inetum
Positive digital flow

Afpa

Dossier technique

— **Christophe Burckard** (CDA 8 – session d'août 2022)

Projet RCPS

Applications de gestion des postes et structures du Ministère de l'Agriculture

MINISTÈRE
DE L'AGRICULTURE
ET DE LA SOUVERAINETÉ
ALIMENTAIRE
Liberté
Égalité
Fraternité

Tuteur de stage
Roger-Henri Tastayre

Référent AFPA
David Mendouze

Sommaire

Avertissement....4
Remerciements....5
Abstract....6
1. Introduction....7
2. Présentation de l'entreprise Inetum....8
2.1. Inetum, ex-Gfi....8
2.2. La région Sud-Ouest et l'agence toulousaine....9
3. Le client, Inetum et le projet RCPS....13
3.1. Le client : le Ministère de l'Agriculture....13
3.2. Inetum Toulouse et le Ministère de l'Agriculture....13
3.3. Exemples de projets gérés par l'équipe....14
3.4. Le projet RCPS....15
4. Prise en main du projet....17
4.1. Environnement technique....17
4.1.1. Base de données....17
4.1.2. Hébergement et versioning des sources....17
4.1.3. Framework Orion et dépendances....18
4.1.4. Environnement de Développement Intégré (IDE)....18
4.2. Processus et méthodes de travail....19
4.2.1. Les spécifications fonctionnelles détaillées....19
4.2.2. Demandes d'évolutions/corrections et portail *JIRAAF*....20
4.2.3. Échanges avec le client....22
4.2.4. Chiffrage par unité d'œuvre....22
4.2.5. Validation par le client et livraisons....24
4.2.6. Synthèse du processus....25
4.3. Introduction à Orion....25
5. Ma mission au sein du projet RCPS....30
5.1. Un lot de demandes d'évolutions chiffrées et validées....30
5.2. Fonctionnement des applications RCPS-webapp et agent-structures-renoirh....31
5.2.1. *B*ase et modèle de données rcps....31
5.2.2. RCPS-webapp....33
5.2.3. agent-structures-renoirh....36
5.3. Traitement d'une Jira spécifique à RCPS-webapp : RCP-138....37
5.3.1. Analyse, conception et chiffrage....37
5.3.2. Scripts SQL....40
5.3.3. Front-end....41
5.3.4. Code Java - Exemples....43
5.3.5. Bilan de la Jira....46
5.4. Tests et validation....47
5.4.1. Tests unitaires....47
5.4.2. SonarQube....49
5.5. Remarques et conclusion....50
6. Bilan et perspectives....52
Annexes....53
Annexe 1 : SFD – Exemple de Cas d'Utilisation (CU)....54
Annexe 2 : SFD – Exemples de règles de gestion....56

Annexe 3 : Tableau de chiffrage pour l'unité d'œuvre POLIS.....58
Annexe 4 : Modèle de données des tables t_orion_*.....59
Annexe 5 : Modèle de données des tables de référence.....60
Annexe 6 : Modèle de données des tables paramétrant la création de bordereaux.....61

Avertissement

Le Ministère de l'Agriculture et la société Inetum demeurent les seuls propriétaires des documents et données liés au projet RCPS. Aucune utilisation ou transfert des éléments constituant le projet RCPS (tels que la documentation fonctionnelle et technique, le code, les données) ne peut être effectué sans accord explicite écrit des parties prenantes.

Il a été validé auprès du tuteur de stage et chef de projet, M. Roger-Henri Tastayre, qu'aucune information à caractère confidentiel n'est divulguée dans le présent rapport technique.

Remerciements

Tout d'abord, je souhaite adresser mes remerciements chaleureux à mon tuteur de stage, **M. Roger-Henri Tastayre**, pour sa disponibilité, son professionnalisme, la confiance qu'il m'a témoignée et l'autonomie qu'il m'a donnée dans la réalisation de ma mission.

Un grand merci à toute l'équipe d'Inetum en charge du Ministère de l'Agriculture, et notamment à **M. Thierry Lagrange**, responsable technique, pour le temps passé à partager patiemment ses connaissances et son expérience tout en conservant son indéfectible humour, et à **M. Thomas Rialland**, référent pour le projet RCPS, et qui a su me guider dans les méandres parfois obscurs des nombreuses lignes de code du projet.

Je voudrais ensuite adresser mes remerciements à toute l'équipe de l'**AFPA de Balma** pour son accompagnement au cours de cette année de formation, et notamment à Pascal Dangu, Lana Poplavskaya et Philippe Viguier, formateurs, pour leur implication dans la transmission de leurs savoirs.

Merci également à mon camarade de stage Quentin Moneger et à tout le groupe de stagiaires CDA 8 de l'AFPA pour l'entraide sans faille, l'humour de chaque instant et les précieux moments de détente en fin de journées autour de réhydratants. De belles rencontres et une belle aventure ensemble, qui ne fait que débuter.

Abstract

1. Introduction

Après une formation d'ingénieur suivie de plus d'une quinzaine d'années à exercer différents métiers à travers le monde, j'ai décidé d'entamer une reconversion professionnelle dans le domaine du développement informatique et ai ainsi suivi la formation de **Concepteur Développeur d'Applications** à l'AFPA de Balma (31) de septembre 2021 à août 2022.

Dans ce cadre, trois mois sont consacrés à une **Période d'Application en Entreprise**, que j'ai effectuée au sein de la société Inetum à Toulouse, dans l'équipe dédiée aux projets du Ministère de l'Agriculture, du **16 mai au 12 août 2022**.

Le choix d'une **ESN** pour une première expérience professionnelle dans le monde du développement informatique s'est imposée de par la diversité des missions, technologies et compétences qu'une telle structure propose et mobilise, ainsi que la variété des profils que nous sommes amenés à y côtoyer. **Les montées en compétences sont multiples et immédiates**, aidées par des équipes solides et des moyens adaptés.

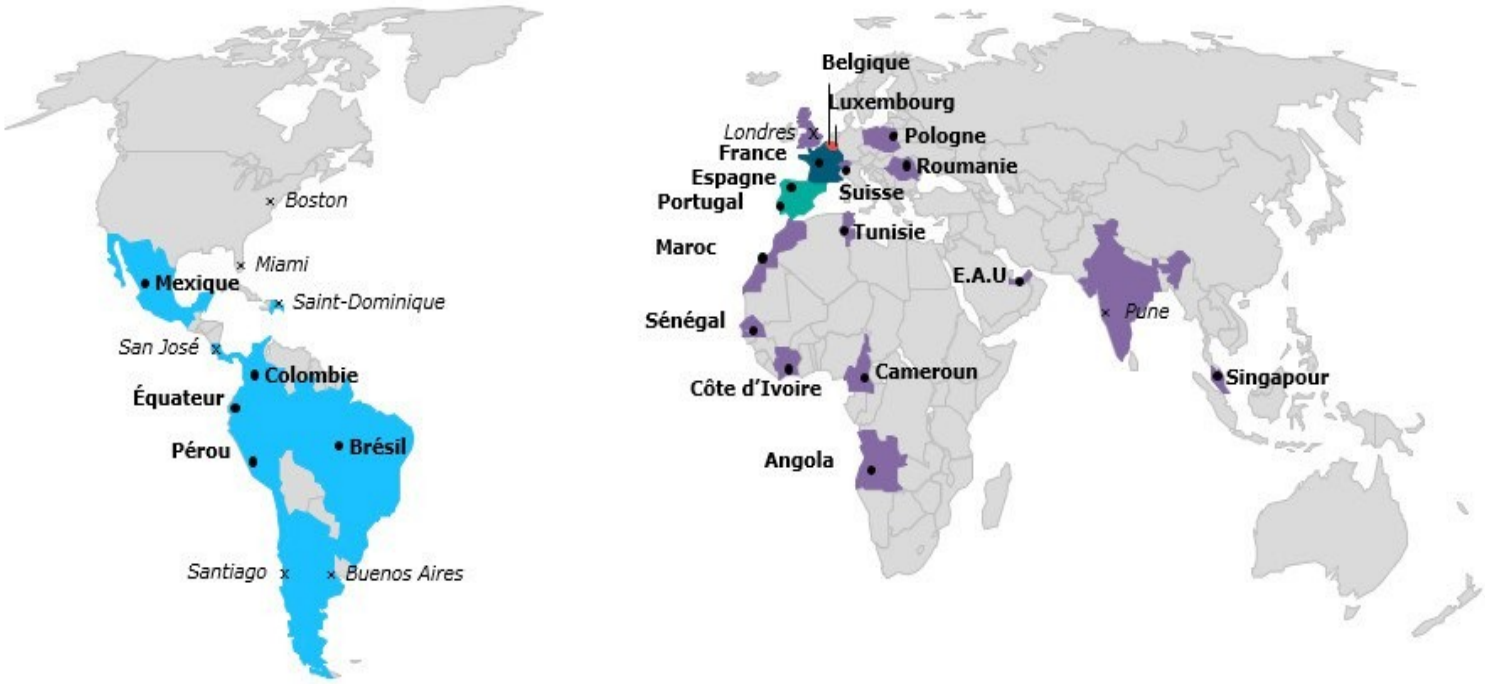
Les applications du **Ministère de l'Agriculture** sont codées quasiment exclusivement en **Java**, que ce soit via des frameworks réputés comme **Spring et Springboot**, ou un framework maison dont nous aurons l'occasion de reparler plus loin, **Orion**, basé sur JEE et JSF. Les bases de données sont essentiellement de type Postgresql. Les compétences mobilisées par ces projets couvrent aussi bien le front-end (via Orion et JSF), le back-end (via Orion, Spring et Springboot) et la modélisation et manipulation de bases de données. Le fonctionnement au forfait suppose les **chiffrages** en amont des tâches à réaliser et des échanges avec le client pour **affiner les besoins**. La mission est extrêmement formatrice quant à l'ensemble des processus de gestion au forfait d'applications auprès de clients.

2. Présentation de l'entreprise Inetum

2.1. Inetum, ex-Gfi

Inetum est une **Entreprise de Services du Numérique** (ESN) française de **dimension internationale**, présente en 2022 dans **26 pays**, en Europe, Afrique-Moyen-Orient, Amérique latine, États-Unis et Asie, et comptant environ **27 000 collaborateurs**.

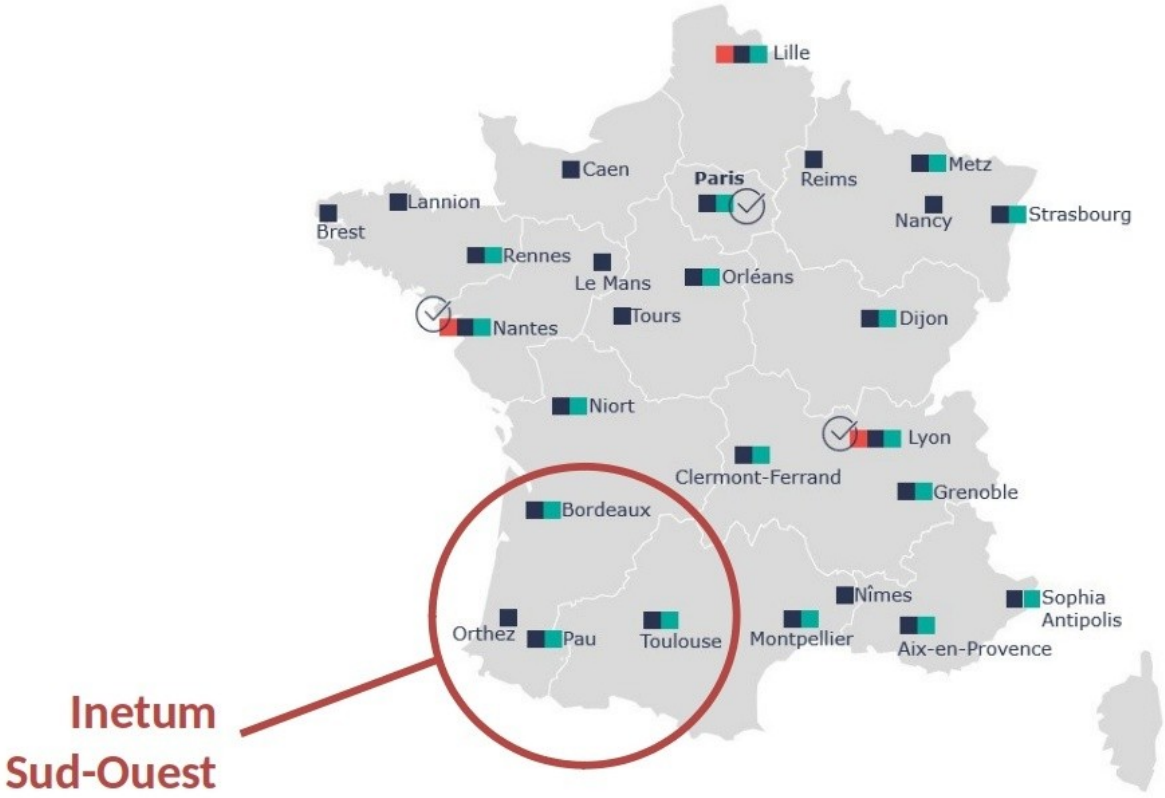
Créée au début des années 90 sous le nom de **Gfi** (« Groupe français d'informatique »), son expansion (aussi bien organique que par le biais de nombreuses acquisitions) et son internationalisation ont mené à un changement de nom en 2020. Le mot « Inetum » a été choisi, en référence au latin « Incrementum », qui signifie **croissance** et symbolise clairement les ambitions du groupe.



En 2021, Inetum a réalisé **2,2 milliards d'euros de chiffre d'affaires**, et vise a brève échéance la barre des 3 milliards.

Le leitmotiv de la société est reflété par son slogan (« **Positive digital flow** ») : aider les entreprises et institutions à « tirer le meilleur du digital flow », à soutenir leur transformation digitale dans un contexte de perpétuelle mutation, et ce en agilité.

En France



Inetum a des implantations sur l'ensemble du territoire français. Elle y réalise environ **40 % de son chiffre d'affaires** et y emploie **11 000 collaborateurs**.

2.2. La région Sud-Ouest et l'agence toulousaine

Inetum en région Sud-Ouest, ce sont **4 implémentations** :

- Toulouse (la principale)
- Bordeaux
- Pau
- Orthez

regroupant près de **1000 collaborateurs** pour un chiffre d'affaires de plus de **100 millions d'euros**.

Clients majeurs

Parmi les **clients principaux**, citons AIRBUS, EDF, la MSA, ORANGE, Navblue, des ministères (Agriculture, Éducation Nationale, Justice, …), la région Occitanie, le CNES, TOTAL, ARIANEGROUP, etc.

Secteurs d'activité

Ces entreprises représentent des **secteurs d'activité** aussi divers que :

- les services financiers
- l'industrie
- la santé
- les télécoms
- l'énergie
- la distribution
- les transports

Métiers

L'accompagnement proposé par Inetum s'articule autour de **5 métiers** :

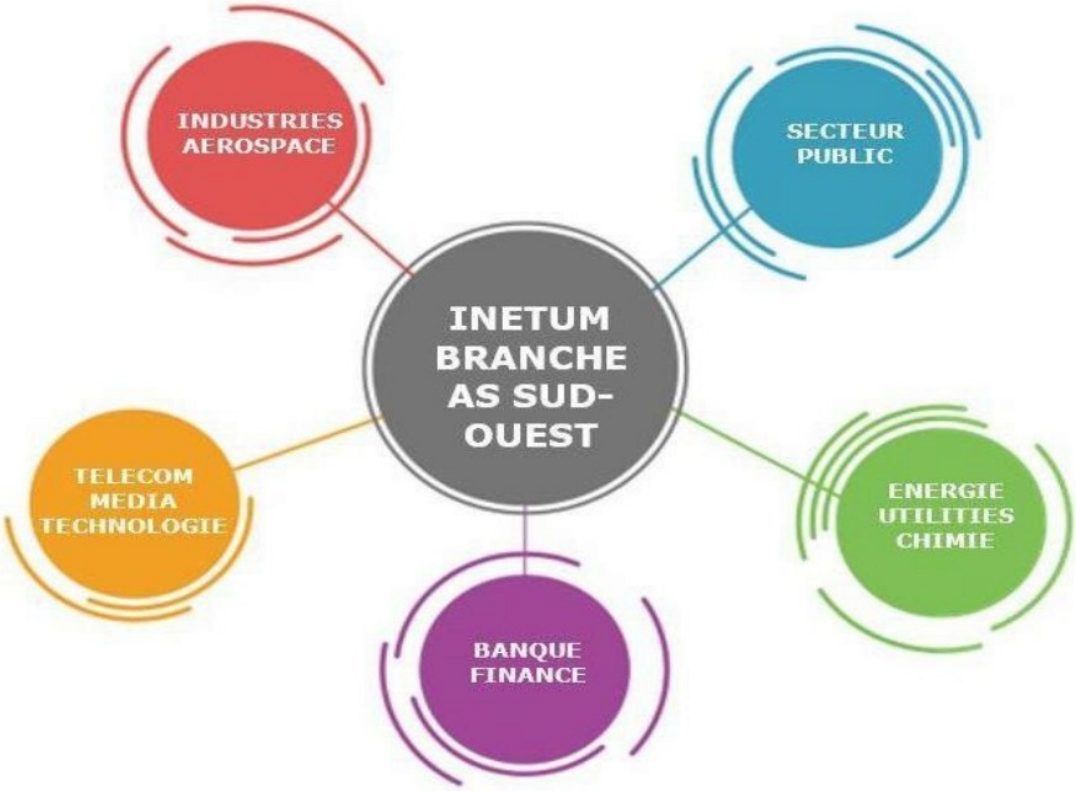
- Business transformation / consulting
- Application services
- Infrastructure services
- Business solutions
- Édition logiciels

Division Application Services Sud-Ouest (AS Sud-Ouest)

Ma période en entreprise s'est déroulée à **Toulouse** au sein de la division **Application Services (AS)**, qui compte **440 consultants** dans le Sud-Ouest pour un chiffre d'affaires de **40 millions d'euros**.

inetum
Positive digital flow

La division AS Sud-Ouest d'Inetum intervient dans **5 secteurs d'activités** majeurs :



3. Le client, Inetum et le projet RCPS

3.1. Le client : le Ministère de l'Agriculture

Nommé **Ministère de l'Agriculture et de la Souveraineté Alimentaire** (MASA) depuis mai 2022 et le nouveau gouvernement Borne, son activité couvre les domaines de l'agriculture, de la pêche, de l'alimentation et des forêts, ainsi que l'enseignement et la recherche liés à ces domaines. Parmi les missions menées par le ministère, nous pouvons citer :

- la sécurité alimentaire
- la formation et l'accès à l'emploi en milieu rural
- l'aménagement des territoires agricoles
- la qualité et la disponibilité de l'eau
- la gestion des espaces naturels et les questions environnementales
- le droit du travail agricole
- la protection sociale dans le domaine agricole
- les politiques d'échanges aux échelles européenne et internationale

L'administration centrale du ministère est basée à Paris et Toulouse avec un effectif d'environ 2000 agents, les services régionaux et départementaux employant pour leur part environ 13000 agents et l'enseignement technique et supérieur 20000.

