surcharge de la méthode **ObtientRequete**() afin de spécifier la requête SQL de chargement de la grille de résultat. On peut définir lors de cette étape les colonnes à afficher avec la méthode **SetAlias**().

L'extrait de code suivant montre également la surcharge de la méthode **PersonnaliseGrid**() appelée après la création du contrôle grille. Dans cet exemple, la colonne « AnaImg » est choisie comme source de l'image de la colonne « Stade ».

```
public class RechercheAnalyse : GestionEvenementsEcranRecherche
{
    19 références
    public override void ConstructionEcran()
    {
        base.ConstructionEcran();
    }

    3 références
    public override IdentificationAction AvantActionSurSelection(DataRow row, Grid.LinkEventArgs e)
    {
        return base.AvantActionSurSelection(row, e);
    }
    20 références
    public override RequeteInfo ObtientRequete(string table, string pkey)
    {
        SetAlias("AnaNumAna", "N°");
        SetAlias("Date", "Date");
        SetAlias("Stade", "Stade");
        SetAlias("PisDes", "Pisciculture");
        SetAlias("AnaObs", "Observations");
        KeyLinkName = "AnaNumAna";

        Fiche.ActionAjouter = Pisciculture.Instance.Actions["CREATEANA"];

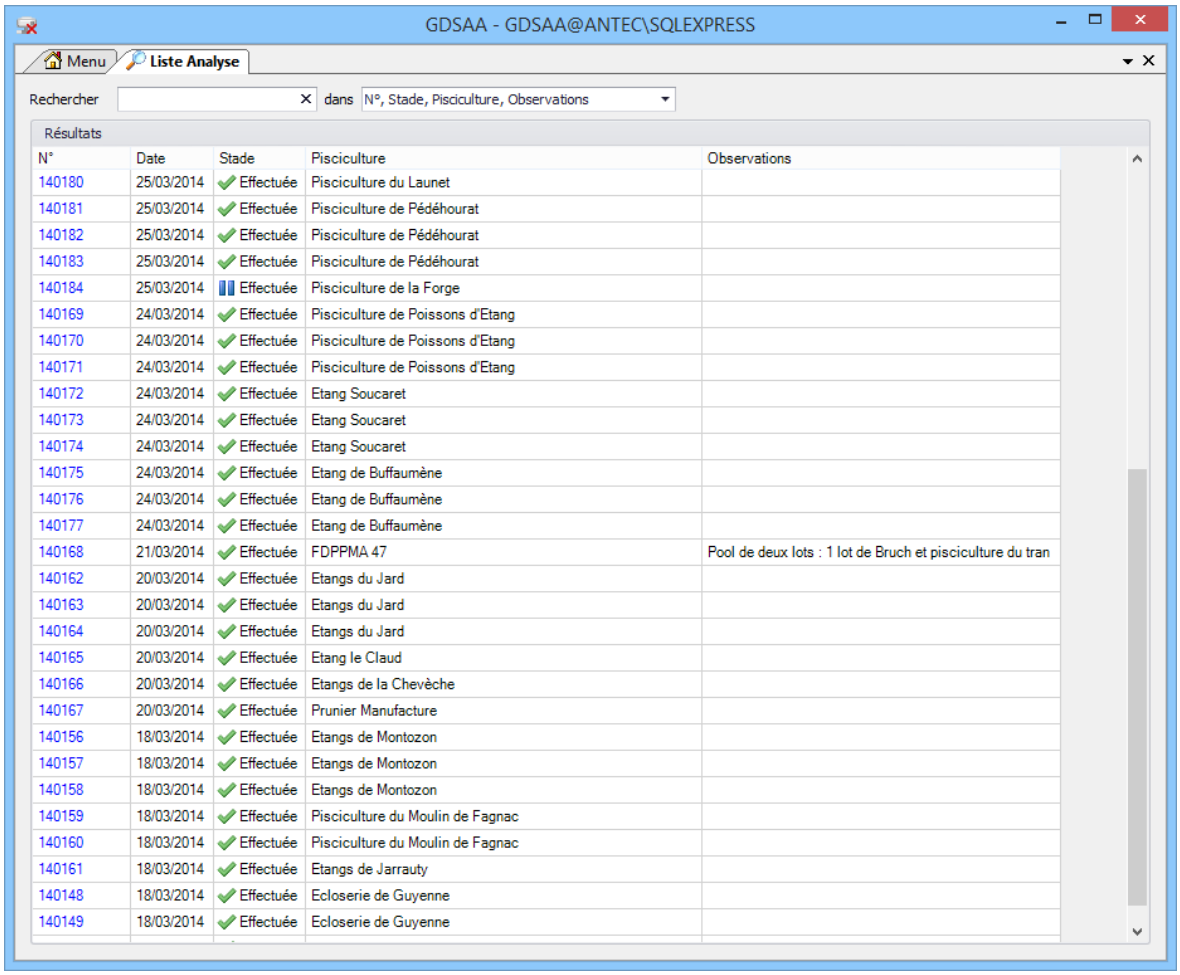
        return new RequeteInfo()
        {
            MaxRow = 50,
            Sql = @"Ana,AnaNumAna,AnaDat , AnaDat Date ,ParDes Stade  , PisDes, AnaObs,
            case    when ParDes = 'Prévision' then 'images\\icon_sablier.gif'
                    when ParDes='Modèle' then 'images\\parametres.png'
                    else case
                        when AliAna is not Null then 'images\\pause.png'
                        else 'images\\check.png'
                    end
            end AnaImg
            from Ana
            inner join ParANASTA on Par = Anasta
            left outer join Pis on AnaPis = Pis

            left outer join (select distinct AliAna from AnaLig where AliSta != 0) W on W.AliAna = Ana",
            OrderBy = "AnaDat Desc "
        };
    }

    10 références
    public override void PersonnaliseGrid(Grid grid)
    {
        grid.SetColumnImage("Stade", "AnaImg");

    }

}
```



