Développement d'outils de stockage de données et métadonnées collectées sur un pilote de traitement d'eaux usées

ISIB-ELIN-TFE-04/2015-16

Mémoire

Kevin Fuks

Maîtrise en Ingénierie Industrielle en Informatique
Maître ès sciences (M.Sc.)

Québec, Canada

Développement d'outils de stockage de données et métadonnées collectées sur un pilote de traitement d'eaux usées

ISIB-ELIN-TFE-04/2015-16

Mémoire

Kevin Fuks

Sous la direction de :

Peter Vanrolleghem, directeur de recherche
Rudi Giot, maître de stage
Elena Torfs, tutrice

Résumé

Le Génie des Eaux et, plus particulièrement, le traitement des eaux usées sont des domaines pour lesquels l'acquisition de données à haute fréquence est indispensable.
Celle-ci, à court terme, permet de contrôler et de modéliser ce qui se passe tout au long du processus.

La station pil*EAU*te est une station miniature de traitement des eaux usées contenant un décanteur primaire, deux réacteurs biologiques en parallèle (formés chacun de 5 bassins : 2 anoxiques et 3 aérobiques) et un décanteur secondaire pour chacun de ces deux derniers. Les informations récoltées en temps-réel par cette station sont enregistrées à différ entes fréquences et sous différentes formes en fonction des capteurs utilisés.

Cependant, si l'on veut pouvoir utiliser ces données sur du long terme tout en conservant les informations capitales à la bonne compréhension de celles-ci (équipement utilisé, but de la mesure, date et heure de celle-ci, etc.), il est important d'avoir un outil performant permettant le stockage de ces données et métadonnées.

La dat*EAU*base est une base de données (BDD) formalisée créée par Q. Plana afin de pouvoir conserver toutes les données et leurs métadonnées. Elle est mise en place dans le but de récolter, périodiquement, les informations des autres bases de données, sous une forme formatée, afin de pouvoir exploiter celles-ci de manière pertinente et aisée.
Les trois types de BDD exportées vers la dat*EAU*base sont :

— Une base de données MSSQL ;
— Une base de données enregistrée dans un fichier .par (lisible par un traitement de texte) ; et
— Une base de données enregistrée dans un fichier .tsdb (lisible avec un lecteur binaire).

Le développement des outils d'import et d'automatisation sont développés dans ce mémoire.

Abstract

Water engineering and, in particular, waste water treatment are branches for which high frequency data acquisition is essential.
This one, in the short-term, allows to control and model what happens during the whole process.

The pil*EAU*te station is a miniature waste water treatment plant including a primary clarifier, two biological reactors (each formed by 5 tanks : 2 anoxics and 3 aerobics) and one secundary clarifier for each of those reactors. Information collected in real time by that station is saved with differents frequencies and/or formats depends on the captors.

Nevertheless, if we want to use this data over the long-term, conserving the capital information for a full understanding of the measurements (equipment used, purpose of the measurement, date and time, etc.), it is important to create an efficient tool allowing the storage of this data and metadata.

The dat*EAU*base is a formalized database (DB) made by Q. Plana in order to retain all the data and metadata. It is setted up with the aim of periodically getting formatted information from other DB to use this pertinently and easily.
The three types of databases we use are :

— A MSSQL database ;
— A database saved in a .par file (readible with a notepad) ; and
— A database saved in a .tsdb file (readible with a Binary Reader.

This thesis will focus on the development and the automatisation of a such database.

Remerciements

J'adresse mes remerciements à toute personne m'ayant aidé de près ou de loin à la réalisation de ce mémoire.
En premier lieu, je remercie le Pr. Peter Vanrolleghem pour son accueil, son enthousiasme ainsi que sa disponibilité pour donner les meilleures réponses et explications à toutes sortes de questions, que celles-ci soient par rapport au stage ou à des domaines plus vastes comme le traitement des eaux usées ou l'organisation interne dans l'équipe.

Je remercie aussi M. Rudi Giot pour ses conseils quant à l'organisation de mon stage, la rédaction du mémoire, la gestion de mon temps ainsi que son soutien dans ce voyage à l'autre bout du monde.

Je n'oublie bien sûr pas M. Jacques Tichon sans qui l'opportunité de partir si loin n'aurait pas été possible.

J'adresse ma reconnaissance aux différentes personnes de l'équipe modelEAU. Elena Torfs pour avoir été ma tutrice, m'avoir laissé agir librement tout en me donnant son avis sur mon travail ainsi que mon intégration dans l'équipe. Queralt Plana pour la création de la datEAUbase sans quoi je n'aurais pas eu matière à travailler. Maxine Dandois-Fafard pour ses appels fréquents afin de me confier différentes tâches et m'occuper lors du début de mon stage. Sylvie Leduc pour sa disponibilité afin d'arranger et accélérer la résolution des différents problèmes administratifs tout le long de ma présence à l'Université Laval. Sey-Hana Saing grâce à qui j'ai appris à me servir des logiciels présents dans le laboratoire du pilEAUte. Et enfin, tous les autres, de l'équipe modelEAU ou d'ailleurs, comme Bernard Patry ou Fabrice Béline, avec qui j'ai pu travailler ou discuter, que cela soit autour d'un café à 10h30 tapantes ou lors de réunions et meetings.

Je tiens finalement à remercier ma famille et mes amis qui m'ont soutenu lors de ce projet, qui ont gardé contact avec moi malgré les 6 heures de décalage horaire et l'emploi d'un vocabulaire étranger composé de "cellulaire", "char", "tabarnak" ou "tuque".

Table des matières

Liste des tableaux

Liste des figures

Abréviations

AMS Automated Monitoring Station

BDD Base De Données

ICMP Internet Control Message Protocol

ID Identifiant

IP Internet Protocol

MeS Matières en Suspension

modelEAU (Groupe de recherche en) modélisation de l'eau

monEAU Monitoring of Water (EAU en français)

MSSQL Microsoft SQL

SCADA Supervisory Control And Data Aquisition

SQL Structured Query Language

SSIS SQL Server Business Intelligence Development Studio

SSMA SQL Server Migration Assistant

TCP Transmission Control Protocol

UDP User Datagram Protocol

UTC Coordinated Universal Time

VB Visual Basic

VPN Virtual Private Network

XML Extensible Markup Language

A Jennifer

La connaissance s'acquiert par
l'expérience,
tout le reste n'est que de
l'information

Albert Einstein

Introduction

Présentation de la société

L'Université Laval

C'est en 1663 que Mgr François de Montmorency-Laval, premier évêque de la colonie, fonde le Séminaire de Québec, premier établissement d'enseignement de la Nouvelle-France. En 1952, l'Université Laval, unique université francophone d'Amérique à cette époque est créée de cet établissement.

Aujourd'hui, il s'agit de 17 facultés, plus de 60 départements, 500 programmes d'étude, plus de 60 000 étudiants et une coopération avec plus de 70 pays. Pour gérer tout cela, pas moins de 9700 employés sont actifs dans cette école.

Le génie des eaux

Parmi les 17 facultés, la Faculté de Sciences et de Génie regroupe différents départements dont celui de Génie Civil et de Génie des Eaux. Le génie des eaux a vu le jour suite à la préoccupation grandissante envers la santé et l'environnement ainsi qu'à l'émergence de nouvelles sciences et de nouveaux domaines apparus avec l'engouement du développement technologique. Il a pour objectif de protéger la sécurité, le bien-être et la santé du citoyen tout en prenant soin de ne pas dégrader l'environnement. Il regroupe différentes autres branches du génie telles que le génie chimique, le génie civil, le génie géologique et le génie rural.

Voici certains objectifs concrets du génie des eaux : le traitement des eaux usées (sur lequel on se concentrera par la suite), la mise à neuf d'infrastructures telles que les aqueducs et les égouts, la prévision des conséquences du réchauffement climatique ou de l'augmentation de la vulnérabilité suite aux inondations et sécheresses.

model*EAU*

Peter Vanrolleghem a fondé model*EAU* en 2006 afin de rassembler des étudiants gradués, des postdoctorants et professionnels de recherche autour d'un sujet commun : la modélisation de la qualité de l'eau. Afin d'améliorer les techniques de modélisation et d'en tirer des conclusions visant à optimiser des systèmes d'eau grâce à différents logiciels, model*EAU* se concentre sur les aspects méthodologiques dans ces domaines. Parmi les projets en cours, mon*EAU* nécessite l'utilisation de bases de données (BDD) complexes et autonomes.

mon*EAU*

Problématique

Les stations automatiques de contrôle de la qualité de l'eau (Automated Monitoring Station : AMS) sont capitales pour la mesure à haute fréquence en qualité de l'eau. Celles-ci rencontrent cependant beaucoup de problèmes sur le terrain et cela, qu'il s'agisse de mesures directement dans le liquide (in-situ) ou dans une boucle en parallèle (on-line). Ces problèmes peuvent être de plusieurs types comme des mesures erronées, l'incapacité de récolter des informations ou de les transmettre, voire la perte totale de certaines données. Les principales causes de ces désagréments sont listées ci-dessous [Rieger et Vanrolleghem, 2008] :

- Un budget limité obligeant de se concentrer sur les capteurs OU la station ;
- Un manque de connaissances en base de données, automatisation, etc. ;
- Des conditions difficiles (interférences électriques, déchets, changements du niveau de l'eau, etc.) ;
- Manque de projets open source ;
- Matériel inadéquat ou défectueux ;
- Design rigide et refus de remplacer les capteurs. ;

Solutions

Afin de lutter contre ces problèmes, Rieger et Vanrolleghem ont réfléchi afin de développer la station mon*EAU*. Celle-ci devait remplir différents critères logiques afin de satisfaire aux besoins. Ainsi, cette AMS est flexible (utilisable à différents endroits, avec des capteurs différents, dans différents objectifs avec des méthodes différentes) ; modulaire et open source (facilité d'ajouter des mises à jour, d'adapter le système en fonction des besoins) ; autonome (nécessite une faible demande énergétique, une maintenance peu régulière, un contrôle à distance, etc.) ; conviviale et simple à l'emploi ; possède une base de données de haute qualité et performance ; etc. [Rieger et Vanrolleghem, 2008]

Bases de données en qualité de l'eau

Les AMS telles que mon*EAU* génèrent énormément de données, et ce, proportionnellement au nombre de capteurs et à la fréquence d'échantillonnage. Sans une bonne BDD, ces informations seraient perdues à court ou long terme. Il est donc indispensable, afin de pouvoir les interpréter et les évaluer, d'avoir un bon système de stockage et d'analyse de données [WSDOT, 2016]. Celui-ci doit contenir toutes les données importantes et pertinentes ainsi qu'être facile d'utilisation et simplifier la représentation future des données [Plana, 2013].
Pour relever tous ces défis, une base de données se doit de respecter les critères suivants : [Camhy et al., 2012, Holmes and Poole, 1996]

- Possibilité de varier les formats des données collectées ;
- Prise en compte de la croissance continue de la BDD ainsi que des modifications du programme de contrôle ;
- Possibilité de changement de la procédure de stockage au cours du temps en fonction des besoins et du matériel ;
- Formalisation des données afin qu'elles persistent dans le temps (en cas de changement d'équipe p.ex.) ;
- Documentation complète et archivage des données et de leurs méta-données (voir section suivante).

L'importance des méta-données

Comme cité dans la section précédente, l'archivage des méta-données est un défi important dans la collecte d'informations liées à la qualité de l'eau. En effet, une valeur peut sembler claire au moment où elle est récoltée mais, à long terme, on ignore totalement les informations essentielles telles que :

- Qui a pris la donnée ?
- Où celle-ci a-t-elle été récoltée ?
- Dans quel but cela a-t-il été fait ?
- Comment a-t-on procédé, automatiquement ou manuellement ?
- Quand a-t-on mesuré cette valeur ?
- Grâce à quel capteur ou quel test en laboratoire ?