Le pôle informatique du ministère de l'Agriculture

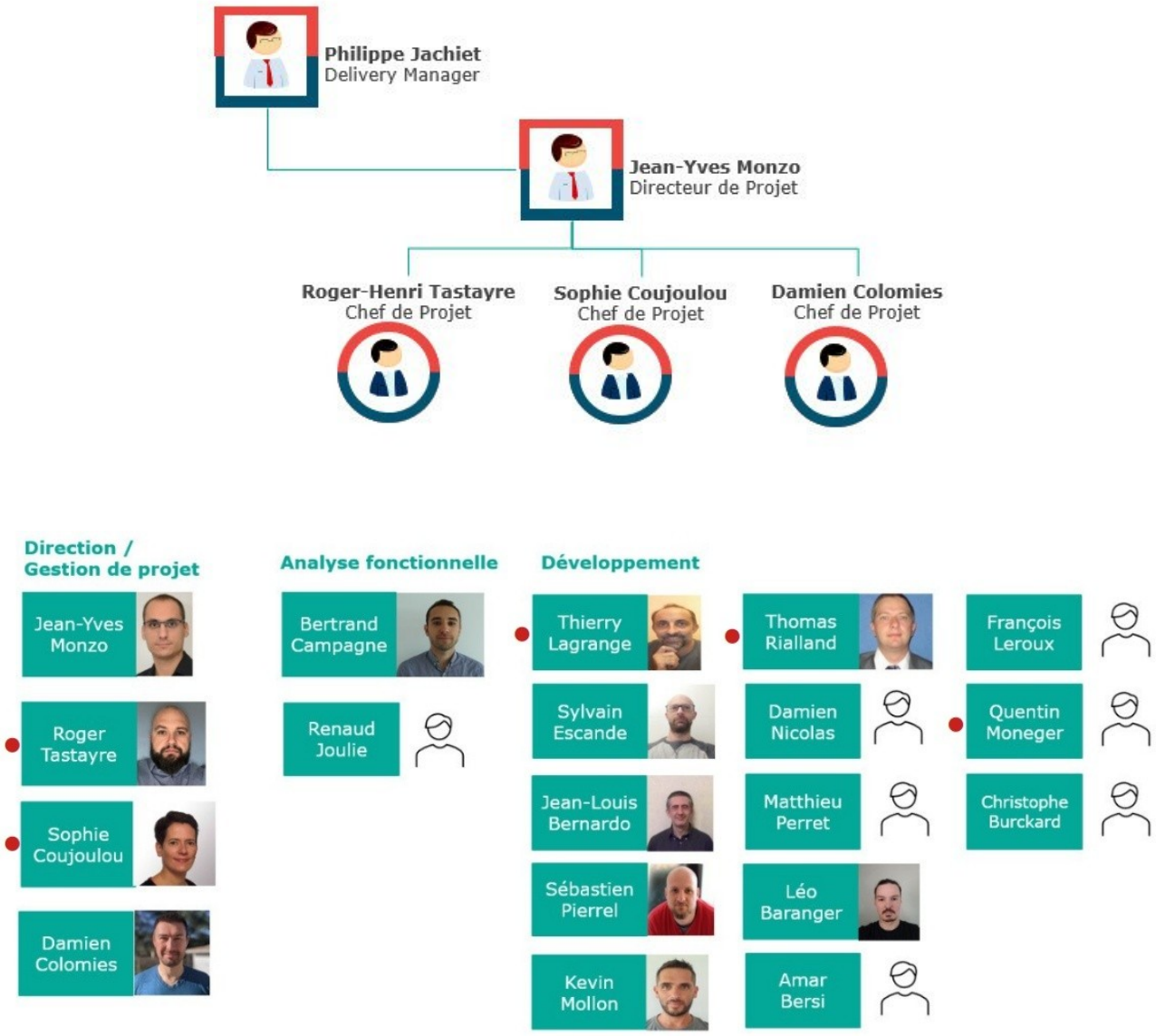
La maîtrise d'œuvre informatique du ministère est principalement de la responsabilité de la **Sous-Direction des Systèmes d'Information**, laquelle dépend de l'administration centrale et est localisée à Paris et Auzeville-Tolosane, en banlieue toulousaine.

3.2. Inetum Toulouse et le Ministère de l'Agriculture

Le Ministère de l'Agriculture est un client majeur d'Inetum Toulouse, qui dispose d'une **équipe dédiée de près de 20 personnes**, incluant un directeur de projet, des chefs de projet et un responsable technique, travaillant exclusivement dans les bureaux d'Inetum (éventuellement en télétravail partiel) et non chez le client.

L'équipe travaille sur plusieurs projets et applications du ministère, répartis entre les développeurs. La collaboration est **au forfait**, ce qui implique des phases d'expression de besoins, de chiffrages, de validation et de livraison, et des échanges fréquents avec le client généralement par l'intermédiaire des chefs de projet.

Organigramme de l'équipe



Mon stage était dirigé par **Roger-Henry Tastaire**, chef de projet. Mes autres interlocuteurs principaux étaient **Thierry Lagrange**, responsable technique, **Thomas Rialland**, développeur référent pour le projet RCPS, **Sophie Coujoulou**, cheffe de projet et responsable fonctionnelle pour le projet RCPS et **Quentin Moneger**, développeur stagiaire.

3.3. Exemples de projets gérés par l'équipe

L'équipe gère un grand nombre d'applications, parmi lesquelles :

- **AGRESTE :** outil de publication des statistiques du Ministère à destination du grand public
- **ODISSEE :** outil de dialogue et de suivi des effectifs et des emplois du Ministère
- **CAPIBARA :** générateur d'enquêtes destinées aux agriculteurs, viticulteurs et autres à des fins de statistiques à l'échelle européenne
- **BACUS :** authentification et inscription à la BDNU (Base de Données Nationale des Usagers) du Ministère
- **RCPS :** centralisation des postes et structures existant au Ministère, et lien avec RenoiRH
- **NOMADE :** application décisionnelle dédiée au contrôle de gestion
- …

3.4. Le projet RCPS

Le projet sur lequel j'ai été placé est le projet **RCPS**, pour **Référentiel Commun des Postes et Structures**.

C'est un projet regroupant plusieurs applications permettant de :

- centraliser les postes et structures existant au Ministère de l'Agriculture dans une base de données interne au ministère,
- créer, consulter, modifier et supprimer des postes et structures via une IHM,
- faire le lien, grâce à des agents, avec **RenoiRH**, système d'information des ressources humaines (SIRH) interministériel dont nous reparlerons plus loin.

Ce projet a été initié par Gfi Paris, avant d'avoir été repris par Inetum Toulouse.

Un développeur de l'équipe est dédié à ce projet et un second stagiaire a été placé dessus. Nous avons donc travaillé à 3 développeurs, sous la coordination d'un chef de projet et avec l'aide du responsable technique.

Un projet comprenant 3 domaines…

Le projet RCPS est composé de 3 domaines :

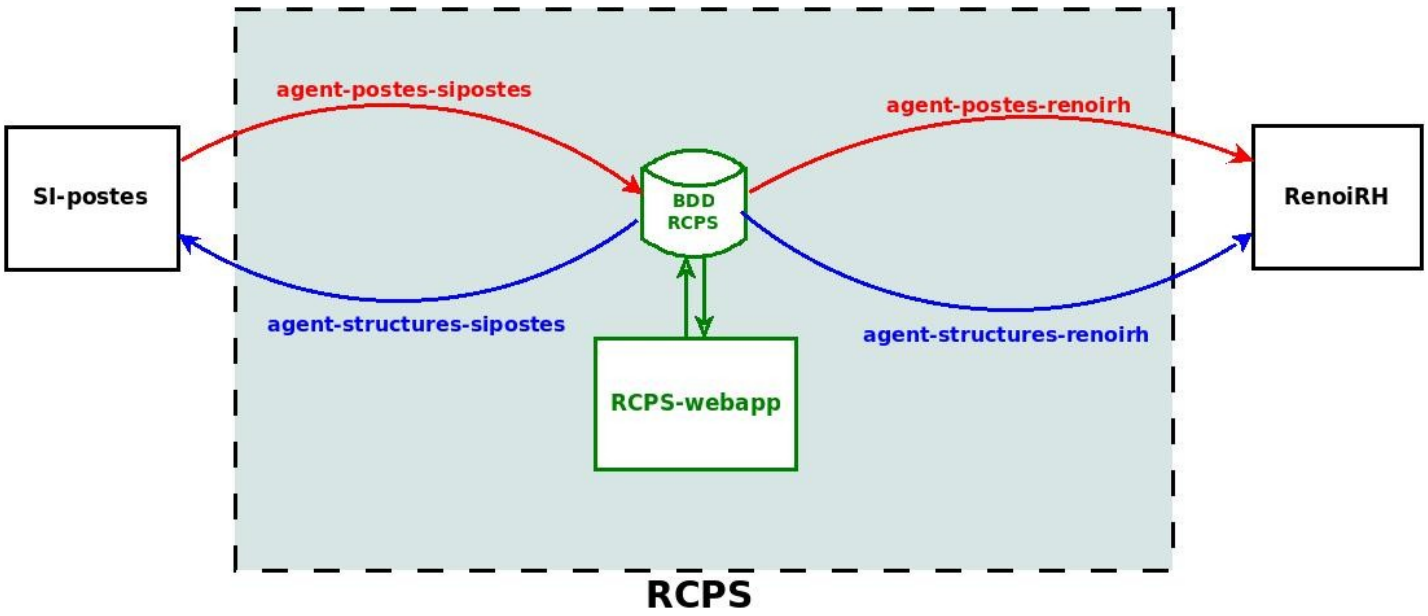
1. **la gestion des structures :** une IHM permet aux utilisateurs, en fonction de leurs habilitations, de gérer les structures du ministère, en lien avec une base de données interne,
2. **la gestion des postes :** récupération quotidienne des informations provenant des SI-postes du ministère, afin de les intégrer à la base de données interne, et de pouvoir les restituer dans l'application,
3. **les flux :** permettent la communication entre les bases de données internes au ministère et RenoiRH, le SIRH interministériel, ainsi que les SI-postes (des SI du ministère externes à RCPS).

...et composé de 7 applications

Concrètement, 7 applications Java assurent le fonctionnement du projet :

1. **RCPS-administration :** pour la gestion des habilitations des utilisateurs
2. **RCPS-webapp :** pour la gestion via IHM des postes et structures (création, modification, ...)
3. **agent-postes-sipostes :** pour les transferts des données de postes issues des SI-postes vers RCPS
4. **agent-postes-renoirh :** pour les transferts de données de RCPS vers RenoiRH
5. **agent-structures-renoirh :** pour les transferts de données de RCPS vers RenoiRH
6. **agent-structures-sipostes :** pour les transferts de données de RCPS vers les SI-postes
7. **agent-postesV2 :** refonte des agents flux postes (items 3 et 4 ci-dessus), afin d'unifier en une seule application le transfert des données de postes des SI-postes à RenoiRH via RCPS

Schématiquement :



Ma mission

Dans le cadre de ma mission de stage, j'ai travaillé sur :

- **RCPS-webapp**, pour la partie structures uniquement
- **agent-structures-renoirh**, pour les flux avec RenoiRH

Il s'agit donc d'une **webapp** et d'un **agent**, à savoir d'une application avec une partie front et une partie back, et d'une application avec uniquement du back. Dans les 2 cas, elles communiquent avec une base de données Postgresql, et sont entièrement codées en Java/JEE.

4. Prise en main du projet

4.1. Environnement technique

Le projet RCPS s'inscrit dans les cadres définis pour le Système d'Information du Ministère et doit suivre un certain nombre de contraintes techniques, afin notamment d'être conforme aux normes et protocoles de sécurité du Ministère et également d'être intégrable aux solutions d'hébergement qu'il propose. En particulier, cela implique l'utilisation du framework interne du Ministère, **Orion**, basé sur Java/JEE et que nous présentons un peu plus loin.

RCPS est destiné à un usage interne, l'ergonomie et le design ont donc leur importance mais à un degré moindre que pour une application grand public.

4.1.1. Base de données

Les bases de données sont en **PostgreSQL 9.3**.

> ***Postgresql*** *est un système de gestion de base de données relationnelle open source orienté objet. C'est un projet libre géré par une communauté de développeurs, multi-plateforme et réputé pour son comportement stable et son respect des normes ANSI SQL. Il est ainsi très largement utilisé en entreprises.*

Pour le développement de l'application, nous utilisons une base de données locale via un fichier *dump* qui permet d'initialiser la base avec l'ensemble des schémas, tables, contraintes ainsi qu'un jeu de données de test.

La base de données est visualisée et gérée via **pgAdmin** ou **Dbeaver**. J'ai utilisé les deux en fonction des besoins.

4.1.2. Hébergement et versioning des sources

Le Ministère utilise **SVN** pour l'hébergement et le versioning des sources. Inetum importe les sources sur un serveur GitLab dédié et au sein de l'équipe nous travaillons donc avec l'outil **Git** et l'hébergeur **GitLab**.

> ***Git*** *est un logiciel de gestion de versions, adapté à toutes tailles de projets. Contrairement à SVN, Git fonctionne de façon décentralisée : le développement ne se fait pas sur un serveur centralisé, chaque contributeur peut développer sur son propre dépôt local et travailler hors connexion. Git facilite ensuite la fusion (merge) des différents dépôts.*
>
> ***GitLab*** *est une plate-forme collaborative open source d'hébergement de code et de développement, basée sur les fonctionnalités de Git. Elle permet de gérer des dépôts de code source et leurs différentes versions. En plus des fonctionnalités propres à Git, GitLab intègre également l'ensemble des étapes du DevOps.*

4.1.3. Framework Orion et dépendances

La contrainte principale du projet est l'utilisation du **framework Orion**. Ayant été développé en interne par le service informatique du Ministère, les seules documentations disponibles sont celles fournies par le Ministère, qui bien que denses sont lacunaires. À noter que la dernière mise à jour du framework date de début 2017.

De l'utilisation d'Orion découlent un certain nombre de contraintes en termes de versions de logiciels et de configurations :

- une version spécifique de **Java 6** pour la webapp et de **Java 8** pour l'agent
- **Maven 3.2.5** avec des fichiers de configuration ciblant les dépôts du Ministère

> ***Maven*** *est un outil de build (construction de projets) open source qui permet de faciliter et d'automatiser certaines tâches de la gestion d'un projet Java, telles que la gestion des dépendances, la compilation, la gestion des tests unitaires, le déploiement des applications ou encore la génération de documentation.*

- un serveur **Glassfish 3.1** modifié

> ***Glassfish*** *est un serveur d'applications open source compatible JEE. C'est l'implémentation de référence, développée par Oracle.*

- les certificats nécessaires à l'authentification du serveur Glassfish et du JDK auprès des portails du Ministère

4.1.4. Environnement de Développement Intégré (IDE)

Les IDE Java théoriquement les plus adaptés à l'utilisation d'Orion sont Intellij et NetBeans, car ils intègrent nativement la gestion d'AspectJ, indispensable au fonctionnement d'Orion, mais l'âge de NetBeans et les nombreux désagréments que cela implique le mettent hors jeu.

La majorité de l'équipe travaille ainsi sur **Intellij Ultimate**.

En tant que stagiaire et faute de licence Intellij, j'ai pour ma part utilisé **Eclipse**, avec un plug-in AspectJ. Faire fonctionner ensemble tous ces logiciels (Java, Maven, Glassfish, AspectJ) dans des versions plutôt anciennes n'a pas été évident mais s'est avéré réalisable.

> *Un* ***IDE*** *(Integrated Development Environment) est un logiciel de création d'applications qui rassemble en une seule interface différents outils de développement, tels qu'un éditeur de code, des utilitaires d'automatisation (compilation, tests, packaging, ...), un debugger, etc.*
>
> ***Eclipse*** *est un IDE extensible extrêmement complet, adapté, historiquement, au langage Java. Multi-plateforme, gratuit et open source, il est l'un des plus utilisés avec Intellij.*

4.2. Processus et méthodes de travail

Inetum travaille **au forfait** sur le projet RCPS avec le Ministère de l'Agriculture (MA).

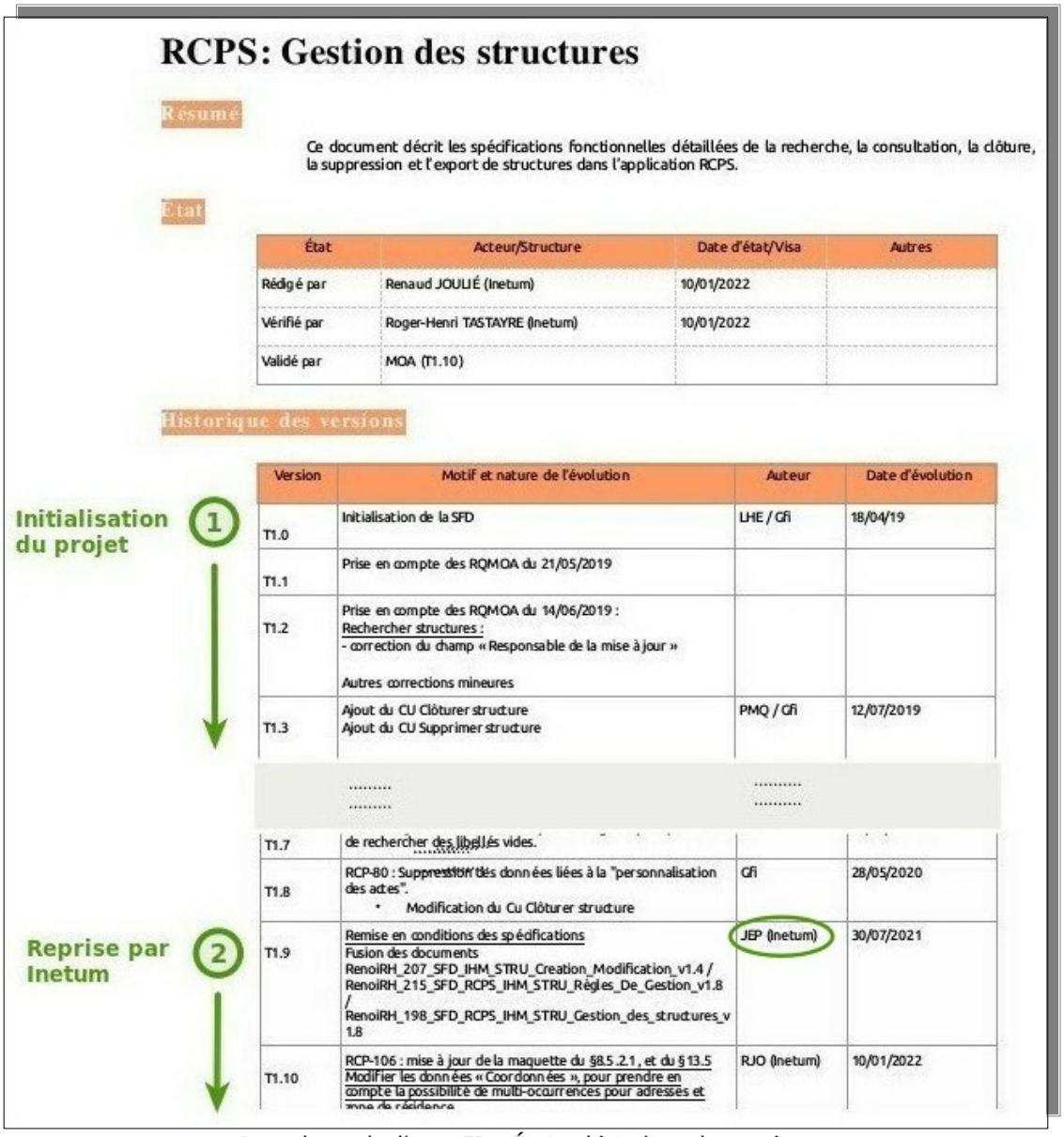
4.2.1. Les spécifications fonctionnelles détaillées

Des **spécifications fonctionnelles détaillées (SFD)** décrivent les fonctionnements attendus des applications. Ces spécifications sont rédigées par l'équipe fonctionnelle d'Inetum (ou du précédent prestataire pour les versions antérieures, en l'occurrence Gfi Paris), vérifiées par le chef de projet, puis validées par le **maître d'ouvrage**, le Ministère.