N°	Date	Stade	Pisciculture	Observations
140180	25/03/2014	Effectuée	Pisciculture du Launet	
140181	25/03/2014	Effectuée	Pisciculture de Pédéhourat	
140182	25/03/2014	Effectuée	Pisciculture de Pédéhourat	
140183	25/03/2014	Effectuée	Pisciculture de Pédéhourat	
140184	25/03/2014	Effectuée	Pisciculture de la Forge	
140169	24/03/2014	Effectuée	Pisciculture de Poissons d'Etang	
140170	24/03/2014	Effectuée	Pisciculture de Poissons d'Etang	
140171	24/03/2014	Effectuée	Pisciculture de Poissons d'Etang	
140172	24/03/2014	Effectuée	Etang Soucaret	
140173	24/03/2014	Effectuée	Etang Soucaret	
140174	24/03/2014	Effectuée	Etang Soucaret	
140175	24/03/2014	Effectuée	Etang de Buffaumène	
140176	24/03/2014	Effectuée	Etang de Buffaumène	
140177	24/03/2014	Effectuée	Etang de Buffaumène	
140168	21/03/2014	Effectuée	FDPPMA 47	Pool de deux lots : 1 lot de Bruch et pisciculture du tran
140162	20/03/2014	Effectuée	Etangs du Jard	
140163	20/03/2014	Effectuée	Etangs du Jard	
140164	20/03/2014	Effectuée	Etangs du Jard	
140165	20/03/2014	Effectuée	Etang le Claud	
140166	20/03/2014	Effectuée	Etangs de la Chevèche	
140167	20/03/2014	Effectuée	Prunier Manufacture	
140156	18/03/2014	Effectuée	Etangs de Montozon	
140157	18/03/2014	Effectuée	Etangs de Montozon	
140158	18/03/2014	Effectuée	Etangs de Montozon	
140159	18/03/2014	Effectuée	Pisciculture du Moulin de Fagnac	
140160	18/03/2014	Effectuée	Pisciculture du Moulin de Fagnac	
140161	18/03/2014	Effectuée	Etangs de Jarrauty	
140148	18/03/2014	Effectuée	Ecloserie de Guyenne	
140149	18/03/2014	Effectuée	Ecloserie de Guyenne	

Bibliographie

[1] MESSAGER V., 2013 ; Gestion de projet agile. Eyrolles, Paris, 278 p.

[2] AUBRY C., 2013, SCRUM Le guide pratique de la méthode agile la plus populaire, DUNOD, Paris, 302 p.