Les réponses à ces questions, ces "données à propos des données" sont les méta-données [Hydrolab, 2016]. Sans elles, non seulement il nous manque de l'information mais il devient difficile de comparer des valeurs entre elles, de juger la qualité de celles-ci (grâce à l'équipement utilisé, on peut savoir la précision p.ex.) ou la compatibilité entre différentes BDD.

Mise en contexte

Le contrôle et la modélisation du traitement des eaux usées prend de plus en plus d'importance dans la recherche afin de pouvoir prévenir les risques sanitaires et écologiques. Dans cette optique, les utilisateurs ainsi que les autorités concernées montrent un intérêt grandissant pour les systèmes de contrôle in-situ [Vanrolleghem et al., 2014]. Une grande fréquence d'échantillonnage permet d'atteindre différents objectifs tels que la modélisation, le contrôle intégré ou la détection de dysfonctionnements [Langeveld et al., 2013, Winkler et al., 2002].

Dans ce cadre, la station pil*EAU*te récolte des informations toutes les secondes (pour un suivi court terme efficace) et sauvegarde une valeur pour différents paramètres à l'aide de différents capteurs et programmes toutes les minutes (pour un suivi long-terme). Ces valeurs sont stockées dans trois BDD différentes pour lesquelles le format et le type varient. En outre, certaines valeurs sont prises en laboratoire et sont notées dans des carnets de papier ou fichiers Excel, formant ainsi une "quatrième" base de données non formalisée. La manipulation de ces données n'est actuellement pas aisée et demande énormément de temps pour certaines opérations pourtant simples (comme le téléchargement de plusieurs variables à la fois).

Afin de pouvoir exploiter ces données de manière optimale, il faudrait être capable de les comparer facilement entre elles ainsi qu'au cours du temps. De plus, les méta-données devraient être accessibles afin de connaître, même à long terme, les informations utiles à la bonne compréhension des résultats.

La solution proposée a été de réaliser une base de données générale (la dat*EAU*base) regroupant les entrées des 3 autres BDD, les mesures en laboratoire ainsi que les différents renseignements tels que l'équipement utilisé, la date de la mesure, le projet pour lequel la valeur a été prise ou le paramètre lié à celle-ci.
En outre, les données devraient pouvoir être importées manuellement à n'importe quel moment ainsi qu'automatiquement toutes les heures.
Enfin, une interface simple d'utilisation doit être proposée afin d'ajouter des valeurs de laboratoire, de représenter graphiquement les données de la dat*EAU*base ainsi que d'y ajouter des commentaires lorsque c'est nécessaire.

Résumé des objectifs

- Mise en place de la dat*EAU*base ;
- Import manuel et automatique dans la dat*EAU*base ;
- Création d'une interface permettant d'exploiter la dat*EAU*base.

Première partie

Etat initial

Chapitre 1

Station pil*EAU*te

1.1 Infrastructure

La station pil*EAU*te (miniature d'une vraie station d'épuration) est basée sur les principes cités au chapitre précédent et intègre un système mon*EAU*. Elle a été conçue par Veolia en 2014 et démarrée en janvier 2015 à l'Université Laval. Elle traite les eaux usées des logements estudiantins situés sur le site. La figure 1.1 montre une vue schématisée de l'installation complète.
On y retrouve une station de pompage ; un bassin tampon permettant d'avoir de l'eau usée à tout moment, même aux heures creuses ; un décanteur primaire ; et deux réacteurs biologiques similaires en parallèle se terminant chacun par un décanteur secondaire.

La figure 1.2 schématise le trajet de l'eau depuis l'affluent jusque l'effluent, que celle-ci passe par le premier réacteur biologique (que nous nommerons le pilote) ou par le second (le copilote). On remarque ainsi que les boues activées décantées (mélange de matières organiques) et les liqueurs mixtes (mélange de boues activées et d'eau usée) peuvent être recyclées respectivement par les deux boucles ou évacuées au niveau de chaque décanteur.

1.2 Procédé général

1.2.1 Bassin tampon et décanteur primaire

La figure 1.3 montre le bassin tampon et le décanteur primaire avec des valeurs d'exemple (valeurs usuelles lors de la rédaction de ce mémoire).
L'eau usée arrive depuis l'affluent (via une station de pompage) au bassin tampon. Ce bassin est réglé via des consignes inférieures et supérieures reliées à un capteur de niveau (indiquant 2.01m sur la figure). Ainsi, lorsque le niveau descend en dessous d'une certaine valeur (1.90m pour cet exemple), le pompage commence pour atteindre la consigne supérieure (2.10m dans notre cas) et s'arrêter. Une surverse à l'égout (visible à toutes les étapes donc nous ne reviendrons

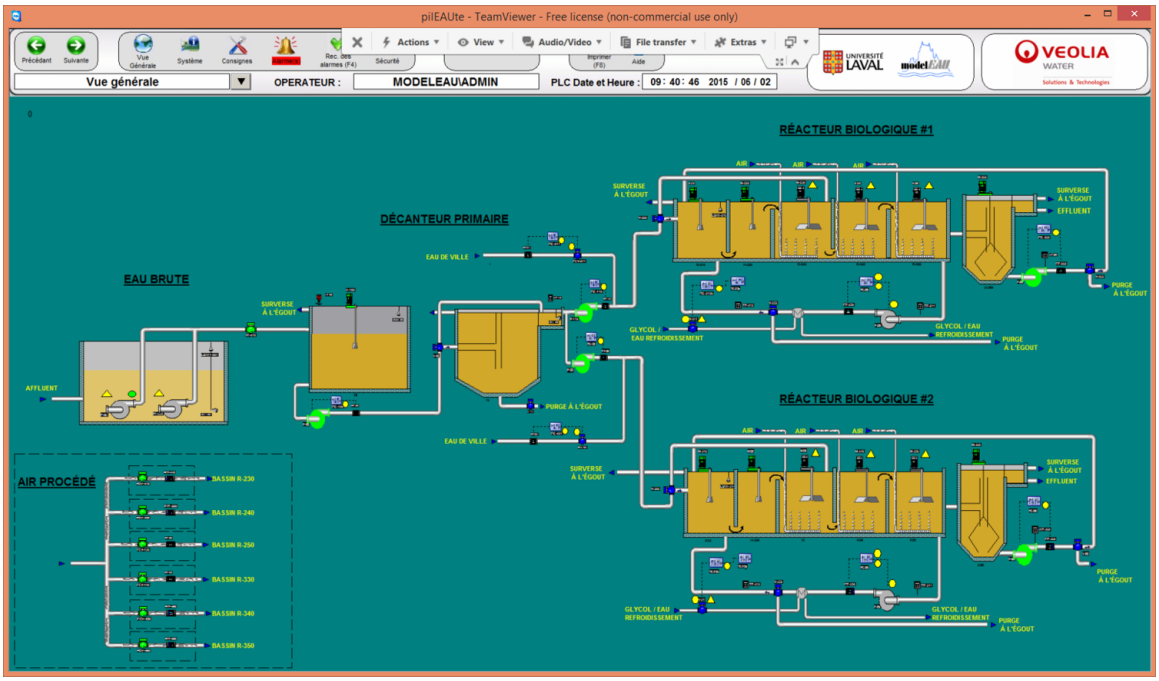

FIGURE 1.1 – Vue de la station pil*EAU*te dans le SCADA [Vanrolleghem, 2015]

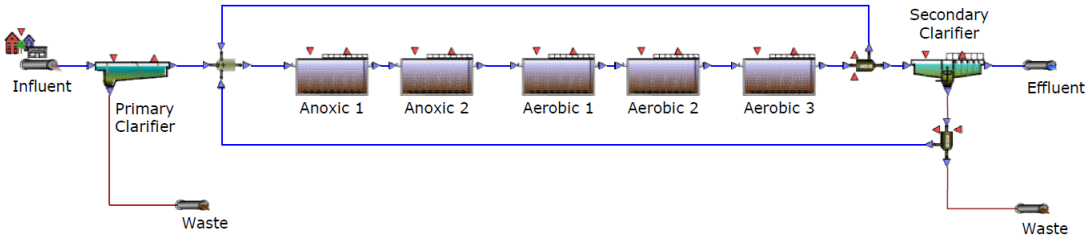

FIGURE 1.2 – Schématisation du trajet de l'eau dans le pil*EAU*te

pas dessus) est quand même prévue en cas de dysfonctionnement. Ce bassin est connecté au décanteur primaire grâce a une pompe et permet d'avoir des eaux usées disponibles quelle que soit l'utilisation ponctuelle en eau. Il est cependant possible, grâce à une vanne, de faire circuler directement les eaux usées depuis le bassin tampon jusqu'aux réacteurs biologiques, sans passer par la décantation primaire, comme le montre le schéma.

Le décanteur primaire sert, grâce à la poussée d'Archimède et à la pesanteur, à laisser retomber les matières en suspension (MeS), plus lourdes que l'eau, et à les évacuer via la purge [EMSE]. Le résultat obtenu, mélangé ou non à de l'eau de pluie, est envoyé vers le pilote et le copilote à raison de $0.5 m^3/h$ chacun (au moment de la rédaction).

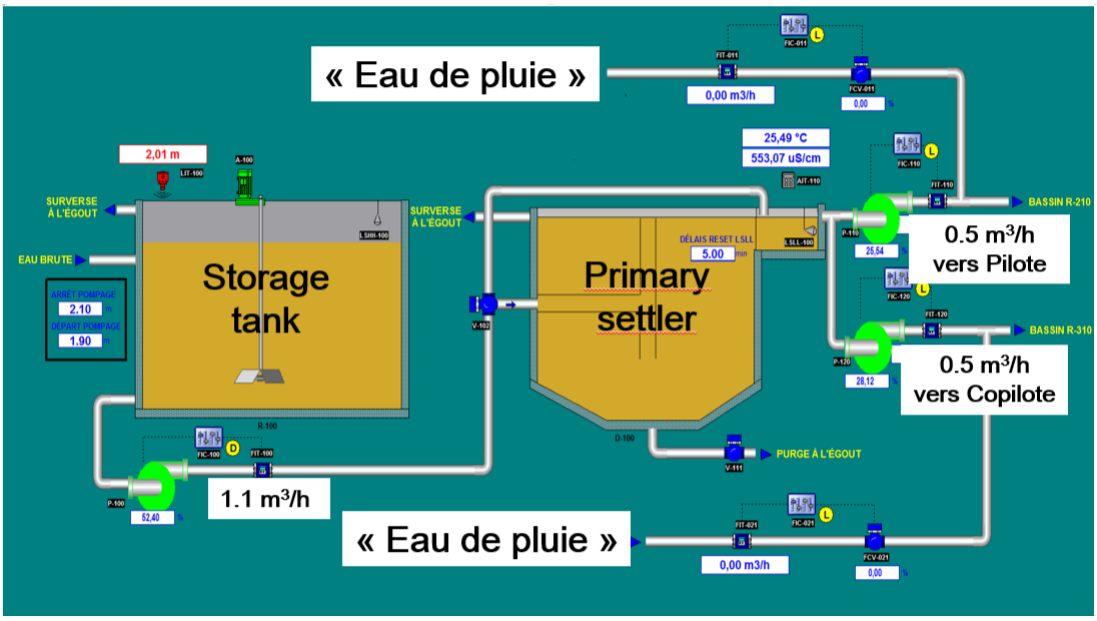

FIGURE 1.3 – Schéma du bassin tampon et décanteur primaire dans le SCADA [Vanrolleghem, 2015]

1.2.2 Les réacteurs biologiques et décanteurs secondaires

La figure 1.4 représente le copilote (semblable au pilote) composé d'un réacteur biologique divisé en 5 bassins (2 anoxiques et 3 aérobiques) et d'un deuxième décanteur. Les eaux usées arrivent depuis la sortie du décanteur primaire jusqu'au premier bassin anoxique (aucun oxygène, mais présence de nitrates) grâce à une pompe. Une vanne permet de passer les deux bassins de ce type pour diriger directement l'eau usée vers le premier bassin aérobique (présence élevée d'oxygène).
Des boues du décanteur secondaire sont retournées en amont des bassins afin de former la liqueur mixte. Après être passé par tous les bassins, le mélange se dirige vers le décanteur secondaire pour ensuite mener l'eau usée à l'effluent. En plus de tous ces bassins, deux boucles sont présentes. Ainsi, une partie du mélange, après être passée par tous les bassins biologiques, repart vers l'entrée du réacteur biologique (recyclage interne avec échange de chaleur et purge "Ekama") tandis qu'une autre boucle sert au recyclage des boues et part de la sortie du décanteur secondaire vers l'entrée du réacteur biologique. Le surplus de boues activées part vers la purge des boues depuis cette dernière boucle.