Le projet ayant été initié par un autre prestataire, Inetum a donc hérité du projet dans un certain état d'avancement, avec des SFD déjà établies. Les SFD ont été initialisées en 2019 et ont connu de nombreuses versions avant d'être reprises en 2021 par Inetum Toulouse.

Les SFD contiennent notamment :

- les maquettes / cinématique des écrans s'il y a une IHM
- les processus
- les cas d'utilisation détaillés
- la liste des règles de gestion
- la liste des messages (d'erreur, d'information, ...)



RCPS: Gestion des structures

Résumé

Ce document décrit les spécifications fonctionnelles détaillées de la recherche, la consultation, la clôture, la suppression et l'export de structures dans l'application RCPS.

État

État	Acteur/Structure	Date d'état/Visa	Autres
Rédigé par	Renaud JOULIÉ (Inetum)	10/01/2022	
Vérifié par	Roger-Henri TASTAYRE (Inetum)	10/01/2022	
Validé par	MOA (T1.10)		

Historique des versions

Initialisation du projet 1

Reprise par Inetum 2

Version	Motif et nature de l'évolution	Auteur	Date d'évolution
T1.0	Initialisation de la SFD	LHE / GFi	18/04/19
T1.1	Prise en compte des RQMOA du 21/05/2019		
T1.2	Prise en compte des RQMOA du 14/06/2019 : Rechercher structures : - correction du champ « Responsable de la mise à jour » Autres corrections mineures		
T1.3	Ajout du CU Clôturer structure Ajout du CU Supprimer structure	PMQ / GFi	12/07/2019
.....	.....	.....	.....
T1.7	de rechercher des libellés vides.		
T1.8	RCP-80 : Suppression des données liées à la "personnalisation des actes". • Modification du Cu Clôturer structure	GFi	28/05/2020
T1.9	Remise en conditions des spécifications Fusion des documents RenoiRH_207_SFD_IHM_STRU_Creation_Modification_v1.4 / RenoiRH_215_SFD_RCPS_IHM_STRU_Règles_De_Gestion_v1.8 / RenoiRH_198_SFD_RCPS_IHM_STRU_Gestion_des_structures_v1.8	JEP (Inetum)	30/07/2021
T1.10	RCP-106 : mise à jour de la maquette du §8.5.2.1, et du §13.5 Modifier les données « Coordonnées », pour prendre en compte la possibilité de multi-occurrences pour adresses et	RJO (Inetum)	10/01/2022

Page de garde d'une SFD – État et historique des versions

Voir :

- Annexe 1 pour des extraits de Cas d'Utilisation
- Annexe 2 pour des extraits de Règles de Gestion

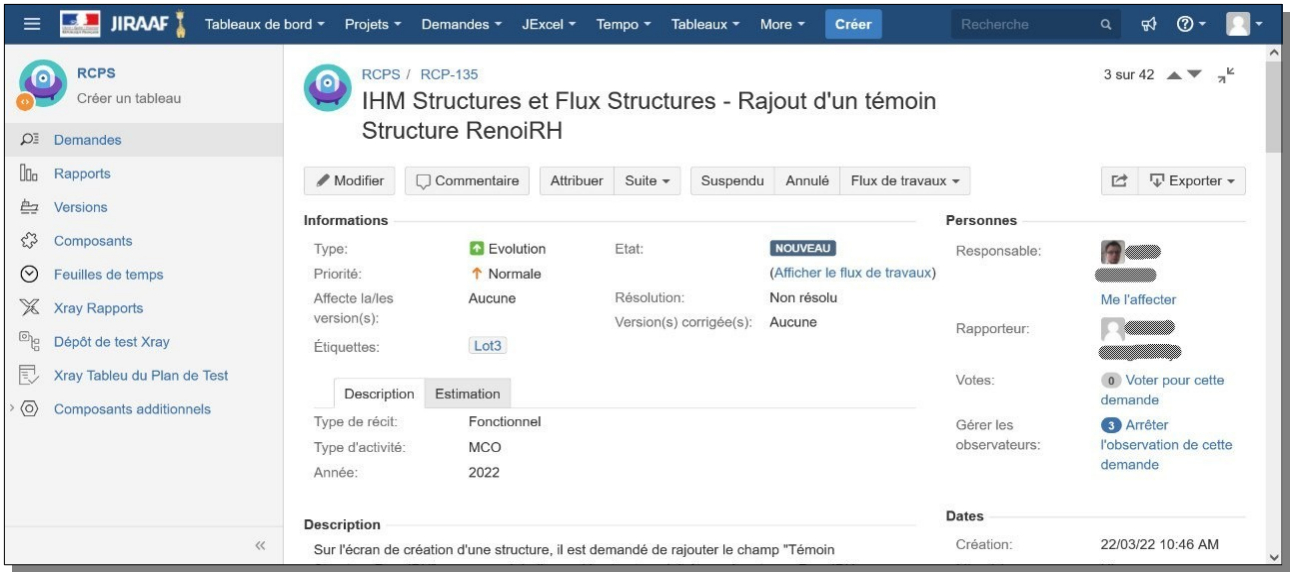
4.2.2. Demandes d'évolutions/corrections et portail JIRAAF

Le Ministère formule des **demandes d'évolutions ou de corrections** sur le code existant, regroupées par **lots**. Toutes les demandes sont visibles sur un portail dédié et sécurisé du ministère, le portail JIRAAF. Chaque demande d'évolution ou de correction est appelée usuellement « une Jira ».

> Une « *Jira* » = Une demande d'évolution ou de correction

Chaque Jira mentionne notamment son **type** (évolution ou correction), le **lot** à laquelle elle appartient, une **description fonctionnelle et/ou technique** et les interlocuteurs concernés, et dispose d'une **zone d'échange de messages** entre les interlocuteurs.

- Exemple de la Jira **RCP-135** du Lot 3, qui est une évolution concernant les applications *RCPS-webapp* et *agent-structures-renoirh* :



Portail JIRAAF - Exemple d'une Jira

Le lot que j'ai été amené à traiter, en équipe avec un second stagiaire et le développeur référent pour le projet RCPS, est le lot 3 ci-dessous :

Lot3

- Fiche RCP-108 : Flux structure - gestion des compte-rendu d'intégration
- Fiche RCP-129 : IHM Structures - Onglet « Récapitulatif et validation » : Absence données et blocage validation
- Fiche RCP-130 : IHM Structures - Onglet « Coordonnées » : Pouvoir modifier une adresse
- Fiche RCP-131 : IHM Structures - Alimentation Liste déroulante « Responsable mise à jour »
- Fiche RCP-132 : IHM Structures - Alimentation Liste déroulante « MAPS »
- Fiche RCP-133 : Flux structure - ZE0F Adresses
- Fiche RCP-134 : Flux structure - ZEXA Zone de résidence
- Fiche RCP-135 : IHM Structures et Flux Structures - Rajout d'un témoin Structure RenoiRH
- Fiche RCP-136 : IHM Structures et Flux Structures - Suppression du témoin générique
- Fiche RCP-137 : IHM Structures et Flux Structures - Liens organisationnels (structure mère et structure administrative)
- Fiche RCP-138 : IHM Structures - Type de territoire

Je détaillerai plus loin le développement de certaines de ces Jira. Certaines d'entre elles ne concernent que la webapp (ex : RCP-138), d'autres seulement l'agent (ex : RCP-108) et d'autres enfin les deux applications (ex : RCP-135).

4.2.3. Échanges avec le client

À partir de la demande initiale du client, des **échanges** ont lieu pour **affiner l'expression des besoins**, qui nécessitent souvent d'être clarifiés, le client ayant une vue plutôt fonctionnelle des besoins, alors qu'Inetum

doit être en mesure de concevoir la solution technique adaptée. La demande initiale doit donc généralement être clarifiée, voire ajustée en fonction des contraintes techniques.

Les échanges peuvent se faire de 2 manières :

- par l'intermédiaire du chef de projet, qui peut solliciter des réunions et ateliers avec le client si nécessaire,
- directement entre les développeurs et les interlocuteurs du Ministère via la zone de commentaires des Jira.

4.2.4. Chiffrage par unité d'œuvre

Une fois les besoins clarifiés et précis, les développeurs doivent effectuer le **chiffrage** de chaque Jira. Il s'agit de faire une estimation du temps nécessaire au traitement de la Jira. Pour ce faire, le client fournit des grilles permettant d'estimer la charge de travail à effectuer par **unité d'œuvre (UO)**, en calculant leurs **granularités** respectives.

7 unités d'œuvre sont à prendre en compte pour un chiffrage d'évolution/correction :

ID de l'unité d'œuvre	Définition
IOND	Spécifications dans le cas d'un nouveau développement => édition, cas d'utilisation, traitement, maquettes *Exemple : ajout d'un bouton qui génère un PDF*
IOME	Spécifications dans le cas d'une maintenant => utilisation pour uniquement l'ajout ou la modification de tables et/ou relations sans traitement
POPAG	Création / Modification d'une page de navigation
POFRM	Création / Modification d'un onglet de formulaire
POLIS	Création / Modification d'une liste
POSPE	Création / Modification d'un développement spécifique
POEDT	Création / Modification d'édition

Chaque unité d'œuvre prend la forme d'un tableau Excel détaillant les éléments à prendre en compte pour cette unité, leurs complexités, les bases de calcul permettant de calculer la note de complexité de chaque élément, afin d'aboutir à un **score de granularité** global pour l'unité d'œuvre en question.

Ce score permet de définir un **niveau de granularité** pour l'unité d'œuvre en question, qui est un qualificatif comme *Très faible*, *Élevé*, *Très complexe*,...

Ainsi, l'analyse d'une Jira donnée pourra par exemple aboutir au chiffrage en unités d'œuvre suivant :

- **POSPE** très élevé (granularité 243)
- **POFRM** moyen (granularité 70)
- **POLIS** faible (granularité 35)
- **POPAG** moyen (granularité 2,2)
- **IOME** très faible (granularité 0,25)

Ce chiffrage, qui doit bien sûr être justifiable, vient étayer auprès du client une **estimation en nombre de jours** nécessaires au traitement de la Jira. C'est une étape fastidieuse mais essentielle de la phase préliminaire au développement. Fastidieuse car les tableaux de calcul de granularité sont délicats à remplir. Voici l'exemple du tableau servant à estimer l'unité œuvre IOND :

Calcul Granularité UO IOND

Spécifications dans le cas d'un nouveau développement

Élément particulier	Niveau de complexité	Base du calcul	Nombre	Note de complexité
Cas d'utilisation standard	simple (nombre d'informations mises en jeu inférieur ou égal à 10)	cas	3	6
	normal (nombre d'informations mises en jeu entre 10 et 25 inclus)	cas	2	8
	complexe (nombre d'informations mises en jeu entre 25 et 50)	cas		0
Réalisation d'une maquette	simple (nombre de pages mises en jeu inférieur ou égal à 10)	Global		0
	normal (nombre de pages mises en jeu entre 10 et 25 inclus)	Global	1	4
	complexe (nombre de pages mises en jeu entre 25 et 50)	Global		0
Calculs complexes dans un cas d'utilisation standard	simple (nombre d'informations mises en jeu	Calcul		0
	normal (nombre d'informations mises en jeu entre 10 et 25 inclus)	Calcul	3	12
	complexe (nombre d'informations mises en	Calcul		0
Éditions (incluant la définition des informations à prendre en compte et la maquette de l'édition)	simple (nombre d'informations mises en jeu inférieur ou égal à 10)	Edition	2	4
	normal (nombre d'informations mises en jeu entre 10 et 25 inclus)	Edition		0
	complexe (nombre d'informations mises en jeu entre 25 et 50)	Edition	1	6
			Granularité	**40**
			Niveau de granularité	**MOYEN**

➢ Pour un second exemple (unité d'œuvre POLIS – création/modification d'une liste), se référer à l'annexe 3.

4.2.5. Validation par le client et livraisons

L'acceptation par le client des différents chiffrages et proposition formulés par le chef de projet mène à la **validation du lot**.

La validation du lot ouvre les phases de :

- **mise à jour des SFD** (spécifications fonctionnelles détaillées) des applications impliquées, SFD pour lesquelles une **première date de livraison** est fixée,
- **développement du code**, pour lequel une **seconde date de livraison** est fixée.

4.2.6. Synthèse du processus

Les étapes de la collaboration au forfait sont donc les suivantes :

DEMANDE CLIENT	ÉCHANGES	CHIFFRAGE	VALIDATION	DEV	LIVRAISON
Formulation par le client d'un lot de demandes d'évolutions ou corrections	*Discussions avec le client pour affiner les besoins*	*Estimation du temps nécessaire à chaque tâche*	*Validation du chiffrage par le client*	*Mises à jour des SFD et codage*	*Livraison des nouvelles SFD puis du code*

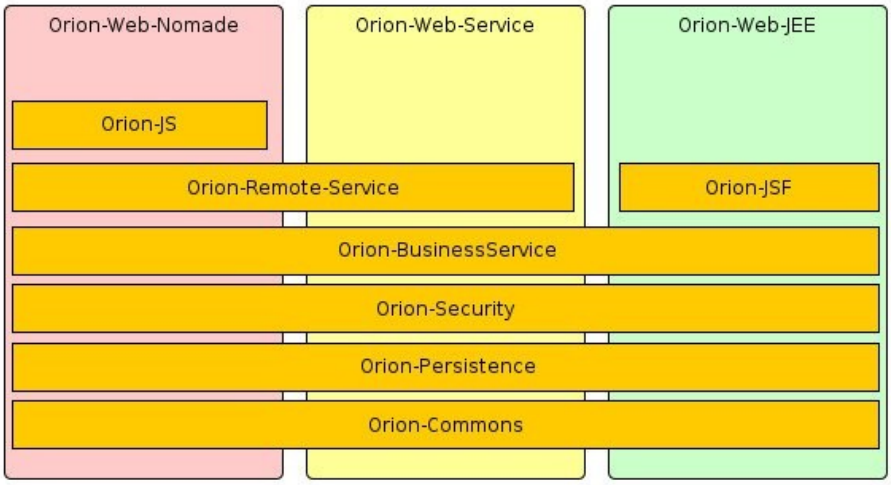
4.3. Introduction à Orion

Orion est l'élément central des outils de développement informatique du Ministère de l'Agriculture. C'est un ensemble de produits utilisés pour la conception d'applications dans les domaines :

- des systèmes d'information opérationnels
- de la prise de décisions
- des systèmes d'échanges
- de la géographie

Orion a pour buts d'industrialiser les développements informatiques, de fournir un outil de développement en client léger et de normaliser les développements. La technologie **Java/JEE** est la technologie standard des développements applicatifs du Ministère (la technologie Js/REST étant utilisée marginalement).

Orion met à disposition plusieurs distributions (assemblages cohérents de composants) destinées à des domaines et besoins précis :



La distribution utilisée pour les **applications Web** du Ministère est **Orion-Web-JEE**, qui en plus des composants communs (Orion-Commons, Orion-Persistence, …) utilise **Orion-JSF**, une surcouche de **JSF**.

JSF (Java Server Faces) *est un framework de développement d'applications Web intégré à JEE, une alternative donc notamment aux JSP/Servlets, à Spring MVC, etc. C'est une spécification, qui a donc besoin d'une implémentation.*

Orion-JSF *fournit une librairie de composants JSF2 pour la construction des pages d'une webapp Orion. Une page web Orion est décrite par un* ***fichier xhtml****, qui est un ensemble de balises html classiques et de balises puisées dans les* ***taglibs JSF*** *et les* ***taglibs d'Orion****.*

structure-coreCommun.xhtml

```
<ui:composition xmlns:ui="http://java.sun.com/jsf/facelets"
                xmlns:o="http://orion.agriculture-gouv.fr/standard"
                xmlns:co="http://orion.agriculture-gouv.fr/core"
                xmlns="http://www.w3.org/1999/xhtml"
        >

    <o:tabPanels selected="identification">
        <o:tabs>
            <o:tab>…
            </o:tab>
            <o:tab rendered="#{mode != 'creation' ? true : false}">
                <o:clickableTab target="coordonnees">
                    <co:ajax execute="@form" render="#{mode == 'modif' ?
                     'mainform:idSaveButtonModifierStructureHeader-act-wrp-cmd
                     mainform:idValidateButtonModifierStructureHeader-act-wrp-cmd
                     mainform:idSaveButtonModifierStructure-act-wrp-cmd
                     mainform:idValidateButtonModifierStructure-act-wrp-cmd': ''}"/>
                     #{Messages['structures.panel.coordonnees.title']}
                </o:clickableTab>
            </o:tab>
```

Import des taglibs

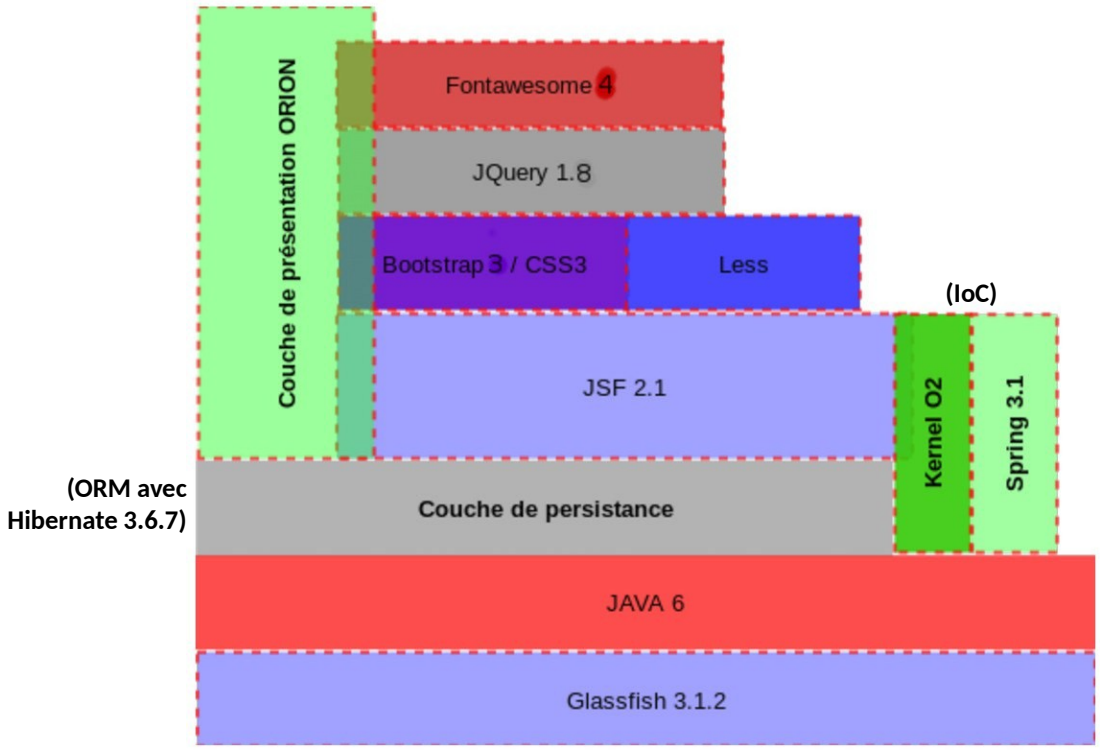
Page Web Orion : import des taglibs et utilisation des balises importées

*Orion-JSF facilite la **factorisation de code** et l'utilisation de **templates** avec l'attribut* `template` *et les balises* `define` *et* `include`.