[3] CNRS, Suivi des risques d'un projet, [En ligne]. http://www.dsi.cnrs.fr/conduite-projet/phasedefinition/qualite/risques/guide-suivi-risques.pdf, juillet 2013

[4] SourceForge, DockPanel Suite [En ligne]. http://sourceforge.net/projects/dockpanelsuite/, juillet 2013

[5] CodePlex, SourceGrid, http://sourcegrid.codeplex.com/, juillet 2013

[6] DevExpress, https://www.devexpress.com/products/net/controls/winforms/editors/, juillet 2013

[7] AvalonEdit, http://www.nuget.org/packages/AvalonEdit/, novembre 2013

Liste des figures

Figure 1 - Gouvernance du CERFRANCE 6
Figure 2 - Organigramme AIS Informatique 7
Figure 3 - Architecture technique de AIS 8
Figure 4 - Planning prévisionnel du projet CNAM 12
Figure 5 - Cycle de vie simplifié de Scrum 14
Figure 6 - Classification des méthodes agiles selon leur degré de formalisme et leurs itérations .. 15
Figure 7 - Une release et ses sprints 17
Figure 8 - Activités en parallèle des sprints 18
Figure 9 - Composition d'une story 19
Figure 10 - Les quatre types de story 20
Figure 11 - Etapes d'une story 21
Figure 12 - Les bacs à story 21
Figure 13 - Exemple de Burndown chart 22
Figure 14 - Etats des stories dans la planification 23
Figure 15 - Architecture du Framework .Net 24
Figure 16 - Schéma de compilation et d'exécution d'un programme C# 25
Figure 17 - Visual Studio 2013 Ultimate 26
Figure 18 - Gestionnaire SQL Management Studio 29
Figure 19 - SQL Server Report Builder 3.0 30
Figure 20 - Visual SVN Server 31
Figure 21 - Tortoise SVN 31
Figure 22 - Configuration de Tortoise SVN 32
Figure 23 - Interface IceSCRUM 32
Figure 24 - Eléments de la boite à outils 33
Figure 25 - Réseau local classique 35
Figure 26 - Etapes de définition de la vision du produit 36
Figure 27 - Carte heuristique 37
Figure 28 - Maquette de l'écran de menu 39
Figure 29 - Maquette d'un écran de type liste 39
Figure 30 - Maquette d'un écran de type fiche 40
Figure 31 - Exemple de convention de nommage des tables 42
Figure 32 - Diagramme des classes utiles pour un chargement de données 45
Figure 33 - Diagramme des classes utiles pour l'enregistrement de données 46
Figure 34 - Digramme de classes du gestionnaire de base de données 48

Figure 35 - Composants d'une application 50
Figure 36 - Initialisation d'une application 51
Figure 37 - Exemple de configuration dans la base de registre 52
Figure 38 - Modèle d'application MDI 53
Figure 39 - Diagramme de classe simplifié d'une Fiche 54
Figure 40 - Diagramme de classe du gestionnaire de fenêtres 54
Figure 41 - Diagramme des classes Identification 56
Figure 42 - Composition d'une application en modules 57
Figure 43 - Diagramme de classe Module 57
Figure 44 - Classes pour l'écran de configuration 58
Figure 45 - Tables de configuration des utilisateurs et des groupes 61
Figure 46 - Cas d'utilisation du mode Configuration 61
Figure 47 - Gestion des utilisateurs dans le menu de configuration 62
Figure 48 - Affichage des utilisateurs et des groupes 62
Figure 49 - Menu contextuel de gestion des utilisateurs 62
Figure 50 - Ecran de saisie d'un utilisateur 63
Figure 51 - Ecran d'affectation de groupe 63
Figure 52 - Menu contextuel de gestion de groupe (écran utilisateur) 63
Figure 53 - Gestion des groupes 64
Figure 54 - Ecran de saisie de groupe 64
Figure 55 - Tables de déclaration des écrans et des autorisations 66
Figure 56 - Exemple de définition d'autorisations 66
Figure 57 - Diagramme de classes des identifications d'écrans 67
Figure 58 - Schéma de l'écran de menu 67
Figure 59 - Diagramme de classes de gestion du menu principal 68
Figure 60 - Contrôle Grille personnalisé 69
Figure 61 - Mode édition du contrôle Grille 69
Figure 62 - Diagramme de classes de gestion d'écran de recherche 70
Figure 63 - Extrait de code ExecuteAction() du gestionnaire de module 71
Figure 64 - Schéma d'une zone contrôle avec libellé 72
Figure 65 - Diagramme de classes pour les contrôles avec libellé intégré 73
Figure 66 - Liaison de données 74
Figure 67 - Changement d'états d'une fiche 75
Figure 68 - Diagramme de classes des définitions d'éléments des écrans 80
Figure 69 - Diagramme de classe DataBaseAera 81
Figure 70 - Diagramme de classe général des Ecrans 82