Les bassins du réacteur biologique servent à dégrader différents composés dans des conditions différentes. Ainsi, en passant dans les réacteurs anoxiques, les nitrates vont réagir avec l'ion hydrogène et le DCO selon la réaction $2\,NO_3{}^- + 2\,H^+ \longrightarrow N_2 + \frac{5}{2}\,O_2 + H_2O$ pour former de l'eau, de l'oxygène et de l'azote. Une dénitrification a donc lieu dans ces bassins. Les matières organiques (bactéries) vont aussi utiliser les nitrates pour former du CO_2 et de l'énergie afin de se multiplier (bactéries hétérotrophes) selon la réaction $2\,NO_3 + 2\,H^+ +$

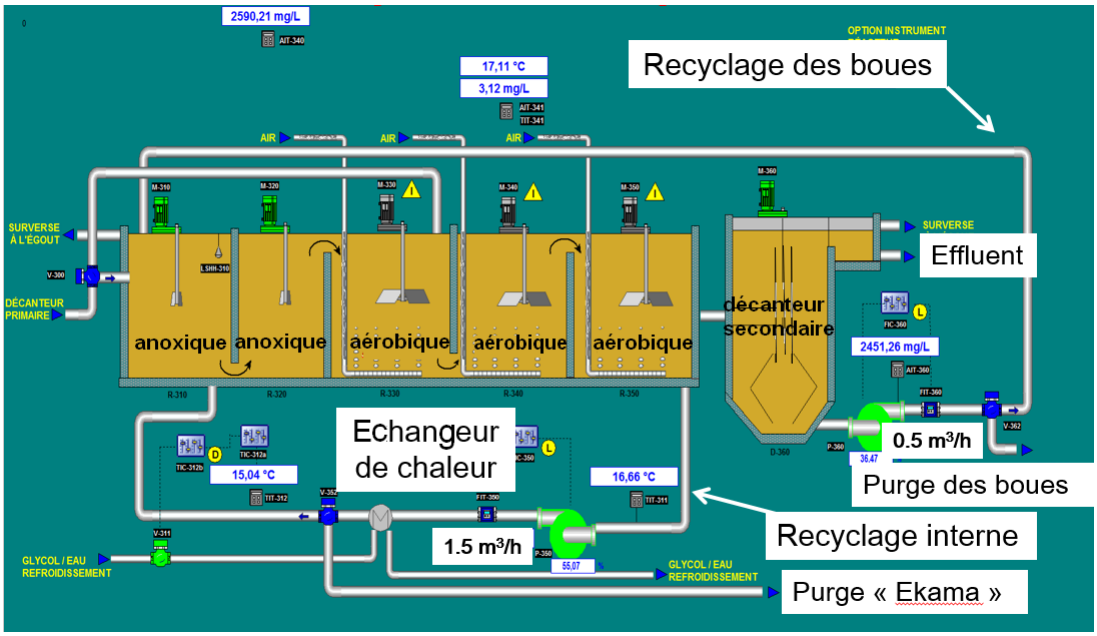

FIGURE 1.4 – Schéma d'un réacteur biologique et décanteur secondaire dans le SCADA [Vanrolleghem, 2015]

DCO (matières organiques) $\longrightarrow N_2 + CO_2 + H_2O$. Les conditions anoxiques sont importantes car ces hétérotrophes préfèrent l'oxygène au nitrate puisque la réaction avec du O_2 produit plus d'énergie qu'avec du ${NO_3}^-$. Dans les bassins aérobiques, l'O_2 va réagir avec l'ammoniac afin de l'oxyder en eau, nitrates et acide grâce à ces deux réactions : ${NH_4}^+ + \frac{3}{2}\,O_2 \longrightarrow 2\,H^+ + 2\,H_2O + {NO_2}^-$ et ${NO_2}^- + \frac{1}{2}\,O_2 \longrightarrow {NO_3}^-$. Une partie des nitrates est renvoyée aux bassins anoxiques grâce à la boucle de recyclage interne afin de les dénitrifier [Aubry, 2003] et une autre est envoyée au décanteur secondaire.

La différence entre les deux décanteurs vient du mélange à décanter. En effet, le premier décanteur s'occupe de particules présentes dans les eaux usées (comme du papier toilette) alors que le second sépare les boues de l'eau traitée qui se rend à l'effluent. Son efficacité est importante au vu de sa position à la sortie du système. Une partie des boues activées décantées est renvoyée vers le premier bassin anoxique tandis que l'autre est rejetée comme purge (les réactions hétérotrophes servant à multiplier les organismes, les boues sont de plus en plus volumineuses).

1.3 Contrôle et représentation graphique

Comme représenté aux figures 1.3 ou 1.4, de nombreux capteurs servent à acquérir de l'information à différents endroits du pil*EAU*te. Ces informations sont disponibles en temps réel sur 3 programmes différents (le SCADA : Supervisory Control And Data Aquisition, ana::pro et BaseStation) en fonction du paramètre mesuré. Par contre, seul le SCADA possède une interface de contrôle.

1.3.1 SCADA

Ce système de contrôle et d'acquisition de données développé par Veolia permet, de manière intuitive, de visualiser les différents bassins du pil*EAU*te comme représenté aux figures 1.1, 1.3 et 1.4. La vision des différentes données et le contrôle des consignes se fait en quelques clics tandis que différents niveaux d'administration permettent une gestion plus ou moins complète des paramètres.
Ces informations sont prises toutes les secondes et l'on enregistre une valeur dans une base de données sous un langage SQL (Server Query Language) et plus particulièrement MSSQL (MicroSoft SQL) toutes les minutes pour la plupart des capteurs. La fréquence de ces ajouts est modifiable en fonction des besoins. Lorsque la BDD atteint une taille de 10Gb, il faut en créer une nouvelle en raison des limitations de la version de SQL Server (Express). A long terme, aucune donnée n'est perdue mais elles se situent dans des bases de données différentes.

1.3.2 ana::pro

ana::pro (figure 1.5) fait partie, avec BaseStation, des deux programmes compris dans la station mon*EAU* implantée dans le pil*EAU*te. Ce programme développé par s::can récolte et enregistre des données toutes les 30 secondes dans un fichier .par. Toutes les 10h20 à peu près, celui-ci atteint une taille de 125 ko et un nouveau fichier est créé pour enregistrer les nouvelles données. Ils sont conservés deux mois puis s'effacent automatiquement, amenant à une perte d'information à long terme.

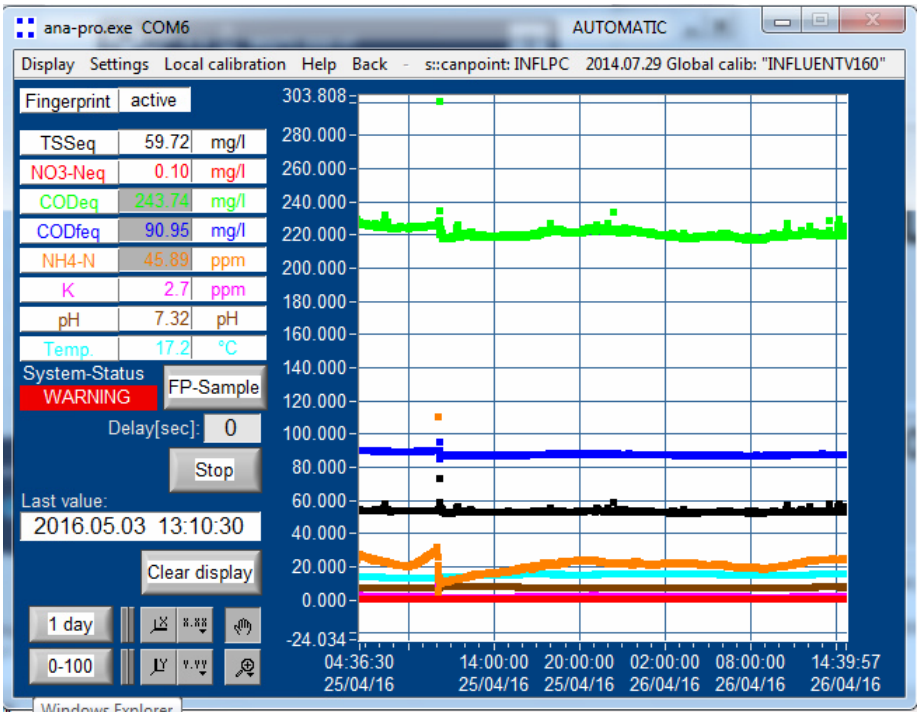

FIGURE 1.5 – Interface d'ana : :pro

1.3.3 BaseStation

BaseStation (figure 1.6) est le deuxième programme de la station monEAU du pilEAUte. Il a été développé par Primodal et acquiert / enregistre les données collectées toutes les 10 secondes dans un fichier .tsdb (time series DataBase). Il y en a un par capteur et par année, et ceux-ci ne sont jamais effacés. Ils atteignent une taille d'environ de 70 Mo lors de leur dernière modification le 31 décembre à 23:59:50 (aux conditions d'enregistrement actuelles).

1.3.4 Représentation des données

Les données se situant dans 3 bases de données différentes exploitant des technologies différentes, l'exploitation et la représentation de données prend du temps. Ainsi, il est nécessaire d'utiliser (manuellement) l'outil de conversion de BaseStation pour transformer les .tsdb en .csv ou .xls. Ensuite, pour les données du SCADA, les manipulations pour extraire les données se font directement dans Excel grâce à une connexion à la BDD tandis que des macros ont été développées pour ana::pro.
Deux scripts matlab sont aussi utilisés. Un premier permet à matlab de récolter les données des stations monEAU tandis que le deuxième les utilise afin de réaliser des graphiques utilisés lors des réunions.
Pour le SCADA, des captures d'écran sont faites et éditées comme représenté à la figure 1.7.