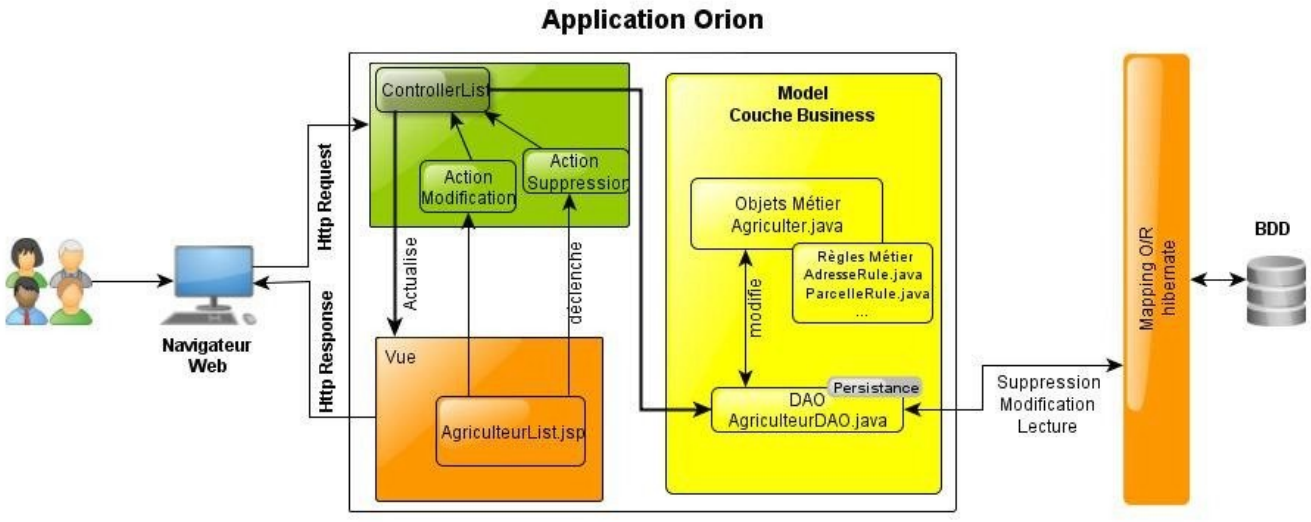
```
<ui:composition xmlns:ui="http://java.sun.com/jsf/facelets"
                xmlns:o="http://orion.agriculture-gouv.fr/standard"
                xmlns="http://www.w3.org/1999/xhtml"
                template="/template/splitpanel-template.xhtml">
    <ui:param name="pageLabel" value="#{Messages['accueil.structures.page.title']}"/>
    <ui:define name="title">#{Messages['accueil.structures.page.title']}</ui:define>
    <ui:define name="leftContent">
        <ui:include src="/structures/menu/structuresMenu.xhtml"/>
    </ui:define>
```

Page Web Orion : exemple d'une page basée sur un template, redéfinissant des composants et insérant une autre page

Socle technique



Une application Orion



Une application Orion suit l'architecture **MVCA** (Modèle-Vues-Contrôleurs-**Actions**). Les **actions** définissent des traitements sur le modèle.

Créer et configurer un projet Orion

Un projet Orion est initialisé via un **archétype Maven** : `orion-web-archetype`. Plusieurs dépôts Maven sont fournis par le Ministère, configurés dans le fichier settings.xml du dépôt local, et listés dans le pom.xml.

Pour la configuration du projet, les fichiers **env.properties** par profil sont utilisés. Ils permettent notamment de paramétrer la `PersistenceServiceFactory` (connexion à la base de données métier de l'application) et la `SecurityPersistenceServiceFactory` (connexion à la base de données d'habilitations de l'application).

Fonctionnement du Kernel Orion

Le **Kernel Orion** est basé sur Spring. Tous les beans (actions, controllers, services, DAO, ...) sont déclarés dans un fichier **kernel-config.xml**. Ces déclarations permettent notamment de faire le lien entre les alias (d'actions, de controllers, de services, ...) utilisés dans les vues (fichiers xhtml) et les classes Java correspondantes, de déclarer le *scope* des beans, d'affecter des valeurs à des propriétés, etc.

```
<bean name="creationModificationStructureFormCtrl"
      classname="fr.gouv.agriculture.dsis.renoirh.controller.structures.CreationModificationStructureFormController"
      parent="fr.gouv.agriculture.orion.controller.form.AbstractBusinessServiceFormController"
      scope="session">
    <property name="businessClass"
              value="fr.gouv.agriculture.dsis.renoirh.dto.IdentificationDTO"/>
    <property name="businessService"
              value-ref="fr.gouv.agriculture.dsis.renoirh.service.structures.GestionStructuresService"/>
</bean>
```

kernel-config.xml : déclaration d'un contrôleur de formulaire, qui étend une classe abstraite du framework ; déclaration de la classe métier liée (`IdentificationDTO`) et du service associé (`GestionStructuresService`)

Les beans peuvent être injectés dans une classe grâce à l'annotation `@Inject` fournie par le kernel Orion.

Paramétrage d'Hibernate

Orion est basé sur **Hibernate**. Les **classes métiers persistantes (entités)** héritent de la classe `fr.gouv.agriculture.orion.business.BaseEntity` du framework. Chaque entité persistante est mappée avec la table correspondante de la base de données via un fichier dédié **.hbm.xml**, où sont également configurées les relations entre entités/tables. Il y a donc autant de fichiers .hbm.xml que d'entités persistantes.

Tous ces fichiers .hbm.xml doivent ensuite être référencés dans les fichiers de configuration d'Hibernate **hibernate.cfg.xml** (classes métier) et **hibernate-administration.cfg.xml** (classes internes pour les habilitations, préférences, ...), lesquels sont référencés dans kernel-config.xml.

5. Ma mission au sein du projet RCPS

Les applications sur lesquelles j'ai travaillé sont complexes, et obéissent à de nombreuses contraintes, notamment pour accéder aux données sécurisées du Ministère et pour rendre possible leur déploiement et développement en local : compte AGRICOLL d'accès aux dépôts Maven du Ministère, *fake login* pour accéder aux données de la base, **Docker** local pour accéder à un OFS (Orion File System) local et simuler le serveur de fichiers du Ministère, etc.

Ce chapitre a pour objet de présenter brièvement mais, je l'espère, suffisamment clairement, **ma mission concrète**, le **fonctionnement des applications** et leurs objectifs, et quelques **exemples et extraits concrets** des actions de développement menées.

5.1. Un lot de demandes d'évolutions chiffrées et validées

Un lot de demandes d'évolutions a été formulé par le client, concernant les applications *RCPS-webapp* et *agent-structures-renoirh* :

Lot3

- Fiche RCP-108 : Flux structure - gestion des compte-rendu d'intégration
- Fiche RCP-129 : IHM Structures - Onglet « Récapitulatif et validation » : Absence données et blocage validation
- Fiche RCP-130 : IHM Structures - Onglet « Coordonnées » : Pouvoir modifier une adresse
- Fiche RCP-131 : IHM Structures - Alimentation Liste déroulante « Responsable mise à jour »
- Fiche RCP-132 : IHM Structures - Alimentation Liste déroulante « MAPS »
- Fiche RCP-133 : Flux structure - ZE0F Adresses
- Fiche RCP-134 : Flux structure - ZEXA Zone de résidence
- Fiche RCP-135 : IHM Structures et Flux Structures - Rajout d'un témoin Structure RenoiRH
- Fiche RCP-136 : IHM Structures et Flux Structures - Suppression du témoin générique
- Fiche RCP-137 : IHM Structures et Flux Structures - Liens organisationnels (structure mère et structure administrative)
- Fiche RCP-138 : IHM Structures - Type de territoire

Les détails de chaque demande figurent sur le portail JIRAAF.

Après avoir analysé le code existant et le fonctionnement des applications, ma première mission a été de **chiffrer** ces demandes (voir 4.2.4). Certains besoins n'étant pas totalement clairs, ou certaines demandes pouvant être solutionnées de différentes manières, il m'a fallu **échanger avec le client**, soit via l'intermédiaire de mon chef de projet soit directement, afin d'affiner les besoins et de valider certaines propositions techniques.

Un **chiffrage complet** a ainsi pu être produit et soumis par l'intermédiaire du chef de projet au client. Ce chiffrage a été **validé**, et des **dates de livraison** pour la mise à jour des SFD d'une part, et le code d'autre part, ont été fixées, pour début septembre et début octobre respectivement.

Nous nous sommes réparti les demandes avec un collègue et avons pu **lancer le développement** (déjà partiellement initié lors des phases de chiffrage).

5.2. Fonctionnement des applications RCPS-webapp et agent-structures-renoirh

Avant de prendre l'exemple d'une Jira spécifique, une brève explication sur l'objet et le fonctionnement des applications s'impose.

RCPS-webapp est une application web permettant de consulter, modifier, ajouter, supprimer des structures du Ministère de l'Agriculture via une IHM. Les données sont stockées dans la **base de données *rcps***, une base de données interne au Ministère.

agent-structures-renoirh est un agent, c'est-à-dire une application sans IHM qui tourne en permanence et lance des traitements automatiquement à des dates ou intervalles de temps donnés. Il scrute la base de données *rcps* pour y détecter les modifications et les envoyer, sous forme de bordereaux, à un système d'informations interministériel nommé **RenoiRH**.

Les 2 applications partagent donc la **même base de données**.

5.2.1. Base et modèle de données rcps

La base de données *rcps*, interne au ministère, est une base données de Postgresql de **106 tables avant** ce lot de demandes, et **109 tables après**. Certaines de ces tables concernent les postes *(RCPS = « référentiel commun des postes et structures »)* et nous les mettons donc de côté puisque nous ne traitons que des structures.

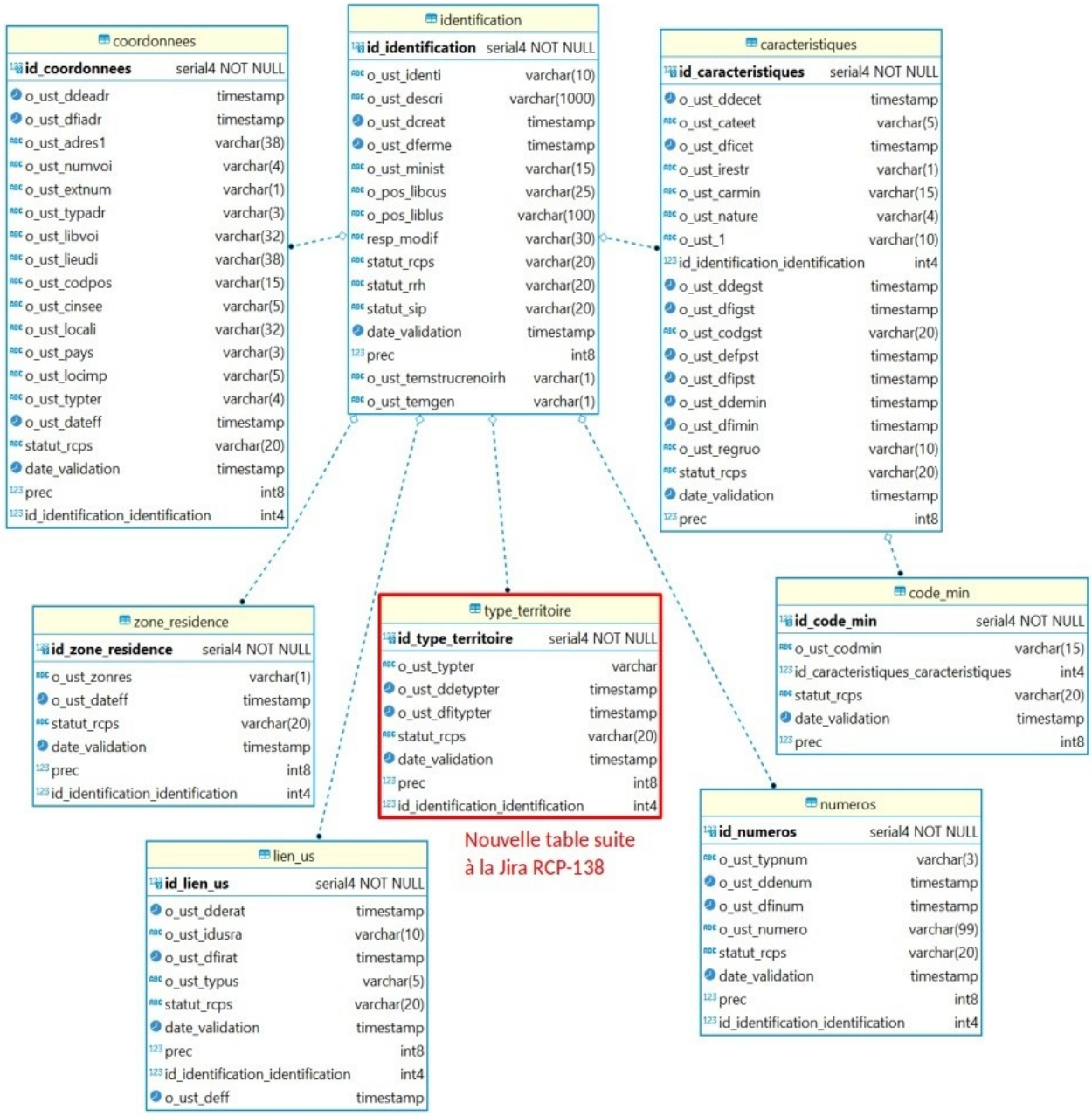
Plus d'une trentaine de tables concernent les **configurations Orion**, y compris la sécurité, les habilitations, etc. Ce sont les tables préfixées par `t_orion`. Nous ne les aborderons pas dans le corps de ce dossier car elles ne sont pas du ressort de notre équipe de développement. *(Pour quelques explications sur le modèle de données de ces tables `t_orion_*`, se référer à l'annexe 4.)*

La **partie métier** relative aux structures de la base de données est composée :

- de **tables de référence** (13 avant le lot, 15 après) contenant des données de référence relativement figées dans le temps (listes de pays, de types de territoire, de ministères, …) servant principalement à alimenter des listes déroulantes. Ces tables **ne sont en relation avec aucune autre** (une table métier qui a, par exemple, une colonne pays, stocke la donnée sous forme d'une chaîne de caractères et non d'une clé étrangère vers la table de référence).

  Voir l'annexe 5 pour le modèle de données correspondant.

- des **tables métiers** (7 avant le lot, 8 après), correspondant aux objets métiers réels :



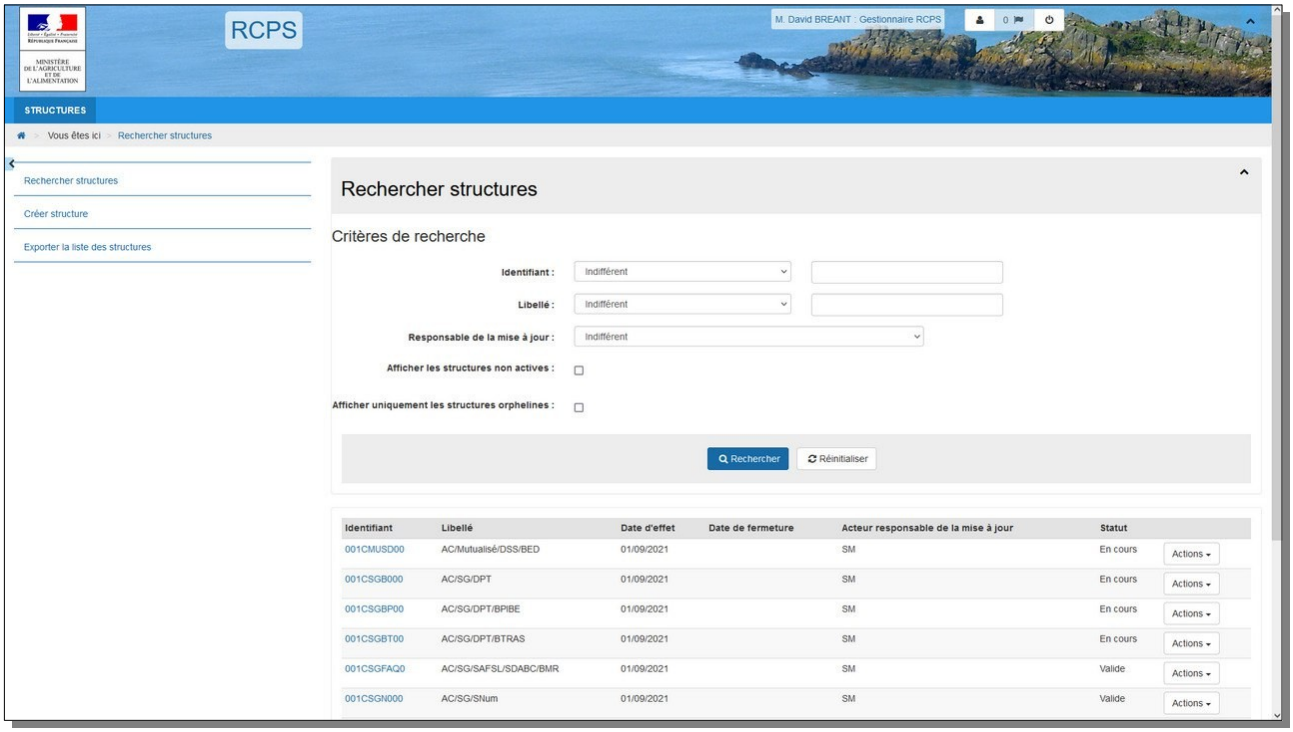
Modèle de données des tables métiers de la base de données rcps

Chaque **structure du ministère** est représentée par une ligne de la table `identification`. Chaque structure peut avoir plusieurs *coordonnées*, *zones de résidence*, *liens US*, *types de territoire*, *numéros* et *caractéristiques*. Chaque entité *caractéristiques* peut avoir plusieurs *codes ministériels*. Les relations mises en jeu sont donc toutes de type 1-n. Des **relations 1-n bidirectionnelles** sont configurées dans les fichiers .hbm.xml de toutes les entités concernées.