Figure 71 - Etapes de fonctionnement du système de notification 82
Figure 72 - Diagramme de classe du gestionnaire de notifications 85
Figure 73 - Les acteurs du système de gestion des jours de repos 89
Figure 74 - Cas d'utilisations 89
Figure 75 - Architecture technique de la solution de GESTIONCP 90
Figure 76 - Diagramme de séquence de dépense de jour de congé 94
Figure 77 - Schéma relationnel de la base de GESTIONCP 97
Figure 78 - Menu de l'application de paramétrage 98
Figure 79 - Fiche exercice et périodes 98
Figure 80 - Fiche exercice et quotas 99
Figure 81 - Jours spéciaux 99
Figure 82 - Ecran de modification du calendrier pour l'administrateur 100
Figure 83 - Emplacement de l'éditeur de tables dans le module configuration 102
Figure 84 - Ecran d'ajout de table 102
Figure 85 - Modification des champs d'une table 103
Figure 86 - Edition des relations d'une table 104
Figure 87 - Modification directe du contenu d'une table 105
Figure 88 - Schéma d'une table de liste 105
Figure 89 - Emplacement de l'éditeur de listes dans le module configuration 106
Figure 90 - Edition d'une liste 107
Figure 91 - Emplacement de l'éditeur d'écran dans le module configuration 107
Figure 92 - Editeur d'écran 108
Figure 93 - Types de contrôles visuels disponibles 109
Figure 94 - Options d'affichage de l'aperçu dans l'éditeur d'écran 109
Figure 95 - Editeur d'élément écran 110
Figure 96 - Emplacement de l'éditeur de scripts dans le module configuration 111
Figure 97 - Nouveau script dans l'éditeur de scripts 112
Figure 98 - Edition de scripts 113
Figure 99 - Affichage des erreurs de compilation dans les scripts 113
Figure 100 - Ecran de menu 114
Figure 101 - La fiche pisciculture 115
Figure 102 - La fiche analyse sanitaire 115

Liste des tableaux

Tableau 1 - Forces et faiblesses des principales méthodes agiles 16
Tableau 2 - Cycle de vie d'une application et élements de la PDR 34
Tableau 3 - Nomenclature des tables et des champs 42
Tableau 4 - Correspondance des types de données 43
Tableau 5 - Paramètres d'une application 52
Tableau 6 - Structure de la table _Dec 56
Tableau 7 - Droits standards enregistrés 65
Tableau 8 - Actions standards par type d'écran 65
Tableau 9 - Liste des classes avec libellé intégré 73
Tableau 10 - Structure de la table _EcrDef 78
Tableau 11 - Structure de la table de définition des colonnes _EcrCol 79
Tableau 12 - Messages utilisés dans les notifications 84
Tableau 13 - Structure de la table société (Soc) 91
Tableau 14 - Structure de la table site (Sit) 92
Tableau 15 - Structure de la table service (Ser) 92
Tableau 16 - Structure de la table jour (Day) 92
Tableau 17 - Structure de la table type de jour (Jou) 93
Tableau 18 - Structure de la table exercice (Exo) 93
Tableau 19 - Structure de la table période (Per) 93
Tableau 20 - Structure de la table évènement (Evt) 94
Tableau 21 - Structure de la table profil (Prf) 95
Tableau 22 - Structure de la table de liaison profil-jour (PrfJou) 95
Tableau 23 - Structure de la table règle (Reg) 95
Tableau 24 - Structure de la table de liaison profil-règle (PrfReg) 95
Tableau 25 - Structure de la table employé (Emp) 96
Tableau 26 - Structure de la table rôle (Rol) 96
Tableau 27 - Structure de la table de liaison employé-service (EmpSer) 96
Tableau 28 - Structure de la table de liaison employé-type de jour (EmpJou) 96
Tableau 29 - Structure de la table commentaire (Cmt) 97
Tableau 30 - Zones disponibles par type dans l'éditeur d'écran 108

Etude et mise en œuvre d'une plateforme de développement rapide au travers de 2 projets

Mémoire d'Ingénieur C.N.A.M., Saint Pierre du Mont 2014

RESUME

Le développement d'applications spécifiques de gestion qui utilisent une base de données relationnelle requiert de nombreuses tâches répétitives. La réalisation ou l'adoption d'un framework de développement permet au développeur de gagner en productivité. Cette étude concerne la réalisation de ce type d'outillage dont la particularité est d'être utilisable également par l'utilisateur final. Les applications basées sur ce framework seront entièrement modifiables après la mise en production.

Mots clés : framework, développement, paramétrable, utilisateur final.

SUMMARY

The development of specific management applications that use a relational database requires many repetitive tasks. Construction or adoption of a development framework allows developers to increase productivity. This study concerns the implementation of this type of framework which can be also used by the end user. Applications based on this framework will be fully customizable after the deployment phase.

Key words: framework, development, customizable, end user.