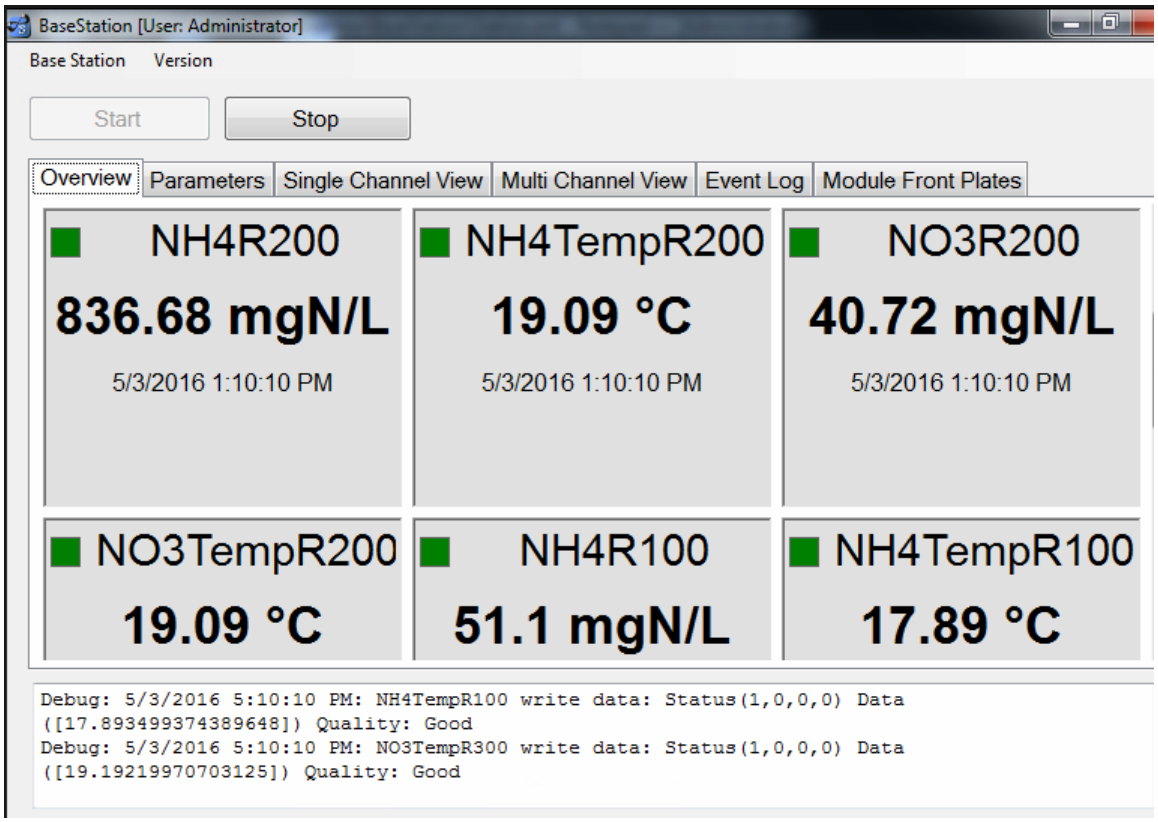

FIGURE 1.6 – Interface de BaseStation

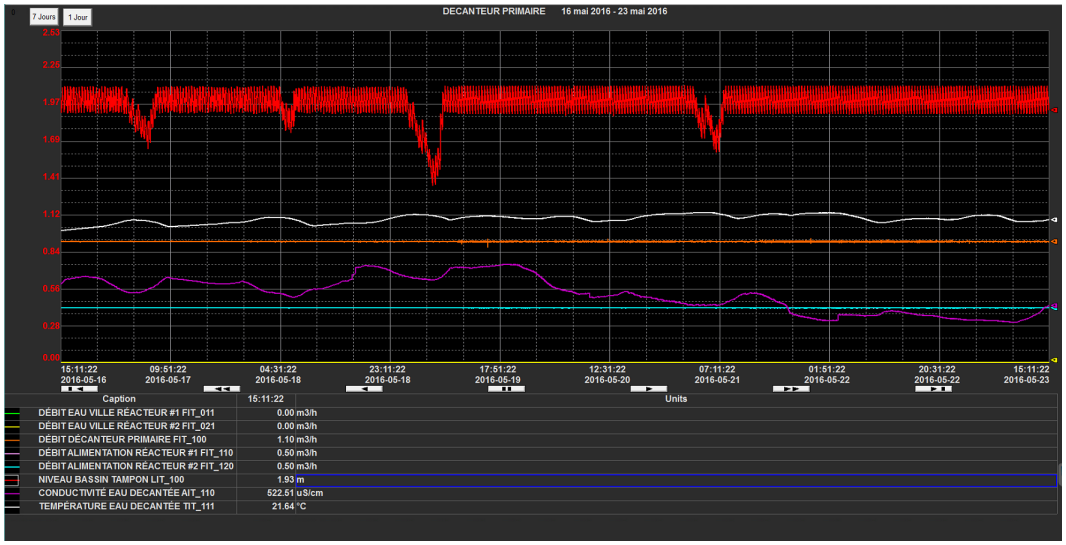

FIGURE 1.7 – Représentation des données du SCADA

1.4 La dat*EAU*base

Les méthodes expliquées à la section précédente sont chronophages et non formalisées. De plus, seules les valeurs ainsi que la date des mesures sont enregistrées, ignorant ainsi toutes les méta-données liées à celles-ci. Enfin, les fichiers utilisés n'ont pas la même durée de vie et sont parfois rapidement effacés. Il est donc nécessaire de créer une unique entité reprenant toutes ces données ainsi que leurs méta-données et de les conserver dans le temps.
Cette base de données, correspondant aux critères pré-cités et facilitant les backups est la dat*EAU*base et a été, initialement, développée en MySQL [Plana, 2013].

1.4.1 Structure

La figure 1.8 représente la structure générale de la dat*EAU*base. Cette structure répond à toutes les questions sur les méta-données que l'on se posait dans l'introduction. On enregistre donc le but, le lieu, une personne de contact, les conditions, le projet concerné, l'équipement utilisé ainsi que les informations sur la valeur (son unité, le paramètre, etc.).

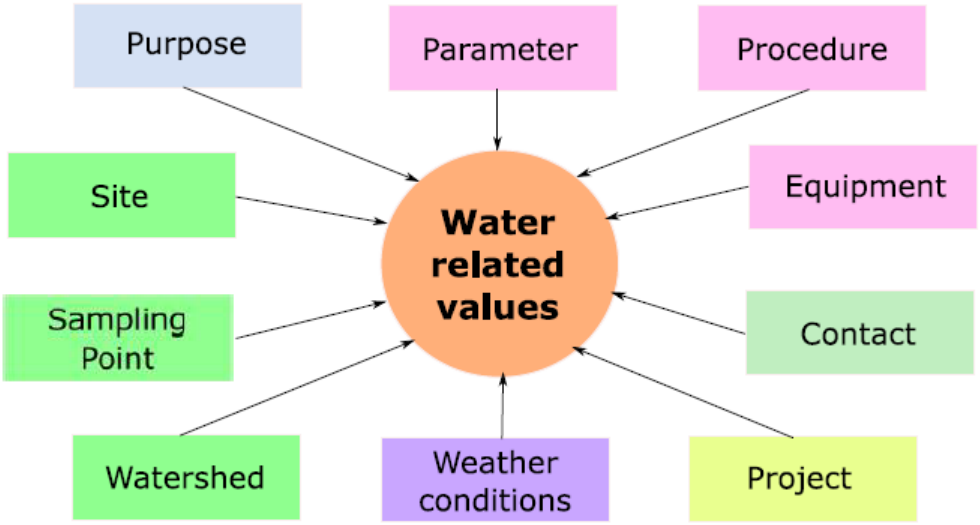

FIGURE 1.8 – Structure de la dat*EAU*base [Plana, 2013]

Cet outil devant être le plus flexible possible et utilisable pour un maximum de projets, toutes les données enregistrées sont formalisées. Ainsi, la date et l'heure seront données sous forme de timestamp Unix (nombre de secondes depuis le premier janvier 1970 à minuit à Greenwich), les champs sont remplis en anglais (langue internationale et évite les problèmes d'accents) et les valeurs sont enregistrées en unités SI.

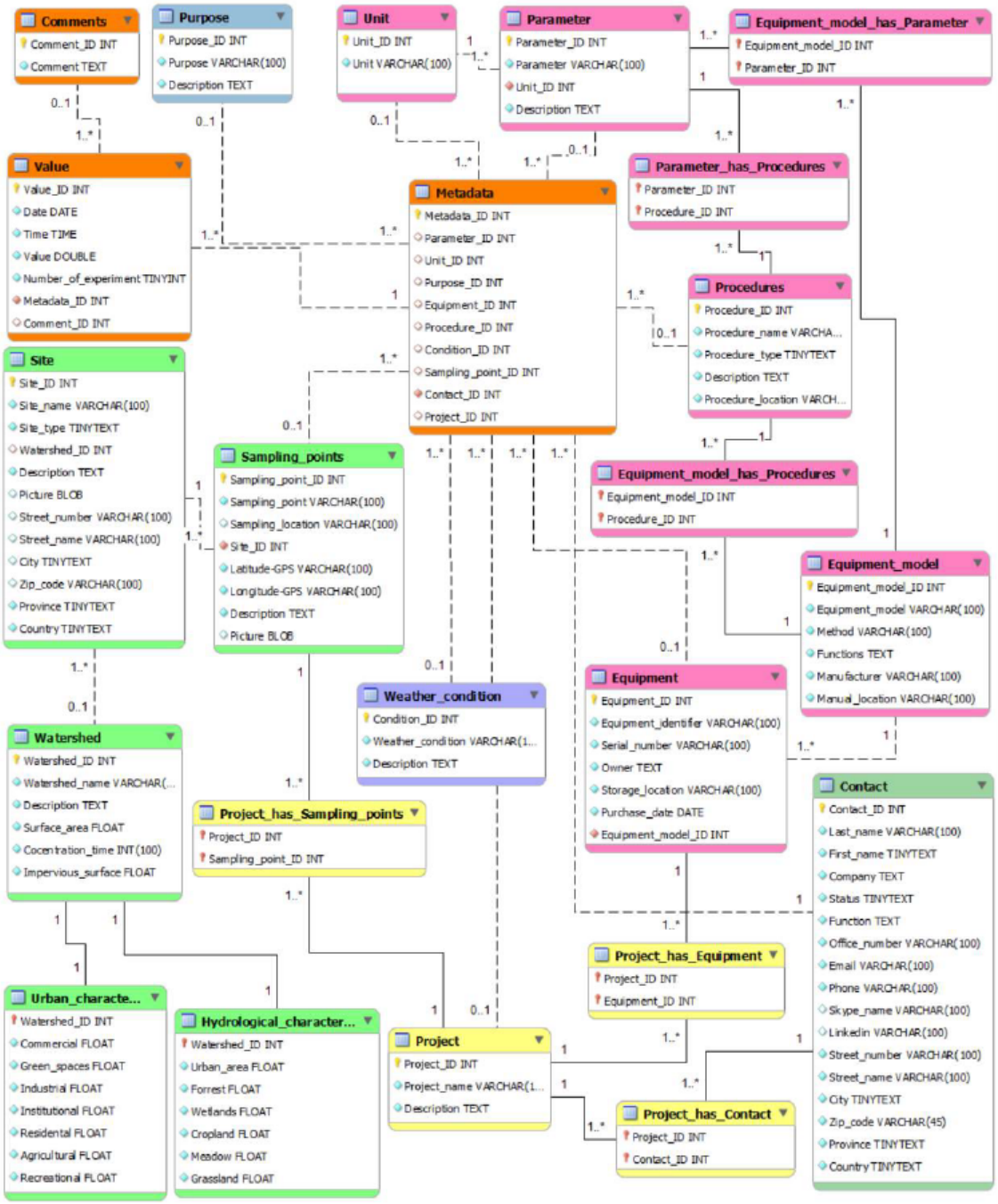

FIGURE 1.9 – Représentation détaillée de la dat*EAU*base [Plana, 2015]

La figure 1.9 est une représentation de toutes les tables de la dat*EAU* base, de leurs attributs ainsi que des liaisons entre elles (voir la sous-section suivante pour les modifications apportées). Les tables principales sont en orange. Pour chaque valeur enregistrée, un commentaire peut être ajouté tandis qu'un identifiant (ID) de métadonnée est obligatoire. Cet ID représente une entrée du tableau Metadata qui contient toutes les informations (au minimum la personne de contact) enregistrées pour cette valeur.
Pour une explication complète et détaillée de cette BDD, voir le mémoire de Plana.