- des tables paramétrant la **création de bordereaux** par l'agent (3). Voir annexe 6.

5.2.2. RCPS-webapp

RCPS-webapp est une IHM permettant de lister l'ensemble des structures avec des critères de recherche, et de consulter, modifier, supprimer et créer une structure.



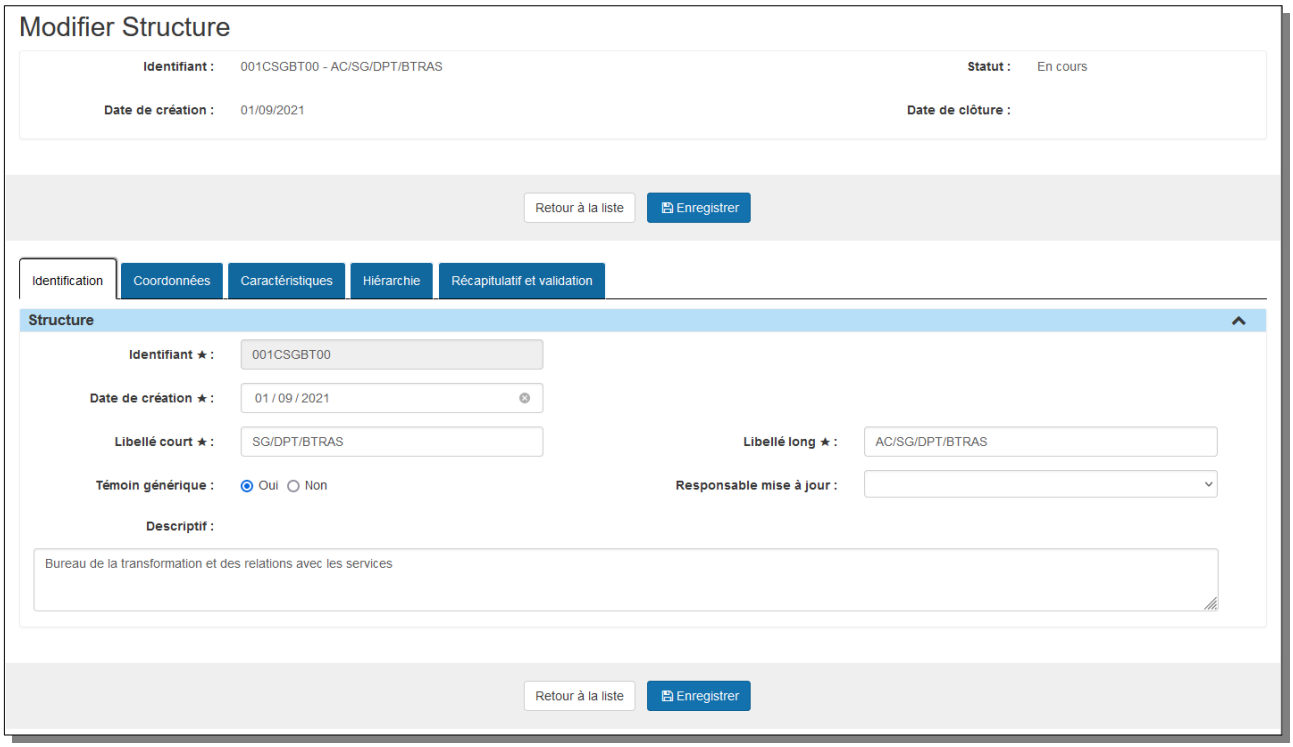
RCPS-webapp : liste des structures

La liste des structures est gérée par un **contrôleur de liste**, configuré pour utiliser l'entité persistante `IdentificationEntity`, laquelle est mappée sur la table `identification` de la base de données. La liste est ainsi peuplée automatiquement.

Chaque structure dispose d'un bouton **Actions ↓** qui présente 4 options : *Modifier-Consulter-Clôturer-Supprimer*, chaque option déclenchant une ***Action*** au sens du framework Orion (classe Java étendant la classe abstraite `BusinessAction` du framework).

Modification d'une structure

L'action *Modifier* par exemple ouvre un **formulaire** composé de 5 onglets. Ce formulaire est géré par un **contrôleur** configuré pour utiliser la classe métier `IdentificationDTO` et le **service métier** `GestionStructureService`. Ce service, entre autre choses, permet de construire et initialiser l'instance de `IdentificationDTO` correspondant à la structure cliquée, instance liée au formulaire. Nous avons donc un objet, nommons-le `identificationDTO`, lié au formulaire (*data binding*), lequel formulaire est étendu sur plusieurs onglets.



RCPS-webapp : formulaire de modification

Les sous-entités métiers d'une structure (Coordonnées, Caractéristiques, Zones de résidence, … voir modèle de données métiers plus haut) sont toutes créées par le service à l'initialisation du formulaire. Ces sous-entités étant en multi-occurrence (relations 1-n), ce sont des collections, attributs de l'objet `identificationDTO`. `identificationDTO` possède donc des collections de `CoordonneesDTO`, de `CarateristiquesDTO`, de `ZoneResidenceDTO`, etc.

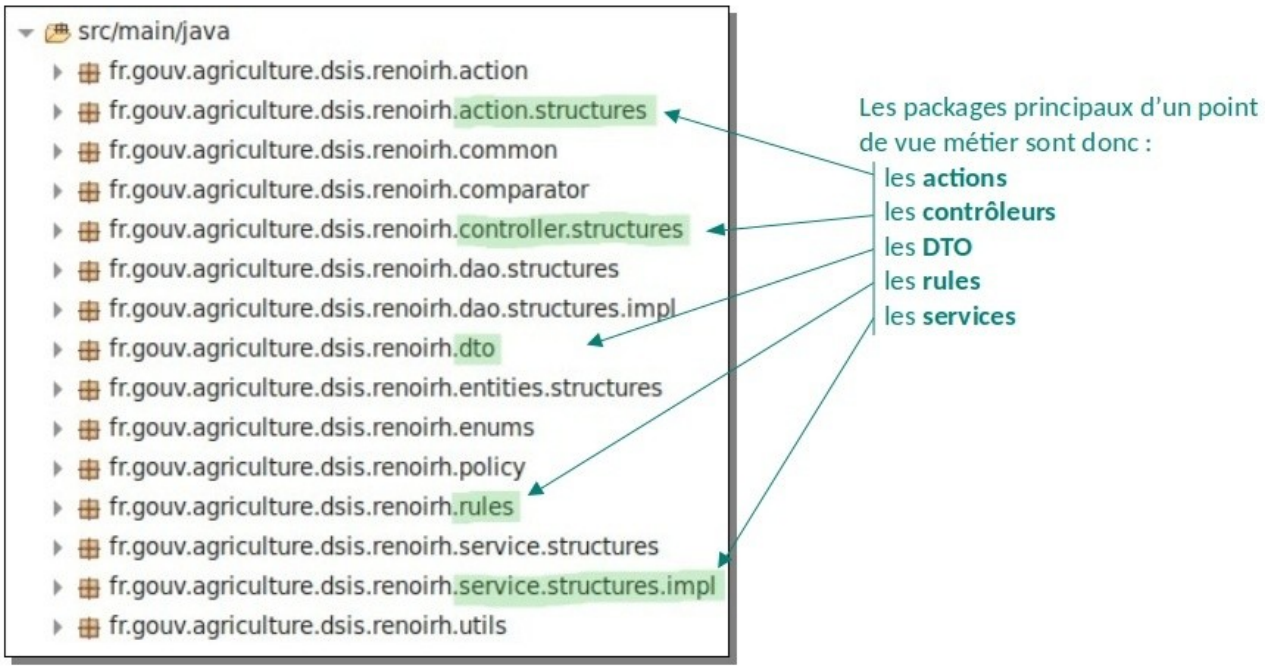
Enregistrement et validation

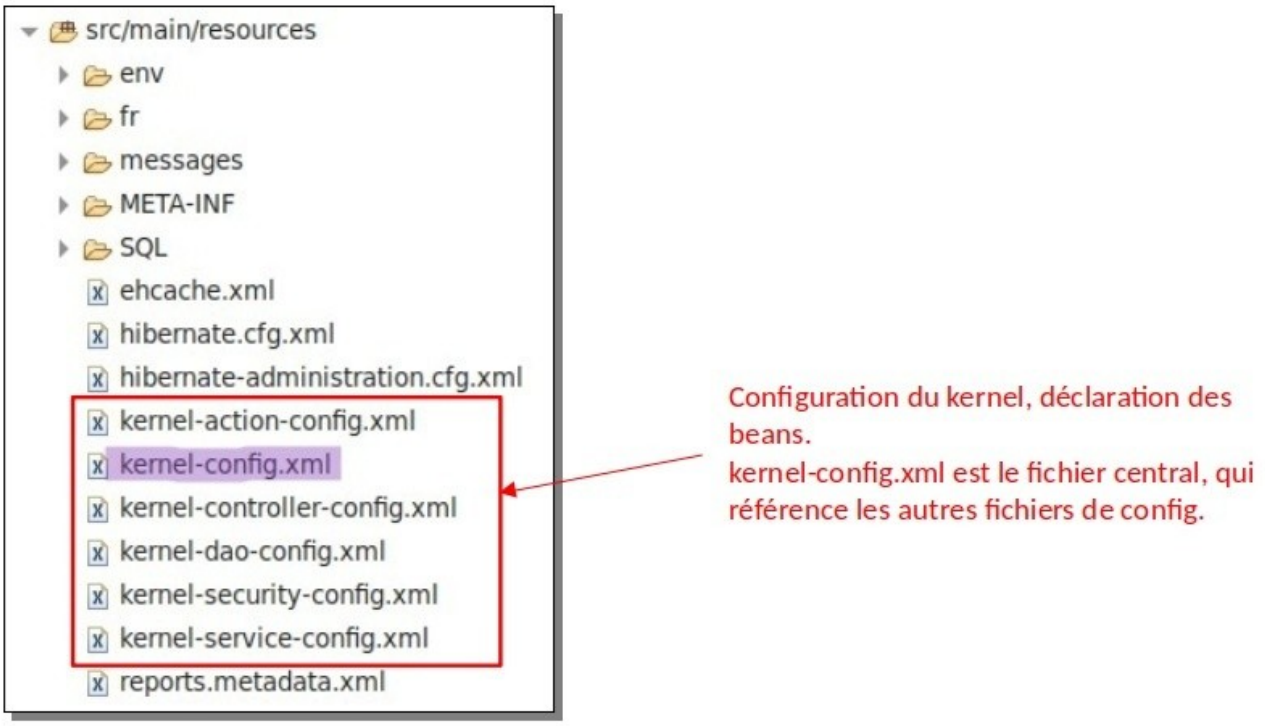
Un clic sur le bouton **Enregistrer** sauvegarde l'état de la structure en base de données, tout en maintenant la structure au statut '*En cours*' (colonne `statut_rcps` de la table `identification`). L'onglet ***Récapitulatif et validation*** permet quant à lui de visualiser l'état de la structure, et de **valider** la structure (bouton **Valider**), ce qui a pour conséquence de passer le statut de la structure à '*Validé*'. Une structure validée doit être **envoyée à RenoiRH par l'agent** (sous quelques conditions supplémentaires).

Règles de gestion

La validation est soumise à des **règles de gestion**, (voir Annexe 2 pour des exemples de règles de gestion telles qu'elles figurent dans la SFD). Ces règles sont définies dans des classes Java surchargeant la classe abstraite `AbstractBusinessRule` du framework. Chacune de ces classes, via la surcharge de la méthode `validate()`, instancie un `RuleReport` auquel elle ajoute, en fonction des validations réalisées, des `RuleMessage` de divers niveaux de criticité. En fonction du contenu du `RuleReport` final retourné par la méthode, la validation de la structure est acceptée ou rejetée. Le framework permet d'**afficher facilement les messages d'erreurs** sur l'interface.

Structure du projet Java





Les **vues** (fichiers xhtml) sont quant à elles classiquement placées dans src/main/webapp.

5.2.3. agent-structures-renoirh

L'écosystème Orion inclut un univers fonctionnel nommé **Univers Système d'Échange (ORION-SE)**, lequel fournit la distribution AGENT-JAVA pour aider à la création d'agents. Un **archétype Maven** est mis à disposition (`agent-camel-archetype`) permettant de générer un squelette d'agent répondant aux normes du Ministère et contenant notamment des classes de lancement, d'interrogation de statut et d'arrêt de l'agent.

agent-structures-renoirh est une application sans IHM qui tourne en permanence et permet à intervalles réguliers, suivant les déclarations de CRON faites dans le fichier env.properties, de lancer des traitements afin d'envoyer des informations à destination de RenoiRH. Le fonctionnement repose sur les classes `AgentStructuresRenoirhStart`, `AgentStructuresRenoirhStatus` et `AgentStructuresRenoirhStop`.

Création et envoi des bordereaux

Le rôle de cet agent est de **scruter la base de données RCPS** et de transmettre quotidiennement à RenoiRH (système d'information interministériel) les informations nécessaires afin que les bases restent synchronisées. Pour ce faire, des traitements permettent la génération de **fichiers de flux** contenant les mises à jour de données à envoyer à RenoiRH sous la forme de **bordereaux** contenus dans un fichier unique.

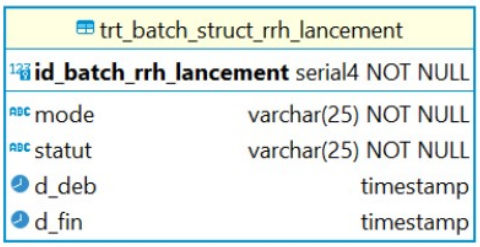
Réception d'un compte-rendu

Une fois les données de structures transmises à RenoiRH, le système attend un **fichier de compte-rendu** de la part de RenoiRH indiquant les structures qui ont été rejetées à l'intégration dans RenoiRH. Tant que le compte-rendu n'est pas arrivé, les structures concernées par le dernier envoi de données sont fermées à la modification.

Paramétrages de création des bordereaux

3 tables de la base de données entrent en jeu dans la création des bordereaux (voir Annexe 6) :

- `param_bord_struct` : pour notamment faire le lien entre les tables RCPS et les tables RenoiRH, et également spécifier quels champs RCPS sont obligatoires pour la génération de bordereaux.
- `trt_batch_struct_rrh_lancement` : pour provoquer le lancement des traitements de création de bordereaux. Une nouvelle ligne est créée automatiquement dans cette table toutes les 24h, avec `mode` = `'Automatique'` et `statut` = `'En cours'`.

  En plus de cela, on peut également **provoquer manuellement** la création de bordereaux en insérant à la main via une requête SQL une nouvelle ligne spécifiant `mode` = `'Manuel'` et `statut` = `'A traiter'`. Cette table est scrutée toutes les 5 minutes par l'agent à la recherche de telles lignes.

```
INSERT INTO rcps.trt_batch_struct_rrh_lancement (mode, statut) VALUES ('Manuel', 'A traiter');
```

- `trt_batch_struct_rrh` : pour rendre compte du processus de création de bordereaux (succès, erreurs, ...).

Lorsque l'agent lance les traitements de génération de bordereaux, il scanne la table `identification` (= la liste des structures) et recherche les structures ayant le champ `statut_rrh` est sur `'A traiter'` (une précondition à cela étant, comme nous l'avons vu, que `statut_rcps` soit sur `'Validé'`, c'est-à-dire que la structure ait été validée via l'IHM).

Pour chaque structure concernée, l'agent compare l'état de la structure au dernier état connu de RenoiRH, et compile les modifications sous forme de texte dans un bordereau selon un format précis spécifié dans la SFD et implémenté par les traitements Java et la table `param_bord_struct`.

Dépôt du fichier de bordereaux

Une fois le fichier de bordereaux finalisé, il est déposé sur un emplacement d'un serveur de fichiers du Ministère, tel que paramétré dans le fichier env.properties.

5.3. Traitement d'une Jira spécifique à RCPS-webapp : RCP-138

5.3.1. Analyse, conception et chiffrage

Jira RCP-138

En modification et en consultation d'une structure, il est demandé de rajouter un bloc **"Types de territoire"** sur l'onglet **"Coordonnées"**.

Ce nouveau bloc contient la **liste** des Types de territoire de la structure (multi-occurrence possible) avec les colonnes suivantes : "Date de début", "Date de fin" et "Type de territoire".

Dans le cas d'une modification de structure, il est possible de rajouter et de supprimer une ligne de la liste via un **formulaire** présent sous la liste composé des champs "Date de début" (format date), "Date de fin" (format date) et "Type de territoire" (liste déroulante).

L'alimentation de la liste déroulante "Type de territoire" se fait à partir d'une **table de référence** en base de données, en prenant les **instances valides à la date du jour**.

Le système doit effectuer les **contrôles** suivants :

- la date de début saisie doit être postérieure ou égale à la date de création de la structure en cours de modification

- la date de fin saisie doit être postérieure ou égale à la date de début saisie

Cette évolution implique de **multiples changements**, aussi bien en **base de données**, en **back-end,** qu'en **front-end**.