1.4.2 Modifications apportées

La dat*EAU* base, initialement développée en MySQL, a subi quelques modifications. Tout d'abord, pour des raisons de compatibilité et d'import à chaud, celle-ci est maintenant utilisée en MSSQL. Ensuite, pour cause de formalisation, les champs Date et Heure ont été supprimés pour être remplacés par un unique champ Timestamp. Enfin, l'information de l'unité se retrouvait deux fois dans la BDD et était redondante. Le Unit_ID a donc été supprimé de la table Metadata.

1.4.3 Introduction des métadonnées

Une fois la structure finalisée, les différents tableaux de métadonnées ont été remplis manuellement afin d'avoir les informations nécessaires sur les personnes de contact, les différents projets pour lesquels la dat*EAU* base sera utilisée, les capteurs employés, etc..
Les tables étant liées entre elles, un certain ordre doit être respecté. Les tables possédant une clé étrangère (lien entre cette table et une autre) doivent toujours être remplies après la table référencée. Ainsi, par exemple, "Equipment" doit être complété après "Equipment model" ; "Parameter" doit être complété après "Unit" ; etc.. Les champs représentant des clés étrangères sont visibles à la figure 1.9 grâce au logo qui les précède (soit un losange à contour rouge, soit une clé rouge).
La dernière table à devoir être remplie manuellement est "Metadata" tandis que "Value" et "Comments" seront remplies automatiquement grâce aux programmes des chapitres suivants.

Deuxième partie

Solutions et développement

Chapitre 2

Mise en place du serveur et installation logicielle

2.1 Choix du matériel et des logiciels

Le choix du matériel et des logiciels / langages de programmation influence autant le temps à consacrer pour la mise en place de la datEAUbase que celui à consacrer pour la maintenir en place (à minimiser au maximum puisqu'elle doit être autonome) ainsi que ses performances et sa stabilité. Cependant, de manière générale, plus le matériel est puissant et performant, plus il est coûteux. Il faut donc choisir judicieusement les outils que l'on va utiliser afin de répondre aux objectifs sans dépasser le budget alloué.

Différents critères étaient à respecter :

- Terminer l'implémentation de la base de données ainsi que sa mise à jour automatique et son interface (sommaire) avant la fin du stage ;
- Rester raisonnable quant au budget (inconnu) ;
- Avoir des performances suffisantes pour mettre la BDD à jour automatiquement toutes les heures ;
- Sécuriser l'accès aux données et privilégier un serveur se trouvant dans l'Université ;
- Utiliser Windows ; et
- Ne jamais interrompre les prises de mesure et le stockage des données dans les BDD existantes, même pour l'import de données (import à chaud).

Le critère le plus restrictif est le critère spatial. En effet, l'utilisation d'un serveur interne à l'Université nous oblige soit à commander un nouvel ordinateur avec les spécificités requises (mais cela est onéreux et prend beaucoup de temps, donc ne remplit ni le critère pécuniaire ni temporel), soit de louer un serveur virtuel au service informatique de l'Université. C'est cette dernière solution qui a été choisie.

Lors de la commande, il est important de connaître l'espace disque nécessaire, le nombre de cœurs du processeur ainsi que la mémoire et le système d'exploitation souhaités. Chaque paramètre a une influence sur le prix mais peut être modifié au cours du temps. Il a donc été décidé de prendre une configuration minimale pour la phase de tests et d'améliorer certains paramètres si nécessaire (ce qui n'a pas été le cas). Voici la configuration choisie :

— Un processeur avec 1 cœur ;
— 4 Go de RAM ;
— 200 Go d'espace disque et
— Windows Server 2008 R2 (disponible gratuitement).

Afin d'assurer la sécurité et de ne permettre qu'à certaines machines extérieures de se connecter au serveur, un serveur VPN (Virtual Private Network) a été installé.

Pour ce qui est de la programmation, le script d'import de données pour BaseStation a été réalisé en Python. Cela permet d'avoir un langage de programmation gratuit et performant déjà employé et connu par l'équipe de model*EAU*.

Par contre, l'emploi de Microsoft SQL (logiciel payant) a été obligatoire pour le SCADA car sa base de données initiale est aussi en MSSQL. Par conséquent, le seul moyen efficace de faire des imports à chaud était de se servir du même logiciel. La version utilisée est MSSQL 2014 Enterprise Edition. La version Express (gratuite) ne permet que de faire des BDD de 10 Go (comme c'est le cas pour le SCADA) et ne possède que peu d'outils de développement. Cette édition-ci, quant à elle, n'a pas de restriction de taille de base de données, ce qui est important car l'on veut que le serveur puisse tourner le plus longtemps possible sans intervention humaine.
De plus, elle inclut SQL Server Business Intelligence Development Studio (SSIS). Cet outil permet de programmer visuellement des séquences de programmes permettant de nombreuses utilisations diverses, dont l'import à chaud depuis un autre serveur SQL sous MSSQL.
Enfin, l'Université possédant diverses licences Microsoft gratuites, cela ne posait pas de problèmes quant au budget. Ce logiciel a aussi été exploité pour Ana::Pro car Python ne permettait pas de lire correctement les données des fichiers .tsdb (voir chapitre 4).

2.2 Configuration des pare-feux et activation du protocole TCP/IP

Après installation de Windows Server [ToutWindows], il faut le configurer afin qu'il réponde à nos besoins. En effet, si l'on veut que le serveur soit protégé, le pare-feu doit être activé. Celui-ci risque donc de poser problème lorsque l'on veut établir différentes connexions (SQL ou VPN p.ex.). Pour éviter cela, il faut modifier les règles du pare-feu (qui se trouvent dans Outils d'administration => Pare-feu avec fonctions avancées de sécurité). Avec un clic droit sur les règles de trafic entrant, il est possible d'en ajouter plusieurs. Voici celles qui sont nécessaires :

Règles des pare-feux				
	Port	*Entrant / Sortant*	*Ordinateur*	*Protocole*
Ping (debug)	//	Entrant	Tous	ICMPv4
VPN	1194	Entrant	Serveur	UDP
SQL	1433	Entrant	SCADA	TCP
Partage de fichiers	137, 138	Entrant	BaseStation / Ana::Pro	TCP
Partage de fichiers	139, 445	Entrant	BaseStation / Ana::Pro	UDP

TABLE 2.1 – Règles des pare-feux

Seul le ping demande une règle personnalisée (une des options proposées) où l'on peut choisir directement le protocole à accepter.

L'ouverture du port SQL est réalisée mais ne sert à rien si l'ordinateur utilisant le SCADA n'active pas celui-ci.
Il faut donc configurer l'utilisation du port TCP/IP dans le SQL Server Configuration Manager. En sélectionnant le serveur utilisé (figure 2.1) et en allant dans les propriétés du protocole TCP/IP (figure 2.2), il est possible d'activer un port pour une certaine adresse IP. Le port est celui par défaut (celui que l'on ajoute dans les règles du pare-feu) et l'adresse IP est celle que l'on choisit sur l'ordinateur hôte (ici, celle de la connexion VPN afin de sécuriser la liaison).

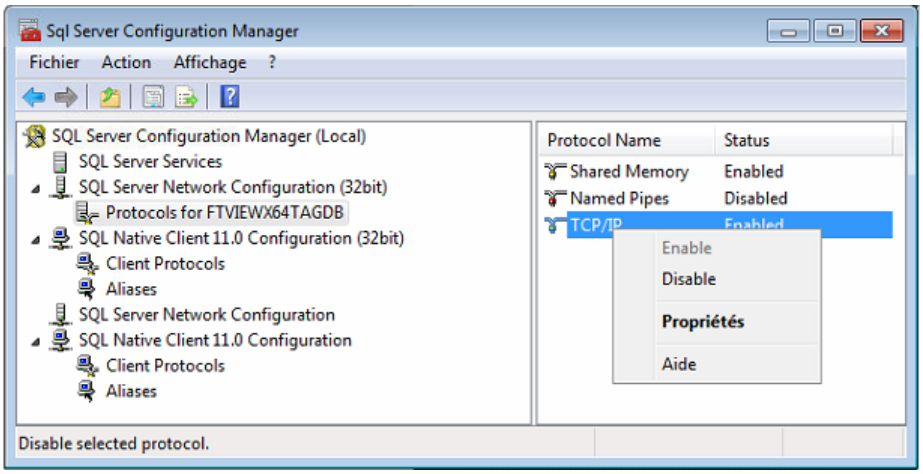

FIGURE 2.1 – Configuration Manager

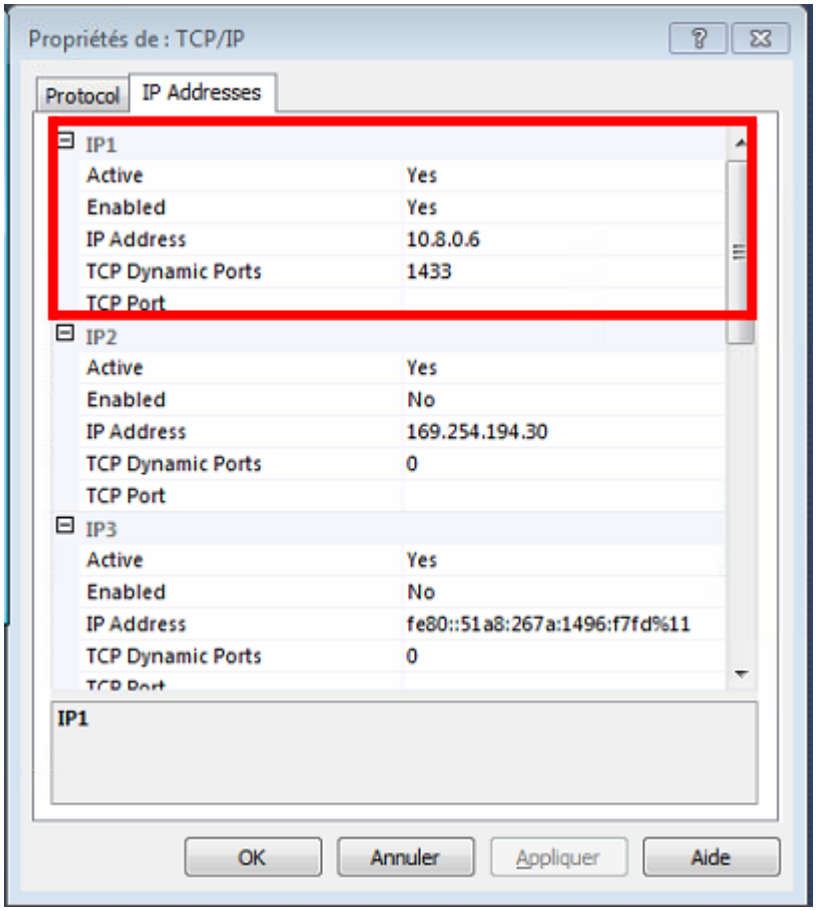

FIGURE 2.2 – Configuration du protocole TCP/IP pour SQL

2.3 Mise en place du tunnel VPN

Un réseau virtuel privé (VPN) est un système qui permet à plusieurs ordinateurs distants de se connecter sur un même réseau local qui leur est propre. On crée un "tunnel" entre le serveur et différents clients par lequel passent les données. Différents protocoles existent tels que PPTP, L2TP, IPsec ou OpenVPN. OpenVPN est un choix judicieux car le chiffrement est plus élevé, le débit est rapide, le système est stable et il est compatible avec quasiment tous les systèmes d'exploitation (même les tablettes).

Les principaux avantages de ce mode de connexion sont :

- Possibilité d'accéder aux ordinateurs distants comme dans un réseau local ;
- Si un tiers intercepte les données, on reste anonyme (l'IP affichée est celle acquise dans le réseau VPN et non l'IP privée) ; et
- Chiffrement des données.

Le chiffrement se fait à l'aide d'une clé publique, d'une clé privée et d'un certificat. La clé privée doit rester secrète, la clé publique peut être partagée avec les personnes qui devront déchiffrer nos messages et le certificat vient d'une autorité de certification (cela peut être la personne configurant le VPN pour une petite entreprise).
Tout message chiffré avec une clé publique ne peut être déchiffré que par le possesseur de la clé privée associée (ceci atteste la confidentialité). Tandis que tout message crypté avec la clé privée peut être décrypté par tout possesseur de la clé publique associée (ceci certifie l'origine du message) [Debian Handbook].
Le certificat sert à identifier l'origine du paquet. Cela sert, par exemple, lorsque l'on se connecte au site internet d'une banque, à être sûr que l'on est pas sur un site pirate.
Ainsi, si Alice veut envoyer un message crypté à Bob (noms généralement employés dans la littérature spécifique), elle chiffrera les données avec la clé publique qu'il lui aura envoyée au préalable. Bob sera le seul à pouvoir le décrypter avec sa clé privée. Tandis que si Alice veut affirmer que c'est elle qui envoie un message, elle le chiffrera avec sa propre clé privée et Bob le déchiffrera avec la clé publique de celle-ci (donnée au préalable) tandis que le certificat assurera l'origine du message.

Les opérations à suivre (dont les lignes de commandes se trouvent en Annexe A) sont donc de créer l'autorité de certification avec son certificat et sa clé privée (qui servira à signer les clés du serveur et des clients). Ensuite, il faut générer les clés pour le serveur et enfin pour chaque client.
Une autre étape est de créer une clé de Diffie-Hellman. Celle-ci est une clé partagée créée à l'aide de la clé publique d'une des deux personnes et de la clé privée de l'autre, permettant d'assurer un message crypté lisible uniquement par les deux entités. Cette clé est employée

par le serveur lors de l'échange de clés de chiffrement au moment de la connexion au serveur. L'algorithme utilisé est trop complexe pour être développé ici mais toutes les informations sont disponibles sur Wikipédia.

2.4 De MySQL à MSSQL

Pour les raisons citées à la section 2.1, il est important d'utiliser MSSQL. Cependant, la base de données dat*EAU*base a été développée par Q. Plana en MySQL. Il est donc important d'avoir un outil permettant d'importer la BDD initiale vers Microsoft SQL. Pour cela, nous utilisons SQL Server Migration Assistant (SSMA). Ce logiciel gratuit est puissant et permet de transférer la structure de la base de données, les liaisons entre les tables mais aussi les éventuelles données déjà présentes dans la BDD.

Plusieurs pré-requis sont nécessaires au bon fonctionnement de l'import :

- Avoir installé et connecté la BDD MySQL (en local ou non) ;
- Avoir installé le connecteur/ODBC (Open Database Connectivity) permettant à MSSQL de se connecter à la base de données MySQL ;
- Avoir installé MSSQL et créé la BDD dans laquelle on veut importer la structure et les éventuelles données ;
- Avoir installé SSMA via le site de Microsoft en sélectionnant SQL Server Migration Assistant for MySQL sur la Microsoft Web Platform Installer ; et
- (Avoir configuré le pare-feu si la BDD MySQL n'est pas en local).

Après installation et lancement de SSMA, vous arriverez sur son interface principale ressemblant à la figure 2.3.
En cliquant sur la première icône de la barre d'outils de projet ("Project Toolbar"), on crée un projet qu'il faudra nommer. Après cela, toujours dans la même barre d'outils, il faut cliquer sur "Connect to MySQL" (qui devient "Reconnect to MySQL" si la connexion a déjà été établie). Cela va ouvrir une fenêtre (figure 2.4) où il faut rentrer les informations sur le serveur SQL (nom du serveur, port de connexion, nom d'utilisateur et mot de passe éventuel). Il faudra aussi sélectionner l'ODBC pour le bon fonctionnement de la connexion. En sélectionnant la BDD dans l'explorateur de métadonnées MySQL (MySQL Metadata Explorer) et en cliquant ensuite sur "Create Report" (dans la "Project Toolbox"), il est possible d'avoir un aperçu des éventuelles erreurs de conversion afin de pouvoir les modifier avant l'import (p.ex. des dates au mauvais format).
Ensuite, on utilise le bouton de connexion à SQL Server se trouvant entre les deux boutons précédemment utilisés et l'on remplit les champs requis (semblables à ceux de MySQL, sauf qu'il ne faut pas choisir de connecteur et qu'on sélectionne la base de données précédemment créée) comme le montre la figure 2.5.

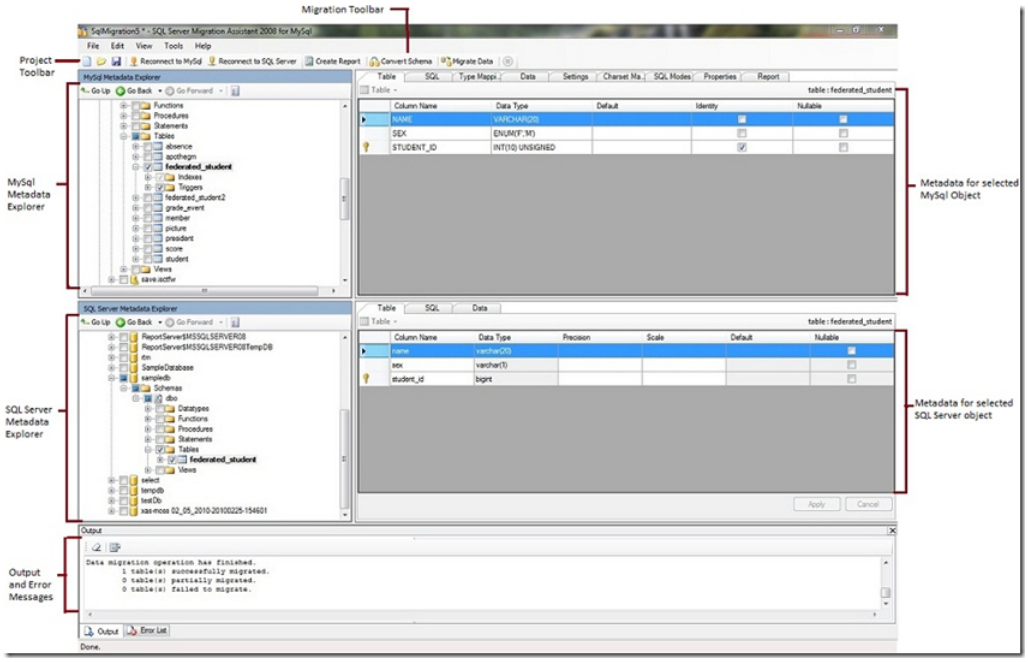

FIGURE 2.3 – Interface de SSMA [Microsoft, 2016]

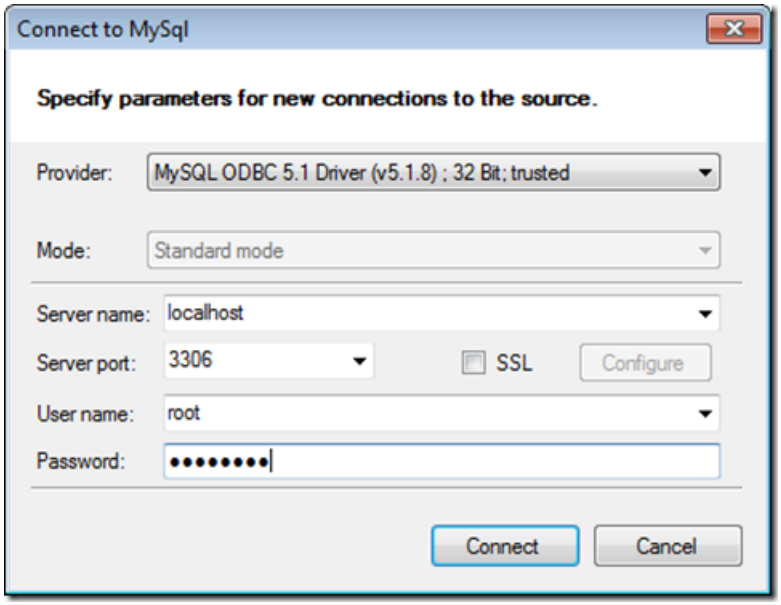

FIGURE 2.4 – Interface de connexion à MySQL [Microsoft, 2016]

Pour convertir la structure, il suffit maintenant de cliquer sur Convert Schema dans la barre d'outils de projet. Puis, pour importer, de faire un clic droit sur la base de données dans l'explorateur de métadonnées SQL Server (SQL Server Metadata Explorer) et de sélectionner la synchronisation avec la BDD (Synchronise with Database). Enfin, le bouton "Migrate Data" gère automatiquement l'import des données présentes.

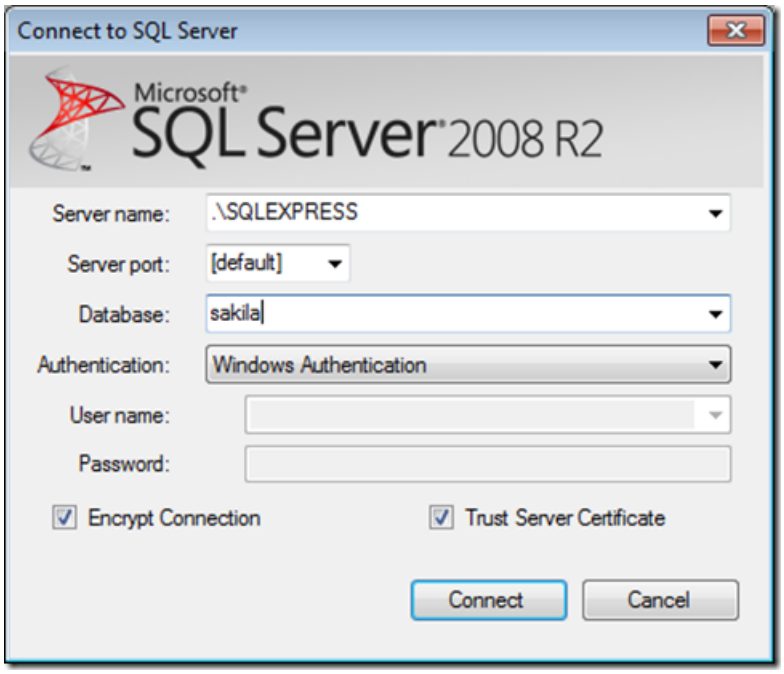

FIGURE 2.5 – Interface de connexion à MSSQL [Microsoft, 2016]

2.5 Python

Pour des raisons expliquées précédemment, nous utilisons Python pour transférer les données depuis les fichiers .par vers la dat*EAU* base. La distribution choisie est Anaconda de Continuum Analytics. La version 2.7 est suffisante pour ce que nous allons faire mais si vous possédez une version 3.5, il est possible de quand même utiliser un environnement 2.7 pour être compatible avec les programmes développés.
La raison de l'utilisation d'Anaconda est que cette distribution possède déjà une grande bibliothèque de paquets et qu'en installer de nouveaux est simple (je vous invite à lire la documentation officielle). Les paquets utilisés sont :

- "os" pour la lecture/écriture des fichiers ;
- "glob" pour faciliter l'utilisation des chemins d'accès aux fichiers ;
- "time" et "datetime" pour la gestion/converstion des dates et heures ainsi que des timestamps ;
- "re" pour gérer les expressions régulières ; et
- "pymssql" pour pouvoir connecter Python à MSSQL.