Un nouvel attribut de structure : ses types de territoires

Une structure ne possédait pas jusque là de **Types de territoire**, et par ailleurs ce nouvel attribut est en multi-occurrence. Il faut donc :

- créer une table `type_territoire` en relation 1-n avec la table `identification`
- créer l'entité persistante java `TypeTerritoireEntity` ainsi que le fichier .hbm.xml de mapping avec la base
- ajouter un attribut de type `Collection<TypeTerritoireEntity>` à `IdentificationEntity` et éditer le fichier .hbm.xml pour le *reverse mapping* (relation one-to-many bidirectionnelle)

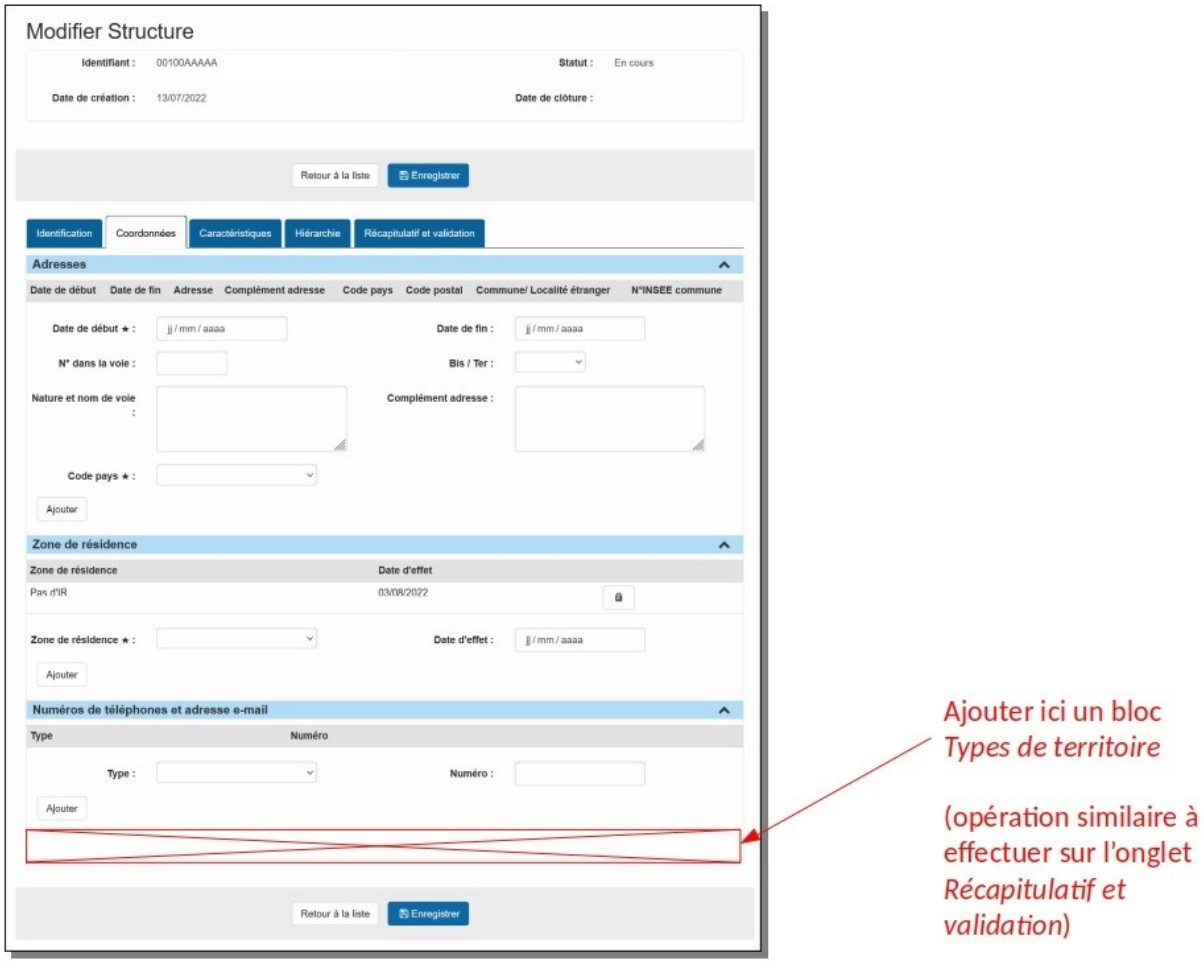
Une nouvelle table de référence

Par ailleurs la liste déroulante proposant les choix de types de territoire doit être alimentée par une table en base. Il faut donc :

- créer une table `ref_type_territoire`
- créer l'entité persistante `RefTypeTerritoireEntity` et le fichier .hbm.xml associé

IHM

Au niveau de l'IHM, un nouveau bloc doit être inséré sur les onglets *Coordonnées* et *Récapitulatif*.



Des adaptations dans toutes les couches de l'application

Il faut également créer les **DTO** et **DAO** nécessaires, les **contrôleurs**, les **actions**, adapter les **services** existants, adapter et créer des **rules**, adapter et créer des **tests unitaires**, adapter les **vues**. Voici la liste des développements principaux à effectuer :

BASE DE DONNÉES	créer la table `type_territoire` en relation 1-n avec `identification` créer la table de référence `ref_type_territoire`, chaque type de territoire ayant des dates de validité insérer dans `ref_type_territoire` le jeu de données fourni par le client
MODÈLE	créer `TypeTerritoireEntity` et son fichier .hbm.xml associé créer `RefTypeTerritoireEntity` et son fichier .hbm.xml associé adapter `IdentificationEntity` pour ajouter une `Collection<TypeTerritoireEntity>` et adapter son fichier .hbm.xml créer `TypeTerritoireDTO` adapter `IdentificationDTO`
ACCÈS AUX DONNÉES	créer `RefTypeTerritoireDAO` et son implémentation, afin de pouvoir récupérer les types de territoire valides à une date donnée, et pour récupérer un type de territoire à partir de son code (String)
CONTRÔLEURS	créer `RefTypeTerritoireListController` pour gérer la liste déroulante des types de territoire possible (en vérifiant les validités) créer `TypesTerritoiresSlaveListController`, pour contrôler la liste des types de territoire de la structure sélectionnée. C'est un contrôleur "esclave" (au sens d'Orion) du contrôleur du formulaire global de la structure, car la collection de types de territoire est un attribut de la structure.
ACTIONS	créer `NewTypeTerritoireAction`, qui sera appelée lors du clic sur un bouton pour ajouter un type de territoire à une structure
SERVICES	modifier `GestionStructuresServiceImpl` pour notamment : → adapter les méthodes `init`, `load`, `save`, … qui initialisent les DTO à partir des données de la base et inversement génèrent les entités à persister à partir des DTO du formulaire → créer diverses méthodes de gestion requises par des règles de gestion modifier `ExporterStrcturesServiceImpl` modifier `GestionClotureStructureServiceImpl`
RULES (JAVA)	modifier les rules `RuleRgObligatoire` et `RuleRgControleSurface` pour prendre en compte le nouveau bloc TypeTerritoire (lors de la validation d'une structure) créer la rule `RuleRgRcpsAddTypeter` pour les champs obligatoires et la rule `RuleRgRcps022` lors de l'ajout d'un type de territoire compléter la rule `RuleRgRcps009` pour vérifier à la validation la cohérence des dates des types de territoire d'une structure (ordre logique des dates d'effet et cohérence vis-à-vis des dates de la structure)
RÈGLES (SFD)	création d'une règle RG_RCPS_022 sur l'ordre logique des dates d'effet d'un type de territoire
MESSAGES	création des nouveaux messages nécessaires dans messages-gestion-structures.properties et messages-erreur-info.properties
VUES	modification du fichier structures-coordonnees.xhtml qui gère spécifiquement l'onglet *Coordonnées* du formulaire : → ajout d'un bloc « Types de territoire » → ajout d'une *slavelist* pour afficher sous forme de tableau la liste des types de territoire

	de la structure sélectionnée, avec bouton corbeille sur chaque ligne pour supprimer un type de territoire → ajout des champs de formulaire nécessaires à l'ajout d'un type de territoire à la structure → ajout d'un bouton « Ajouter » pour valider l'ajout d'un type de territoire et rafraîchir la *slavelist* modification de structure-recapitulatif-adresse.xhtml qui gère la section de l'onglet *Récapitulatif et Validation* relative aux données de l'onglet *Coordonnées* : → ajout du bloc « Types de territoire » affichant simplement la *slavelist* (cet onglet est juste en consultation)

Les tests unitaires seront discutés dans le paragraphe 5.4.

Chiffrage

Le **chiffrage** par unité d'œuvre et granularité de cette demande a abouti à une estimation globale de **6 jours complets de développement**, ce qui inclut les phases d'analyse du code nécessaires au chiffrage. Le chiffrage a été validé par le client.

5.3.2. Scripts SQL

Exemple du script de création de la table `type_territoire` en relation 1-n avec `identification` :

```
DROP TABLE IF EXISTS rcps.type_territoire;

CREATE TABLE IF NOT EXISTS rcps.type_territoire (
    id_type_territoire serial NOT NULL,
    o_ust_typter VARCHAR,
    o_ust_ddetypter TIMESTAMP,
    o_ust_dfitypter TIMESTAMP,
    statut_rcps VARCHAR(20),
    date_validation TIMESTAMP,
    prec BIGINT,
    id_identification_identification INT,
    CONSTRAINT type_territoire_pk PRIMARY KEY (id_type_territoire),
    CONSTRAINT identification_fk FOREIGN KEY (id_identification_identification)
        REFERENCES rcps.identification (id_identification)
);

ALTER rcps.type_territoire OWNER to sco_rcps;
```

Et du fichier de mapping Hibernate TypeTerritoireEntity.hbm.xml associé :

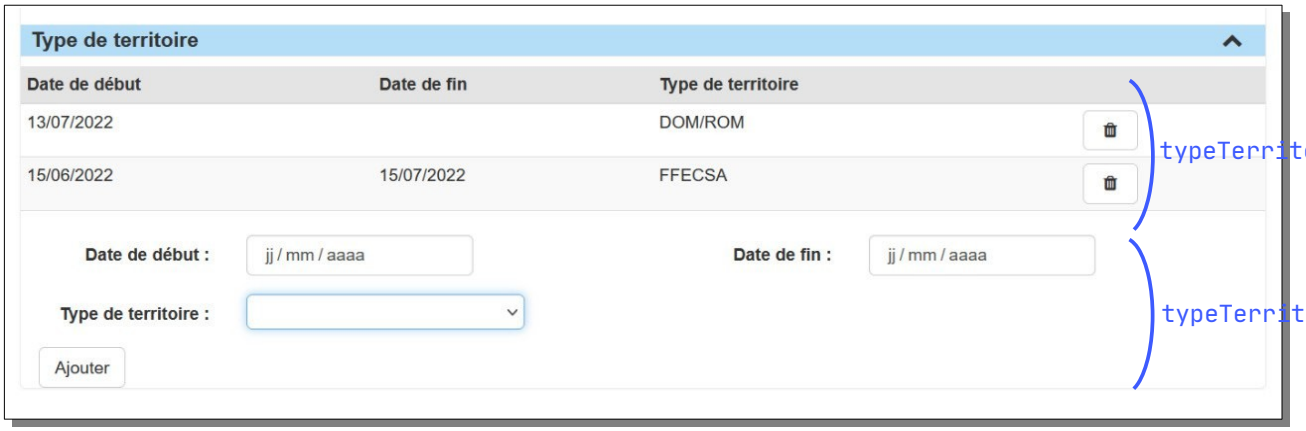
```xml
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class
        name="fr.gouv.agriculture.dsis.renoirh.entities.structures.TypeTerritoireEntity"
        table="type_territoire">

        <id name="idTypeTerritoire" column="id_type_territoire">
            <generator class="sequence">
                <param name="sequence">type_territoire_id_type_territoire_seq</param>
            </generator>
        </id>
        <property name="ustTypter" column="o_ust_typter" />
        <property name="ustDdetypter" column="o_ust_ddetypter" />
        <property name="ustDfitypter" column="o_ust_dfitypter" />
        <property name="statutRcps" column="statut_rcps" />
        <property name="dateValidation" column="date_validation" />
        <property name="prec" column="prec" />
        <many-to-one
            class="fr.gouv.agriculture.dsis.renoirh.entities.structures.IdentificationEntity"
            column="id_identification_identification" name="identification" />
    </class>
</hibernate-mapping>
```

5.3.3. Front-end

Au niveau de l'IHM, voici le bloc qui a été ajouté à l'onglet *Coordonnées* :



Pour rappel, l'ensemble du formulaire, étalé sur plusieurs onglets, est lié à l'instance de `IdentificationDTO` correspondant à la structure sélectionnée. À cet objet, nommons-le `identificationDTO`, j'ai ajouté les 2 attributs suivants :

```
    private TypeTerritoireDTO typeTerritoire;

    private Collection<TypeTerritoireDTO> typeTerritoireDTOs;
```

- `typeTerritoire` est l'objet temporaire lié *(data binding)* aux champs du formulaire ci-dessus. Au chargement de la structure, il est vide.

- `typeTerritoireDTOs` est la collection de types de territoire de la structure. Au chargement de la structure, elle est construite par le service, à partir des données de la base. C'est l'objet utilisé par le contrôleur de la *slavelist* pour produire le tableau des types de territoire de la structure, visible ci-dessus.

Lorsque l'on saisit un nouveau type de territoire et que l'on clique sur **Ajouter**, alors, à supposer que les règles de validation sont passées avec succès, `typeTerritoire` est cloné et son clone ajouté à `typeTerritoireDTOs`, la *slavelist* est rafraîchie et `typeTerritoire` est vidé afin de reproposer un formulaire vide. Inversement, le bouton corbeille présent sur la ligne d'un type de territoire doit permettre de récupérer l'élément de la collection concerné afin de le retirer de la collection. Dans tous les cas il faudra cliquer sur **Enregistrer** pour que ces informations soient sauvegardées en base (mission au service de compiler correctement les différentes informations et de produire les entités à envoyer en base).

Extraits du fichier structure-coordonees.xhtml

```
<o:fieldset id="blocTypeTerritoire">
    <o:legend
        title="#{Messages['structures.type.territoire.type.territoire.header']}">
        #{Messages['structures.type.territoire.type.territoire.header']}
    </o:legend>
    <o:hcontrols>
        <o:grid>
            <o:row>
                <!-- TABLEAU MULTI-OCCURENCES DES TYPES DE TERRITOIRES -->
                <o:slaveList controllerRef="typeTerritoireSlaveListCtrl"
                    parentProperty="typeTerritoireDTOs" id="listTypeTerritoireId">
                    <o:table id="typeTerritoireTable"
                        title="#{Messages['structures.type.territoire.type.territoire.header']}">
                        <o:thead>
                        <o:tbody>
                            <o:repeat var="typeTerritoireDto">
                                <o:fragment>
                                    <o:tr data="true">
                                        <o:td headers="dateDebTypeTerCol"
                                            colspan="#{mode != 'consult' ? 4 : 6}" id="ustDdetypterId">
                                            #{typeTerritoireDto.ustDdetypterStr}
                                        </o:td>
```

*Extrait 1 : déclaration du bloc (o:fieldset) et de la **slavelist**, avec son **contrôleur** et la **propriété de l'objet parent** associée (obligatoirement une collection)*

```
<o:row groupIds="libelleTypeTerGroupeId">
    <o:col6>
        <o:control>
            <o:label size="4">
                #{Messages['structures.type.territoire.ustTypter.label']}
                #{ !empty identificationDto.mapObli['TypeTerritoireEntity.ustTypter'] ? Messages['structures.label.etoile']:':'}
            </o:label>
            <o:comboBox id="typeTerId"
                disabled="#{mode == 'consult' ? true : false}"
                controllerRef="refTypeTerritoireListCtrl"
                title="#{Messages['structures.type.territoire.ustTypter.label']}"
                displayLabelProperty="libelle"
                value="#{identificationDto.typeTerritoire.refTypeTerritoireEntity}"
                labelForNull="#{Messages['structures.type.territoire.liste.type.territoire.vide.label']}"
                size="7" nullable="true">
            </o:comboBox>
        </o:control>
    </o:col6>
</o:row>
```

*Extrait 2 : liste déroulante pour le choix du type de territoire (o:comboBox), avec son **contrôleur** et son **mapping** (value)*

```
<o:button disabled="#{mode == 'consult' ? true : false}"
    actionRef="newTypeTerritoireAction" label="Ajouter" />
```

*Extrait 3 : bouton Ajouter, avec l'**action** associée*

Pour rappel, dans les exemples ci-dessus, le **contrôleur** et l'**action** sont mappés vers leurs **classes Java** correspondantes dans les fichiers de configuration du kernel.

5.3.4. Code Java - Exemples

Des centaines de ligne de code ont été ajoutées pour cette Jira. Je ne montrerai ici que quelques courts exemples en lien avec les extraits de xhtml vus ci-dessus : un **contrôleur**, une **action** et une **rule**.

Un contrôleur de liste qui fait appel à un DAO

La liste déroulante de l'extrait 2 ci-dessus déclare un **contrôleur** (attribut `controllerRef`). Si la liste reprenait tel quel l'ensemble du contenu de la table `ref_type_territoire`, nous pourrions simplement déclarer (dans la configuration du kernel) que ce contrôleur est la classe `DefaultDAOListController` du framework Orion, avec comme *businessClass* la classe `RefTypeTerritoireEntity` (mappée sur la table).

Mais ici la liste ne doit proposer que les types de territoire **valides à la date du jour**. Il y a donc un traitement à effectuer et il faut un contrôleur spécifique pour cela.

Il a donc fallu créer une classe `RefTypeTerritoireListController` qui étend `DefaultDAOListController` et réécrit sa méthode `findBusinesses` chargée de produire la collection d'éléments à afficher :

```
public class RefTypeTerritoireListController extends DefaultDAOListController {

   private static final Logger LOGGER = LoggerFactory.getLogger(RefTypeTerritoireListController.class);

   @Inject
   private RefTypeTerritoireDAO refTypeTerritoireDAO;

   public void setRefTypeTerritoireDAO(RefTypeTerritoireDAO refTypeTerritoireDAO) {
      this.refTypeTerritoireDAO = refTypeTerritoireDAO;
   }

   @Override
   public Collection findBusinesses(OrionBusinessContext orionBusinessContext) {

      Collection<RefTypeTerritoireEntity> listTypeTerritoireEntities = new ArrayList<RefTypeTerritoireEntity>();

         try {
            listTypeTerritoireEntities = refTypeTerritoireDAO.findByDateRef(new Timestamp(System.currentTimeMillis()));
         } catch (Exception exc) {
            String msg = "Can't select business with Service " + refTypeTerritoireDAO.getClass();
            LOGGER.error(msg, exc);
         }

      return listTypeTerritoireEntities;
   }
}
```

Comme on le voit, la méthode `findBusinesses`, pour construire la collection attendue, appelle, à la date présente, la méthode `findByDateRef` du DAO. Il a fallu écrire une implémentation pour ce DAO et écrire la méthode en question :

```java
@Override
@Transactional
public Collection<RefTypeTerritoireEntity> findByDateRef(Timestamp dateRef) throws Exception {

    QueryBuilder queryBuilder = new QueryBuilder();
    queryBuilder = (QueryBuilder) queryBuilder.from(RefTypeTerritoireEntity.class);
    SimpleExpression dateDebutValide = queryBuilder.lessEquals("dateDebVal", dateRef);
    SimpleExpression dateFinValideNotNull = queryBuilder.greaterEquals("dateFinVal", dateRef);
    SimpleExpression dateFinNull = queryBuilder.isNull("dateFinVal", null);

    Disjunction dateFinValide = queryBuilder.or(dateFinValideNotNull, dateFinNull);

    queryBuilder.where(queryBuilder.and(dateDebutValide,dateFinValide));

    OrionBusinessContextImpl orionBusinessContext = new OrionBusinessContextImpl();
    orionBusinessContext.setQuery(queryBuilder.getQuery());

    Collection<RefTypeTerritoireEntity> collection = findBy(orionBusinessContext);

    return collection;
}
```

Il est à noter que, bien qu'étant basé sur Hibernate, Orion ajoute une riche surcouche qui est parfois déroutante. Ainsi, l'annotation `@Transactional` est réécrite par Orion et les classes `QueryBuilder`, `SimpleExpression` et `Disjunction` sont propres à Orion.

Une action pour ajouter un type de territoire

Une **action** est une classe qui étend la classe `BusinessAction` d'Orion, en réécrivant sa méthode `execute`.