Chapitre 3

Import automatique depuis MSSQL

Comme expliqué au chapitre précédent, pour parvenir à faire un import à chaud des données se trouvant sur un MSSQL, nous utilisons les outils fournis par Microsoft Server SQL (SSIS). Nous nous focaliserons, ici, uniquement sur les parties du programme qui servent pour l'import depuis le SCADA tandis que les blocs utilisés pour l'import des données de la BaseStation seront développés au chapitre 4. On remarque aux figures 3.1 et 3.2 qu'une partie du code est commun aux deux. Le bloc d'union (Union All) ne sera plus représenté par la suite. Il sert à rassembler les données du SCADA et de BaseStation dans le même flux avant l'incrémentation automatique afin que ces données subissent le même traitement sur la fin du flux de données.

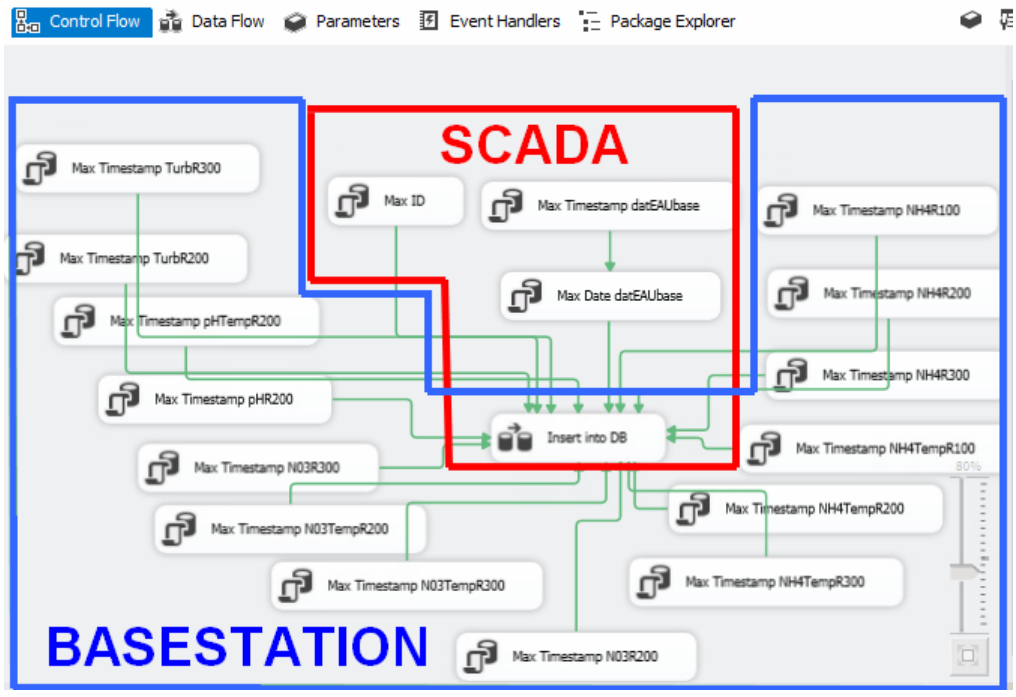

FIGURE 3.1 – Control Flow SSIS

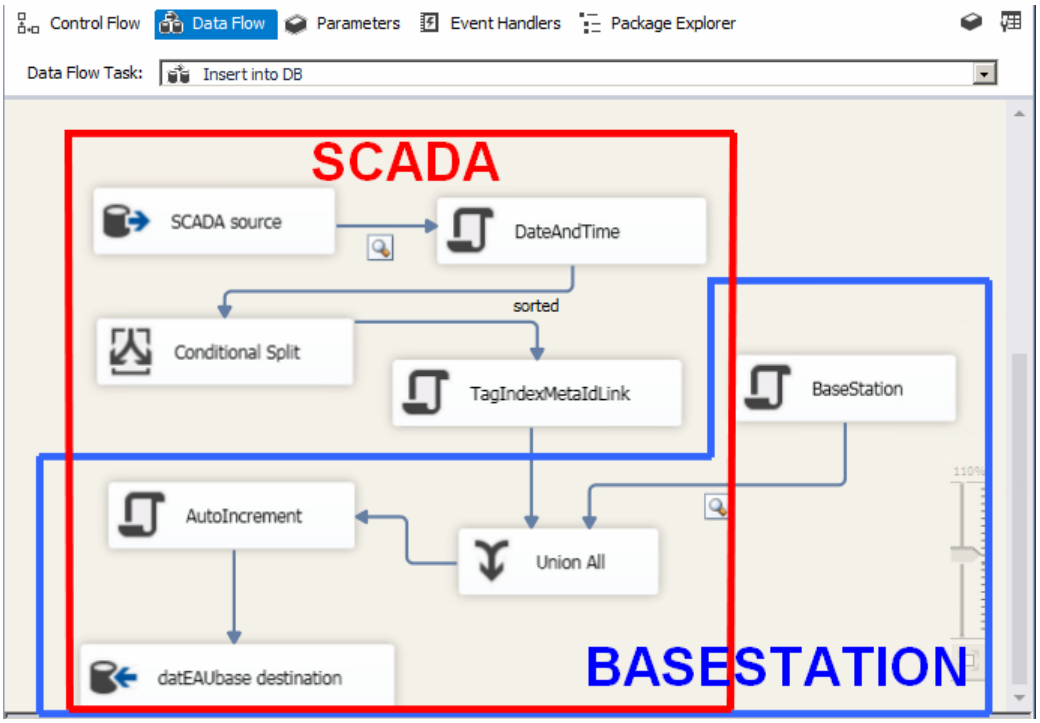

FIGURE 3.2 – Data Flow SSIS

3.1 Avant de programmer

Avant de commencer, il faut configurer certains paramètres afin que le programme puisse fonctionner par lui-même lors de la phase d'automatisation.

3.1.1 Créer des connexions

Tout d'abord, il faut créer deux connexions : une vers la dat*EAU*base et l'autre vers le serveur MSSQL distant. Cette dernière fonctionnera uniquement si vous avez configuré votre pare-feu comme expliqué au chapitre précédent.
Pour ce faire, il suffit de faire un clic droit sur les Connexion Managers (figure 3.3 - cadre 1) et d'en créer un nouveau de type OLEDB.

La figure 3.4 montre la fenêtre sur laquelle il faut arriver. Pour la dat*EAU*base, le serveur se trouvera dans la liste déroulante et le choix d'authentification Windows est préférable tandis que, pour le serveur externe, vous devrez faire une connexion TCP/IP et entrer le compte administrateur ("sa" par défaut) ainsi que son mot de passe.

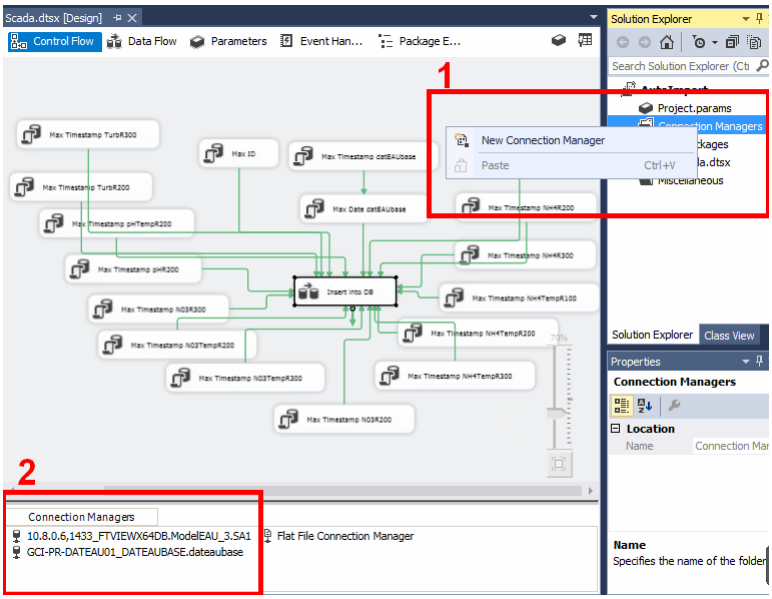

FIGURE 3.3 – Interface du SCADA et Connection Managers

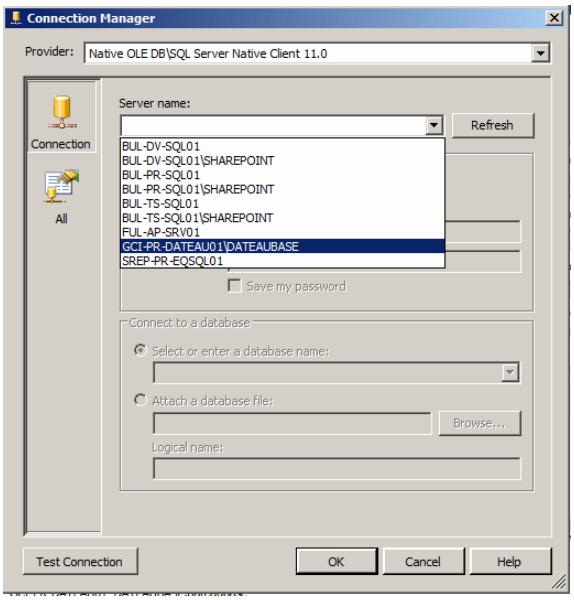

FIGURE 3.4 – Nouveau Connection Manager

Pour une connexion TCP/IP, il faut entrer l'adresse IP du serveur externe, le port de communication ainsi que le nom du serveur sous la forme "xxx.xxx.xxx.xxx(IP),yyyy(port)\NOM_DU_SERVEUR" comme indiqué à la figure 3.5. La liste déroulante des bases de données permet ensuite de choisir la BDD souhaitée sur le serveur. Enfin, une fois cela fait, il faut faire un clic droit sur chacune de nos connexions dans les Connexion Managers (figure 3.3 - cadre 2) et cliquer sur "convert to package connection". Cette étape, liée à la création d'un fichier de configuration (sous-section suivante) est capitale pour l'automatisation.

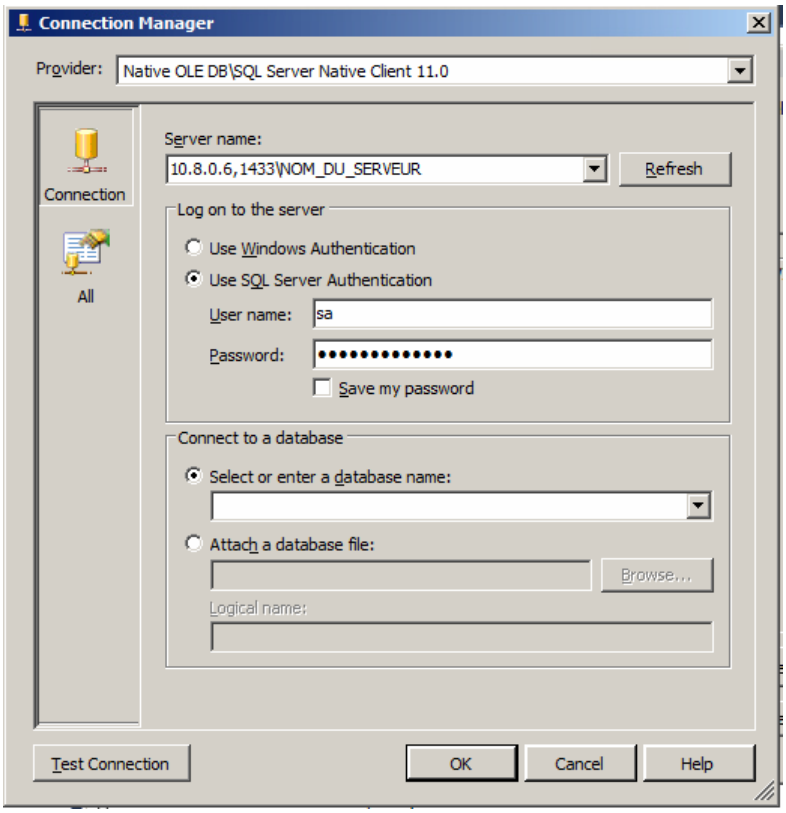

FIGURE 3.5 – Exemple d'ajout de serveur via protocole TCP

3.1.2 Fichier de configuration

Le fichier de configuration est un fichier qui sera lu à chaque fois que le projet s'exécute. Il est important car c'est lui qui contiendra, à chaque connexion, le mot de passe du serveur distant. Son code à mettre dans un fichier XML (Extensible Markup Language) étant court, il ne sera pas mis en annexe. Il faut cependant remplacer les champs entre astérisques par les informations du serveur distant.

```
<DTSConfiguration>
        <Configuration ConfiguredType="Property" Path="\Package
        .Connections[**NOMDEVOTRECONNECTION(le connexion manager)**]
        .Properties[ConnectionString]" ValueType="String">
                  <ConfiguredValue>
                        Data Source=**IP,PORT\SERVEUR**;
                        Initial Catalog=**BASEDEDONNEE**;
                        User ID=SA;password=**MOTDEPASSE**;Provider=SQLOLEDB.1;
                        Persist Security Info=True;Auto Translate=False
                 </ConfiguredValue>
        </Configuration>
</DTSConfiguration>
```

Un fois ce fichier sauvegardé au format XML (grâce à n'importe quel éditeur de texte), il faudra, en cliquant n'importe où dans la page de Control Flow, ajouter cette configuration au projet grâce au champ "Configuration" du menu "Properties" (figure 3.6).