Ici, notre action (de l'extrait 3 ci-dessus) doit se charger d'ajouter les données du formulaire à la collection de types de territoire à condition que les règles de validation soient respectées. Cette action va devoir :

1. Récupérer le contrôleur du formulaire
2. Grâce au contrôleur, récupérer l'objet lié (identificationDTO)
3. De cet objet, récupérer toutes les données sur les types de territoire (formulaire et collection)
4. Construire un nouveau TypeTerritoireDTO, à partir des données du formulaire (clone)
5. Vérifier si l'ensemble des Rules est respecté (2 Rules ici, pour chacune, création d'un RuleReport)
6. **Si les règles ne sont pas respectées** (= si l'un des RuleReport contient des erreurs), jeter une exception

   Si les règles sont respectées :
7. ajouter ce nouveau TypeTerritoireDTO à la collection,
8. remettre à null les champs du formulaire
9. rafraîchir la slavelist pour effectivement montrer le nouvel élément dans le tableau.

Voici le code de cette Action, correspondant aux 9 étapes ci-dessus :

```
public class NewTypeTerritoireAction extends BusinessAction {

  /**
   * @inheritDoc
   */
  @Override
  public Object execute() throws Exception {

    Controller businessController = getController(); (1)
    Object business = null;

    if (businessController instanceof CreationModificationStructureFormController) {

      CreationModificationStructureFormController controller = (CreationModificationStructureFormController) businessController;
      IdentificationDTO identificationDTO = (IdentificationDTO) controller.getFormModel().getObject(); (2)

      if (identificationDTO != null) {
        TypeTerritoireDTO typeTerritoireDTO = identificationDTO.getTypeTerritoire();
        Collection<TypeTerritoireDTO> typeTerritoireDTOs = identificationDTO.getTypeTerritoireDTOs(); (3)

        // Lecture paramètres entrés
        RefTypeTerritoireEntity refTypeTerritoireEntity = typeTerritoireDTO.getRefTypeTerritoireEntity();
        Date ustDdetypter = typeTerritoireDTO.getUstDdetypter();
        Date ustDfitypter = typeTerritoireDTO.getUstDfitypter();

        // Affectation valeurs dans nouveau DTO
        TypeTerritoireDTO newTypeTerritoireDTO = new TypeTerritoireDTO();
        newTypeTerritoireDTO.setRefTypeTerritoireEntity(refTypeTerritoireEntity);
        if (refTypeTerritoireEntity != null) {
          newTypeTerritoireDTO.setUstTypter(refTypeTerritoireEntity.getCode());
        }
        newTypeTerritoireDTO.setDateValidation(identificationDTO.getDateValidation()); (4)
        newTypeTerritoireDTO.setUstDdetypter(DateUtils.getDateWithoutTime(ustDdetypter));
        newTypeTerritoireDTO.setUstDdetypterStr(DateUtils.getStringFromDate(ustDdetypter));
        newTypeTerritoireDTO.setUstDfitypter(DateUtils.getDateWithoutTime(ustDfitypter));
        newTypeTerritoireDTO.setUstDfitypterStr(DateUtils.getStringFromDate(ustDfitypter));
        newTypeTerritoireDTO.setIdentification(identificationDTO);

        // Règles à respecter

          // champs obligatoires
        RuleRgRcpsAddTypeter ruleRgRcpsAddTypeter = new RuleRgRcpsAddTypeter();
        RuleReport ruleReport = ruleRgRcpsAddTypeter.validateObject(newTypeTerritoireDTO);
        if (ruleReport.containsError()) {
          throw new RuleException(ruleReport);
        } (5) (6)

          // bon ordre des dates de début et fin
        RuleRgRcps022 ruleRgRcps022 = new RuleRgRcps022();
        ruleReport = ruleRgRcps022.validateObject(newTypeTerritoireDTO);
        if (ruleReport.containsError()) {
          throw new RuleException(ruleReport);
        }

        // Si règles respectées
        typeTerritoireDTOs.add(newTypeTerritoireDTO); (7)

        // Réinit valeurs dans formulaire
        typeTerritoireDTO.setRefTypeTerritoireEntity(null);
        typeTerritoireDTO.setUstDdetypter(null);
        typeTerritoireDTO.setUstDfitypter(null); (8)

        // Maj de la liste des types de territoires
        Collections.sort((List) typeTerritoireDTOs, new TypeTerritoireComparator());
        TypeTerritoireSlaveListController typeTerritoireSlaveListCtrl = (TypeTerritoireSlaveListController) controller.getControllerForName("typeTerritoireSlaveListCtrl");
        typeTerritoireSlaveListCtrl.processEvent(new ControllerEvent(typeTerritoireSlaveListCtrl, Events.DO_INITIALIZE, typeTerritoireDTOs)); (9)
        businessController.processEvent(new ControllerEvent(businessController, Events.VIEW_MODIFIED, business));
      }
    }

    return super.execute();
  }
}
```

Ce code va nous permettre d'aborder un dernier exemple, celui d'une **Rule**. L'action ci-dessus en utilise deux : `RuleRgRcpsAddTypeter` et `RuleRgRcps022`.

Une Rule pour définir les champs obligatoires

Une note ultérieure du client a précisé que les champs *date de début* et *type de territoire* étaient obligatoires pour l'ajout d'un type de territoire. Il a donc fallu créer une rule `RuleRgRcpsAddTypeter` pour cela.

Une rule étend la classe abstraite `AbstractRuleCommon` du framework et implémente l'interface `Rule`. De fait, il lui reste à réécrire la méthode `validateObject`. Cette méthode :

1. instancie un `RuleReport` vide
2. récupère notre `TypeTerritoireDTO` candidat à l'ajout
3. pour chaque champ obligatoire, si le champ est null, crée un `RuleMessage` et l'ajoute au `RuleReport`. Ce `RuleMessage` contient un message (texte), l'id du champ et une sévérité.
4. retourne le `RuleReport` pour traitement (ici, par notre Action) :

```
public class RuleRgRcpsAddTypeter extends AbstractRuleCommon implements Rule {

   @Override
   public String getName() {
      return "ruleRgRcpsAddTypeter";
   }

   @Override
   public RuleReport validateObject(Object object) {

      RuleReport ruleReport = new RuleReport();  (1)

      if (object instanceof TypeTerritoireDTO) {
         final TypeTerritoireDTO typeTerritoireDTO = (TypeTerritoireDTO) object;  (2)

         RefTypeTerritoireEntity ustTypter = typeTerritoireDTO.getRefTypeTerritoireEntity();
         if (ustTypter == null) {
            RuleMessage ruleMessage = new RuleMessage(Messages.getMessage("M_RCPS_UstTypter"),
                  "typeTerId", MessageSeverity.ERROR);
            ruleReport.addMessage(ruleMessage);
         }  (3)

         Date ustDdetypter = typeTerritoireDTO.getUstDdetypter();
         if (ustDdetypter == null) {
            RuleMessage ruleMessage = new RuleMessage(Messages.getMessage("M_RCPS_UstDdetypter"),
                  "typeTerDateDebId",MessageSeverity.ERROR);
            ruleReport.addMessage(ruleMessage);
         }  (3)

      }
      return ruleReport;  (4)
   }
}
```

5.3.5. Bilan de la Jira

Si la demande de départ semblait relativement simple, les implications en termes de développement et d'adaptation du code existant se sont révélées très importantes.

La commande `git diff --stat` entre le commit du code de cette Jira et l'état précédent, bien qu'imprécis (puisque les lignes d'import par exemple sont prises en compte, tout comme celles ne contenant qu'une accolade), indique **49 fichiers modifiés ou créés** et plus de **2300 lignes nouvelles**, répartis comme suit :

- 35 classes Java (3 `Action`, 4 `Controller`, 1 `DAO` + 1 `DAOImpl`, 2 `DTO`, 5 `Entity`, 6 `Rule`, 4 `Service`, 8 `Test`, 1 `Comparator`)
- 3 fichiers SQL
- 7 fichiers XML de configuration (4 relatifs au mapping Hibernate, 3 à la configuration du kernel)
- 2 fichiers XHTML de vues

- 2 fichiers PROPERTIES contenant les messages

Le fait qu'autant de classes soient à modifier ou créer pour traiter la demande pose des questions sur l'architecture de l'application, la factorisation du code, les niveaux d'abstraction utilisés.

Le chiffrage de 6 jours s'est révélé correct.

Remarque : logiquement la phase suivante sera d'intégrer ces nouvelles données (les types de territoire) aux bordereaux produits par l'agent à destination de RenoiRH, mais cette demande n'a pour l'instant pas été formulée par le client (et nécessite au préalable une adaptation de RenoiRH et de ses tables).

5.4. Tests et validation

5.4.1. Tests unitaires

La mise en place de tests unitaires pertinents dès les phases initiales du développement d'une application est cruciale. Ils doivent couvrir l'essentiel des méthodes des classes métiers et permettent de vérifier continuellement leur bon fonctionnement.

En plus de s'assurer du bon comportement des logiques métiers au moment de leur développement, les tests unitaires permettent, et c'est leur intérêt majeur, de détecter le plus tôt possible les **régressions** résultant d'une **évolution du code**. Un changement dans le code à un endroit donné peut produire des effets inattendus à un autre endroit de l'application, qu'il serait très difficile de détecter sans la présence de tests unitaires exhaustifs et précis, que l'on fait tourner régulièrement. Plus le temps passe entre l'apparition d'une régression et sa détection (avec donc potentiellement d'autres développements effectués entre les deux), plus sa résolution aura un impact fort en termes de ressources.

Demande du Ministère

Officiellement, le Ministère demande une **couverture de 100 %** concernant les tests unitaires, les rapports SonarQube faisant foi *(voir ci-dessous pour une courte présentation de SonarQube)*. Dans la réalité, une telle couverture est utopique pour les projets conséquents, et nous nous efforçons donc d'avoir la couverture maximale, et que dans tous les cas la couverture ne diminue pas entre l'état de l'application avant un développement par Inetum et l'état après. Par ailleurs, couvrir une méthode est une chose, la couvrir efficacement et de manière pertinente en est une autre. La **qualité de la couverture** est donc au moins aussi importante que son étendue.

jUnit, EasyMock, PowerMock

Pour RCPS nous utilisons le framework de tests classique des projets du Ministère : **Unitils**.

> ***Unitils** est un framework qui permet d'agréger simplement plusieurs frameworks comme jUnit, Spring, EasyMock, … et de réduire le volume de code nécessaire à l'écriture des tests en faisant grand usage des annotations.*

Unitils est utilisé avec **EasyMock** et **PowerMock**.

__EasyMock__ est une alternative à Mockito, c'est un framework permettant de créer des __Mocks__, objets simulés qui reproduisent un comportement voulu de manière contrôlée. Ce sont des éléments essentiels des tests unitaires car ils permettent de simuler le résultat retourné par une méthode appelée par la méthode à tester, sans réellement appeler cette méthode (le but du test unitaire étant de tester 1 et 1 seule méthode à la fois).

__PowerMock__ étend les fonctionnalités d'EasyMock, permettant notamment de simuler des méthodes statiques et privées. On utilise PowerMock là où EasyMock ne suffit pas.

Tests unitaires liés à la Jira RCP-138

L'implémentation de la demande RCP-138 a nécessité de modifier ou créer **8 classes de tests** et d'ajouter des centaines de lignes de code. Une partie de ce développement a consisté à **adapter les tests existants** afin qu'ils intègrent la présence du nouveau bloc, une autre partie à **créer de nouveaux tests** pour les nouvelles méthodes introduites dans le code.

Prenons l'exemple de la rule `RuleRgRcps022`, nouvellement créée et qui stipule que la date de fin d'un type de territoire, si elle est renseignée, doit être postérieure ou égale à la date de début. Sans entrer dans le détail de cette rule, son fonctionnement doit notamment garantir que, une date de début étant saisie :

- la **non saisie** d'une date de fin mène à un **RuleReport vide**
- la saisie d'une **date de fin égale** mène à un **RuleReport vide**
- la saisie d'une **date de fin postérieure** mène à un **RuleReport vide**
- la saisie d'une **date de fin antérieure** mène à un **RuleReport contenant 1 RuleMessage**

Pour rappel, la méthode qui implémente la règle dans la classe rule est la méthode validateObject. C'est donc elle qu'il faut tester pour les 4 cas décrits ci-dessus. Pour cela, une classe `RuleRgRcps022Test` a été créée, donc nous donnons ci-dessous quelques extraits.

Voici les imports liés aux tests :

```
import org.junit.Assert;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.powermock.api.easymock.PowerMock;
import org.powermock.core.classloader.annotations.PrepareForTest;
import org.powermock.modules.junit4.PowerMockRunner;
```

Voici la déclaration annotée de la classe :

```
@RunWith(PowerMockRunner.class)
@PrepareForTest({RuleRgRcps022.class})
public class RuleRgRcps022Test {
```

Voici l'implémentation du 4ème test décrit ci-dessus, menant à un RuleReport non vide :

```
@Test
public void testValidateObject_date_deb_sup_date_fin() throws Exception {
   RuleRgRcps022 ruleRgRcps022 = PowerMock.createNicePartialMockForAllMethodsExcept(RuleRgRcps022.class, "validateObject"); (1)

   TypeTerritoireDTO typeTerritoireDTO = new TypeTerritoireDTO();
   IdentificationDTO identificationDTO = new IdentificationDTO();

   SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
   Date dateDeb = sdf.parse("24/03/2020");
   Date dateFin = sdf.parse("23/03/2020");

   identificationDTO.setTypeTerritoire(typeTerritoireDTO);
   typeTerritoireDTO.setUstDdetypter(dateDeb);
   typeTerritoireDTO.setUstDfitypter(dateFin);                                  (2)

   PowerMock.replayAll(); (3)

   RuleReport report = ruleRgRcps022.validateObject(identificationDTO); (4)

   PowerMock.verifyAll(); (5)

   Assert.assertNotNull(report);
   assertFalse(report.isEmpty());
   List<RuleMessage> messages = report.getMessages();
   Assert.assertNotNull(messages);
   Assert.assertEquals(1, messages.size());                                     (6)
}
```

1. On demande à *mocker* (instancier) la classe `RuleRgRcps022` en simulant toutes ses méthodes sauf `validateObject`, la méthode à tester
2. On crée une structure ayant un type de territoire avec une date de fin antérieure à sa date de début
3. On « rejoue » les classes et on *mocke* tous les objets connus de PowerMock
4. On récupère le `RuleReport` retourné par la méthode `validateObject` de l'objet rule simulé
5. On vérifie que toutes les opérations de *mock* se sont bien déroulées
6. On vérifie les résultats attendus (le report n'est pas null, il n'est pas vide, il contient une liste de `RuleMessage` composé d'un seul élément)

5.4.2. SonarQube

SonarQube *est un logiciel open source de* ***monitoring de qualité de code****. Il supporte de nombreux langages et permet détecter les défauts dans le code et de produire des rapports, notamment sur les duplications de code, le niveau de documentation et la* ***couverture du code par les tests unitaires****.*

Sa puissance réside dans le fait que les analyses peuvent être entièrement ***automatisées*** *et* ***intégrées*** *notamment à Maven, Éclipse et des serveurs d'intégration continue comme Jenkins.*

Récupérant une application existante dans un état donné, nous ne pouvons avoir de critères absolus quant à la qualité du code candidat à la livraison. Nous nous engageons néanmoins à ce que la qualité du code après développement soit au moins égale à celle avant, et pour cela SonarQube est utilisé, notamment pour mesurer les couvertures du code par les tests.

5.5. Remarques et conclusion

À l'issue de mon stage la majeure partie du lot 3 a été codée. J'ai principalement travaillé sur la webapp mais ai également traité des évolutions qui concernent l'agent (RCP-133, RCP-134, RCP-135, RCP-136). Rappelons le contenu du lot :

Lot3

- Fiche RCP-108 : Flux structure - gestion des compte-rendu d'intégration
- Fiche RCP-129 : IHM Structures - Onglet « Récapitulatif et validation » : Absence données et blocage validation
- Fiche RCP-130 : IHM Structures - Onglet « Coordonnées » : Pouvoir modifier une adresse
- Fiche RCP-131 : IHM Structures - Alimentation Liste déroulante « Responsable mise à jour »
- Fiche RCP-132 : IHM Structures - Alimentation Liste déroulante « MAPS »
- Fiche RCP-133 : Flux structure - ZE0F Adresses
- Fiche RCP-134 : Flux structure - ZEXA Zone de résidence
- Fiche RCP-135 : IHM Structures et Flux Structures - Rajout d'un témoin Structure RenoiRH
- Fiche RCP-136 : IHM Structures et Flux Structures - Suppression du témoin générique
- Fiche RCP-137 : IHM Structures et Flux Structures - Liens organisationnels (structure mère et structure administrative)
- Fiche RCP-138 : IHM Structures - Type de territoire

Les derniers jours ont été ponctués d'échanges avec le responsable fonctionnel, en charge de la mise à jour des SFD.

Le gros chantier restant est la demande RCP-108, qui consiste en l'implémentation de la gestion par l'agent des compte-rendus produits par RenoiRH. Pour rappel, lorsqu'un fichier de bordereaux est produit par l'agent à destination de RenoiRH, les structures concernées par les modifications doivent être fermées à la modification dans l'attente d'un compte-rendu d'intégration de RenoiRH, qui est un fichier dans un format bien précis déposé sur un serveur de fichiers et indiquant les structures rejetées et les raisons des rejets. Toute l'implémentation est à faire, y compris l'affichage dans l'IHM des informations nécessaires si le compte-rendu fait état de modifications rejetées.

Si les SFD sont à livrer début septembre, le code est lui à livrer début octobre, ce qui laisse suffisamment de temps aux développements nécessaires.

Difficultés rencontrées

La mission a été extrêmement formatrice quant aux différents process d'une relation au forfait avec un client. Expression de besoins, échanges avec le client, analyses de l'existant, chiffrages, validation, développement, … ces étapes se sont bien déroulées mais ont aussi été accompagnées d'un certain nombre de difficultés.

Des technologies relativement anciennes

Java 6, Glassfish 3.1, Maven 3.2.5, … la préparation de l'environnement de développement a mis un certain temps compte tenu de l'âge des versions requises et des problèmes de compatibilité.

Un framework maison

Le framework Orion dispose d'une documentation relativement dense produite par le Ministère, mais qui demeure insuffisante. De nombreuses zones d'ombre subsistent, qu'il est impossible de dissiper par des recherches sur internet. L'impossibilité de trouver des informations, témoignages, explications sur les forums constitue un grand manque, les échanges avec les pairs faisant partie de la vie du développeur.

Des choix architecturaux discutables et le fonctionnement au forfait

Les applications sont développées depuis plusieurs années, par des équipes différentes au cours du temps, et récupérées à un instant donné dans un état donné, pour poursuite du développement. Le fonctionnement au forfait (par opposition à l'assistance technique) suppose qu'une demande précise est formulée par le client à un instant donné sur le code existant, chiffrée, validée, livrée. Cette demande consistera rarement en un nettoyage du code existant, évolution aux effets invisibles aux yeux du client malgré son coût, mais bien à créer une fonctionnalité nouvelle. Ce fonctionnement a tendance à produire une suite de patchs qui font continuellement enfler le volume de l'application, au détriment d'une vision à long terme qui priorise une architecture saine, économe en code, et minimisant les efforts à fournir pour les développement ultérieurs.

Ainsi, les classes de plus de 2000 lignes ne sont pas rares et la séparation des couches n'est pas toujours respectée. Les duplications de code sont nombreuses et les nommages, hétéroclites, ne respectent pas toujours les bonnes pratiques. La Javadoc et les commentaires sont trop peu nombreux. Il n'a donc pas été facile de naviguer dans les méandres des méthodes, classes, packages, fichiers *xml* et *properties* pour comprendre et s'approprier le code de l'application.

6. Bilan et perspectives

Ces 3 mois de stage au sein l'équipe Ministère de l'Agriculture de la division Application Services Sud-Ouest d'Inetum se sont très bien passés. L'équipe était très sympathique, l'environnement de travail agréable.

Après une première phase de montée en compétences sur les technologies utilisées (notamment le framework Orion), j'ai été rapidement intégré au quotidien de l'équipe en tant que développeur à part entière et me suis vu assigner des missions concrètes, en relation avec le client. J'ai pu échanger, suggérer des solutions, chiffrer et développer comme tout autre membre de l'équipe, avec néanmoins la flexibilité de mon statut de stagiaire.

Le projet sur lequel j'ai été placé avait ceci d'intéressant qu'il couvrait un large spectre de compétences, du back-end au front-end et à la base de données. En ceci il a mobilisé l'essentiel des compétences composant la formation de Concepteur Développeur d'Applications.

Cette expérience positive a validé mon choix de poursuivre dans le monde du développement.

Mon travail a été reçu positivement et une proposition de CDI m'a été faite, que j'ai acceptée. Je serai donc dès septembre 2022 membre de l'équipe du Ministère de l'Agriculture au sein d'Inetum Toulouse, en tant que développeur.

Annexes

Annexe 1 : SFD – Exemple de Cas d'Utilisation (CU)

Par commodité nous prenons ici un CU relativement simple et court de l'application *RCPS-webapp* : l'export de la liste des structures, depuis l'IHM, en un fichier au format csv.

- Description générale du CU :

Exporter la liste des structures	
Type	Cas d'utilisation
Objectif	Ce CU permet d'exporter une liste des structures à partir de l'écran d'accueil du Domaine Structure ou depuis l'écran Rechercher structures.
Acteur(s) principal(ux)	Gestionnaire RCPS
Acteur(s) secondaire(s)	Sans objet
Déclenchement du cas	Ce cas se déclenche lorsqu'un utilisateur clique sur l'un des boutons suivants : • Exporter la liste des structures (menu vertical du Domaine Structure) ; • Exporter la liste (écran Rechercher structures, une fois une recherche lancée par l'utilisateur).
Description générale	Ce cas d'utilisation d'export de structures, permet au système de rechercher une liste de structures pour la formaliser dans un fichier au format CSV.
Précondition (s)	L'utilisateur a cliqué sur l'un des boutons suivants. • Exporter la liste des structures ; • Exporter la liste.
Postcondition (s)	Un fichier au format CSV est créé.

CU Exporter la liste des structures – Description générale

- Description détaillée du CU :

Exporter la liste des structures	
Type	Scénario
Description	Ce déroulement correspond au cas nominal de l'export de structures dans l'application RCPS. Si le CU est déclenché depuis le menu vertical du domaine structures, alors l'ensemble des structures est exporté. Si le CU est déclenché depuis l'écran Rechercher structures, alors la liste de structures résultant de la recherche est exportée.
Préconditions spécifiques	Sans objet
Postconditions spécifiques	Sans objet
Description	
Étape	Description
1	Identification des informations à exporter : A partir du menu vertical du Domaine Structure, l'utilisateur clique sur le bouton « Exporter la liste des structures ». Le système recherche pour toutes les structures au statut 'valide' A partir de l'écran Rechercher structures, l'utilisateur effectue une recherche puis clique sur le bouton « Exporter la liste ». Le système identifie la liste de structures résultant de la recherche.

2	Traitement de récupération des données : Pour chaque structure identifiée, le système récupère via l'identifiant de la structure l'ensemble des données nécessaires à l'alimentation du fichier au format CSV. Le système exporte les structures dans un fichier tableur dont la structure est décrite dans le paragraphe Description du fichier «Export de la liste de structures ». Le fichier est nommé selon la nomenclature suivante : [Application]_[Type de flux]_[Code SI-poste]_[horodatage].csv • Application = 'RCPS' ; • Type de Flux = 'S' pour Structure ; • Code SI-poste = 'SIP' ; • Horodatage de génération = Date et heure de génération de fichier au format : AAAAMMJJ-HHMNSSMS. Trier les structures à exporter sur leur 'identifiant' par ordre croissant.
3	Fin de traitement : Le système permet à l'utilisateur de télécharger le fichier au format CSV créé, via l'outil de consultation & téléchargement du navigateur.
4	Le CU se termine.

CU Exporter la liste des structures – Description détaillée

Annexe 2 : SFD – Exemples de règles de gestion

RG	Libellé	Description	Message
HAB_RCPS_01	Habilitation pour le profil Gestionnaire RCPS	Un utilisateur dont le rôle est Gestionnaire RCPS est habilité à accéder aux différents écrans et fonctionnalités des domaines Structures et Postes.	
RG_RCPS_001	Rechercher la présence de postes actifs sur une structure à la date de clôture	Rechercher s'il existe au moins un poste actif lié à la structure à la date de clôture, administrativement [RG_RCPS_070] ou opérationnellement [RG_RCPS_071]. Si c'est le cas, afficher le message d'erreur M_RCPS_002	M_RCPS_002
RG_RCPS_004	Bouton `Supprimer'	La suppression d'une structure n'est possible que si la structure est au statut 'En cours'.	
RG_RCPS_006	Résultat de la recherche des structures	Si le résultat de la recherche est trop volumineux (il dépasse 200 structures), alors ne pas afficher une liste tronquée mais informer l'utilisateur avec le message M_RCPS_005.	M_RCPS_005
RG_RCPS_007	Bouton 'Clôturer'	La clôture d'une structure n'est possible que si la dernière occurrence de la structure [RG_RCPS_008] est : • au statut 'Valide' • n'a pas de date de clôture de renseignée • n'a pas d'occurrence au statut 'En cours'.	
RG_RCPS_008	Dernière occurrence d'une structure	La dernière occurrence d'une structure correspond à l'occurrence de la structure dont le statut est 'En cours', sinon 'Valide'. Note technique : Identification.statut_rcps	
RG_RCPS_009	Validité d'une date	La date d'une entité doit être • postérieure ou égale à la date de création de la structure (*Identification.o_ust_dcreat*) • antérieure ou égale à la date de clôture de la structure si elle est renseignée (*Identification.o_ust_dferme*) Si ce n'est pas le cas, afficher le message bloquant suivant M_RCPS_012	M_RCPS_012
RG_RCPS_010	Contrôle de format Téléphone et fax	Le numéro associé aux types «Téléphone » et «Fax » (Ref_type_numero.code = 'TPR' ou 'CPR') doit être une suite de 10 chiffres commençant par un '0'. Peu importe la position d'éventuels espaces dans cette suite. Si un numéro ne commence par un '0' et/ou ne contient pas exactement 10 chiffres, le système affiche pour chaque numéro concerné le message bloquant suivant : M_RCPS_010	M_RCPS_010
RG_RCPS_011	Contrôle de format e-mail	Le numéro associé au type « email » doit être au format suivant : [adresse]@[domaine].[extension]	M_RCPS_011

SFD – Exemples de règles de gestion

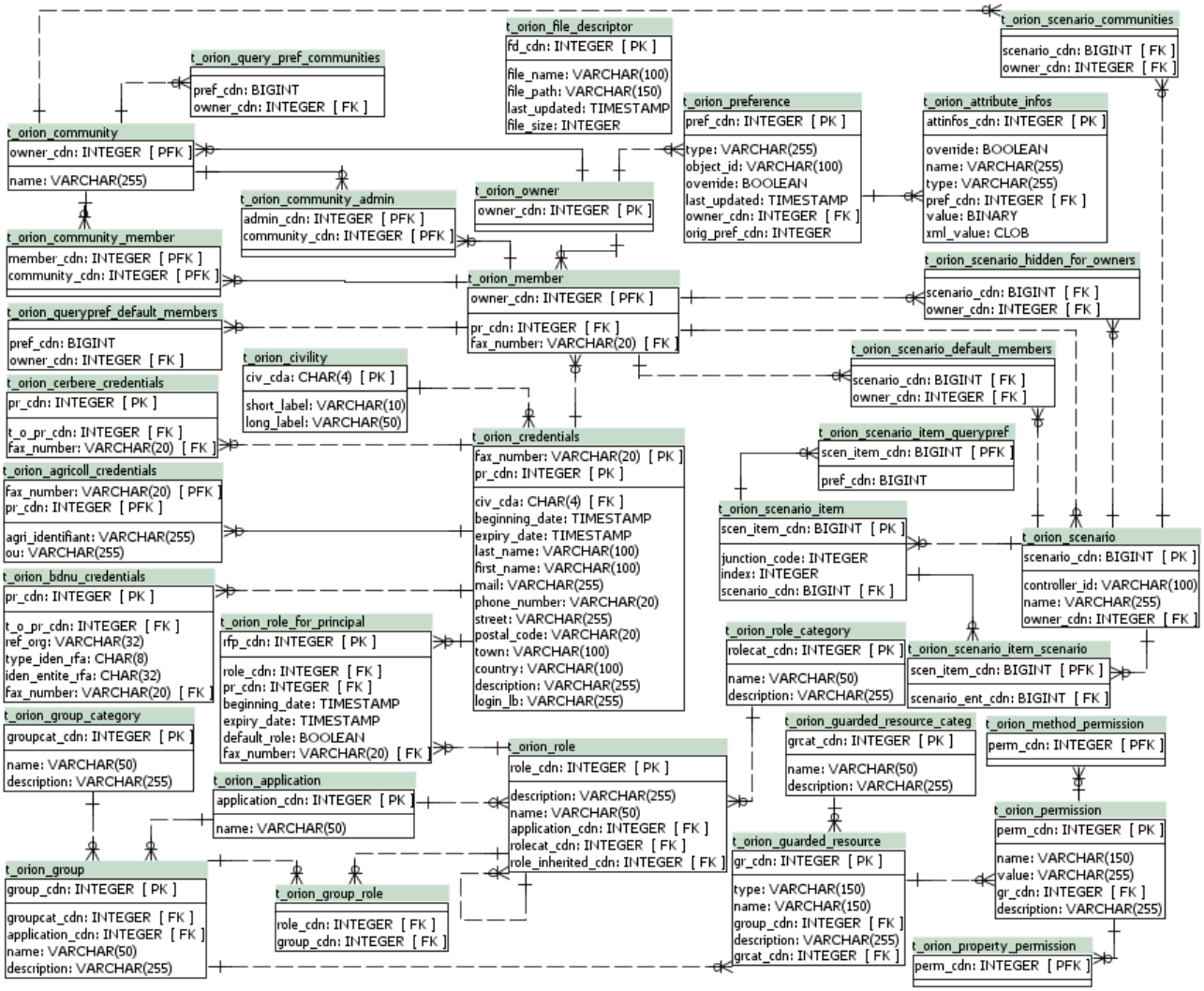
Annexe 3 : Tableau de chiffrage pour l'unité d'œuvre POLIS

Calcul Granularité UO POLIS

Création/modification d'une liste

Élément particulier	Niveau de complexité	Base de calcul	Nombre	Note de complexité
Alimentation de la liste	Alimentation simple faite automatiquement par Orion : toutes les données de la classe, filtrées selon le contexte de navigation	Liste		0
	Les données ne sont pas en base, mais doivent être en dur dans le code	Liste		0
	Filtrer sur une information contenue dans le contexte de navigation, mais non appliquée automatiquement par Orion	Information		0
	Filtrer sur une information inconnue du contexte de navigation	Information		0
	Liste basée sur une classe non mappée (utilisation de façade)	Liste		0
	Alimentation via Webservices			
	Réutilisation d'un webService déjà utilisé	Liste		
	Simple: un seul webservice appelé	Liste		0
	Complexe:plusieurs WS ou WS soap,...	Liste		0
	Application de critères de recherche complexes non automatisés par Orion. La complexité est	critère		0
Présentation de la liste	Mise en forme std orion (avec paragraphe)	propriété		0
	Modifier la présentation d'une propriété. Le rendu	propriété		0
	Le rendu d'une propriété change selon une règle fonctionnelle simple (calculée sur les informations de l'objet métier)	propriété		0
	Le rendu d'une propriété change selon une règle fonctionnelle complexe (calculée sur les informations d'autres objets métiers) (si très complexe voir traitement spécifique	propriété		0
	Liste paragraphe : les propriétés à afficher ne sont pas en colonne	Liste		0
Liendepuis la liste	Lien sur une page de données liée à un élément de la liste courante (lien sur chaque ligne ou sous la liste)	Lien		0
Actions sur la liste	Action standard simplement à poser (Créer, Supprimer, Copier, Coller, Éditions)	action		0
	Complément pour demande de confirmation par javascript	action		0
	Complément pour demande de confirmation par page spécifique (création du formulaire	action		0
	Complément pour javascript autre que demande de confirmation (ex: Ajax)	action		0
	Complément pour message de retour	action		0
	Griser une action selon une règle fonctionnelle	action		0
Traitement spécifique fonctionnel	Pour évaluer la complexité, compter le nombre de métiers intervenant dans le traitement			
	Simple (par ex. 1 à 2 métiers concernés)	traitement		0
	Complexe (par ex. 3 à 5 métiers concernés)	traitement		0
	Très complexe (par ex. + de 5 métiers	traitement		0
Habilitations : ressource à protéger sur la liste	Ressource à protéger suivant une condition simple (booléen)	ressource		0
	Ressource à protéger suivant une expression complexe - complexe	ressource		0
	Ressource à protéger suivant une expression complexe - très complexe	ressource		0
			Granularité	**0**
			Niveau de granularité	**TRES FAIBLE**

Annexe 4 : Modèle de données des tables `t_orion_*`



Ces tables concernent les données de configuration et d'habilitations d'Orion et doivent être présentes pour le bon fonctionnement du framework.

Authentification d'un utilisateur :

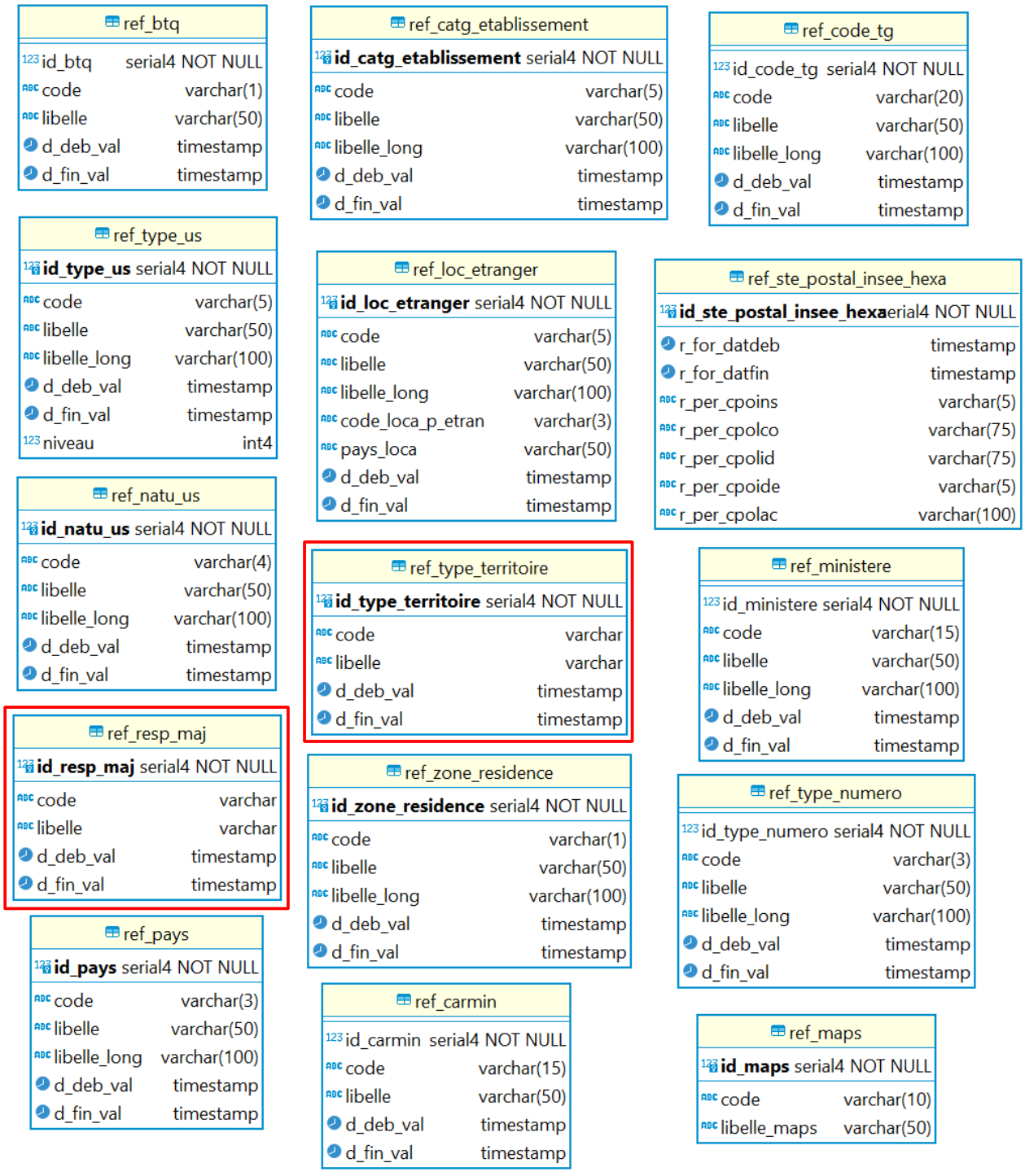
La table `t_orion_credentials` stocke les informations des utilisateurs. Plusieurs tables héritent de `t_orion_credentials`, en effet un utilisateur peut être un utilisateur Agricoll (`t_orion_agricoll_credentials`), un utilisateur BDNU (`t_orion_bdnu_credentials`), …

La table `t_orion_role_for_principal` associe chaque utilisateur à un rôle (`t_orion_role`), et chaque rôle est affecté à une application (`t_orion_application`). Un utilisateur peut donc avoir différents rôles pour plusieurs applications.

Les autres tables concernent les autres configurations Orion, notamment relatives à la sécurité.

Ce modèle de données des tables Orion est commun à de nombreuses applications Orion.

Annexe 5 : Modèle de données des tables de référence



Les 2 tables encadrées en rouge sont les 2 tables que j'ai ajoutées suite aux Jira RCP-131 (`ref_resp_maj`) et RCP-138 (`ref_type_territoire`).

Annexe 6 : Modèle de données des tables paramétrant la création de bordereaux