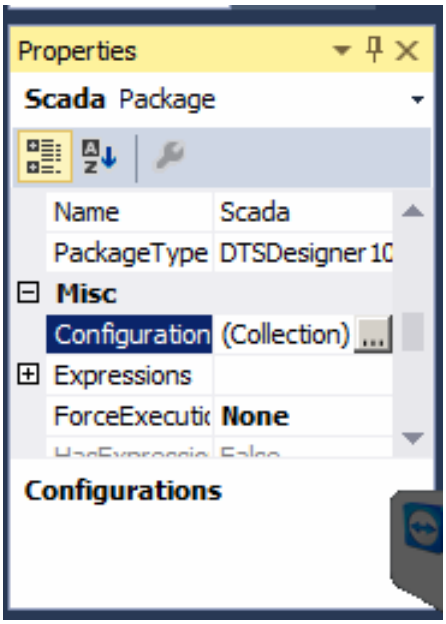

FIGURE 3.6 – Propriétés de projet pour ajout de fichier de configuration

3.2 Programmation

La programmation avec SSIS se fait au moyen de blocs ayant différentes fonctions que l'on sélectionne et place sur notre espace de travail. Ainsi, un bloc sert à la connexion vers la source, un autre permet de séparer les données et d'autres permettent même de créer des fichiers. Il est ensuite aisé de configurer ces blocs selon nos besoins. Les figures 3.7 et 3.8 montrent les blocs utilisés pour la programmation de l'import. Et ce, que cela soit pour l'extraction des données, la conversion des dates, la conversion des valeurs, l'incrémentation automatique ou la référence vers la métadonnée correspondante.

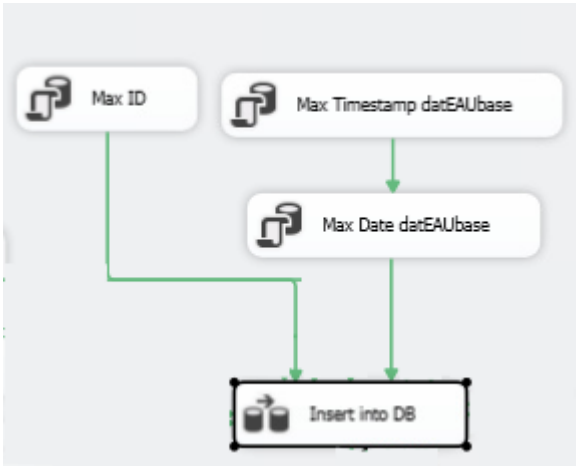

FIGURE 3.7 – Control Flow SSIS pour le SCADA

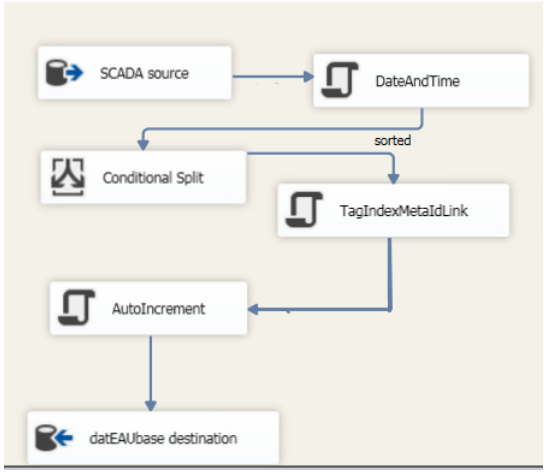

FIGURE 3.8 – Data Flow SSIS pour le SCADA

3.2.1 Control Flow

La figure 3.7 représente le flux de contrôle de notre programme. C'est le corps principal dans lequel les différentes étapes vont se succéder. Parmi ces étapes, se trouve un flux de données qui sera développé à la section suivante.
Les 3 blocs supérieurs représentent des requêtes SQL permettant d'obtenir les valeurs les plus hautes d'ID et de timestamp dans la dat*EAU*base ainsi que la date et l'heure correspondant à ce timestamp. Il faut configurer (dans les options du bloc) la sortie comme une ligne simple (single row) et créer une nouvelle variable qui sera exploitée plus tard. Ces requêtes et le développement de la création de variables sont disponibles dans les annexes B et C.
Le bloc inférieur (Insert into DB) est un flux de données et sert à importer, manipuler et exporter les données.

3.2.2 Data Flow

La figure 3.8 montre le flux de données utilisé pour l'import de la BDD du SCADA. Les éléments se succèdent en suivant les flèches depuis la source (SCADA source) jusqu'à la destination (datEAUbase destination).

Récupération des données

Le premier bloc sert à récupérer les données depuis le serveur distant. En utilisant une source OLE DB connectée à la connexion établie à la section 3.1, on extrait, grâce à du code SQL (disponible en Annexe C), les données du SCADA qui sont plus récentes que la dernière valeur (de cette BDD) enregistrée dans la datEAUbase. Pour gagner en efficacité et en espace mémoire, on extrait seulement les valeurs, leur date, leur heure ainsi que l'index de leur capteur.

Conversion de dates et prise en compte des heures d'été et d'hiver

Le second bloc est un script de transformation. Il sert, grâce à un code C#, à convertir les valeurs de date et d'heure en timestamp UTC (Coordinated Universal Time) après avoir pris en compte l'heure d'été et d'hiver. Le timestamp UTC correspond au nombre de secondes écoulées depuis le 1er Janvier 1970 à Greenwich. L'annexe D reprend le code de ce script.

Séparation conditionnelle, lien avec les métadonnées et conversion des unités

Le bloc de séparation conditionnelle permet de séparer les valeurs en plusieurs canaux afin de les rediriger vers différents flux par la suite. Ici, il sert à séparer les valeurs dont l'index du capteur est égal à 2 et les autres. En effet, cet index de valeur 2 est une erreur dans le SCADA dont la mesure vaut toujours 0. Ainsi, pour ne pas sauvegarder de données inutiles, nous excluons celles-ci. La figure 3.9 montre comment fonctionne ce séparateur.

Order	Output Name	Condition
1	sorted	TagIndex != 2

FIGURE 3.9 – Séparateur conditionnel

On peut remarquer à la figure 3.8 que la sortie du séparateur se nomme "sorted" comme indiqué à la figure 3.9. Il est possible de créer autant de sorties que désiré.
Le bloc suivant est un script C# ayant une double fonctionnalité. Il permet de lier les données à leur métadonnées correspondantes en fonction de l'index du capteur, mais aussi de convertir la valeur de la mesure pour qu'elle corresponde aux unités SI. Ainsi, si un capteur prend une mesure en L/min, celle-ci sera convertie, via un facteur de multiplication de 1/60000 afin de correspondre à des m^3/s.

Une ligne supplémentaire indique que le "Number_of_experiments" (voir figure 1.9) vaut 1 car il s'agit d'une mesure en ligne et non de laboratoire. Le code de ce script ainsi que les variables d'entrée et de sortie se trouvent en Annexe E.

Incrémentation automatique

Le bloc d'incrémentation automatique représente un script C# donc les variables et le code sont développés en Annexe F. Il permet, grâce à la valeur d'ID max récupérée dans le Control Flow, pour chaque donnée à entrer dans la BDD, de créer un nouvel ID ayant la valeur du dernier identifiant augmentée de 1.

Import des données dans la dat*EAU*base

Le dernier bloc représente une destination OLEDB. Dans son onglet de connexion, il suffit d'entrer le connexion manager déjà créé et la table dans laquelle on veut importer les données ("Value" dans ce cas-ci). Ensuite, dans l'onglet "Mapping", on lie les variables acquises dans SSIS aux colonnes de la table sélectionnée afin que le programme sache comment importer les données (figure 3.10).

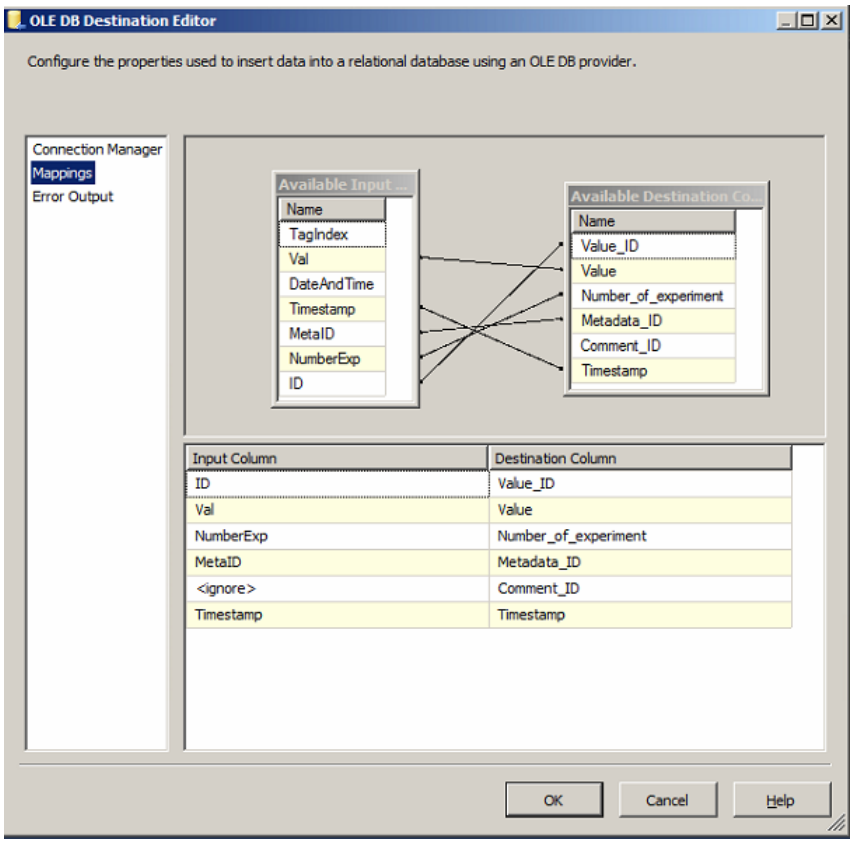

FIGURE 3.10 – Mapping des données vers la destination

3.3 Automatisation

3.3.1 Démarrage de l'Agent SQL Server

L'automatisation se fait grâce à un "Agent SQL Server". Ce service doit être activé dans le "SQL Configuration Manager" (programme déjà utilisé pour activer les connexions TCP/IP à la section 3.1). Si le service n'est pas démarré (running), un clic droit suivi de l'option de démarrage (Start) permet de corriger cela (voir figure 3.11).

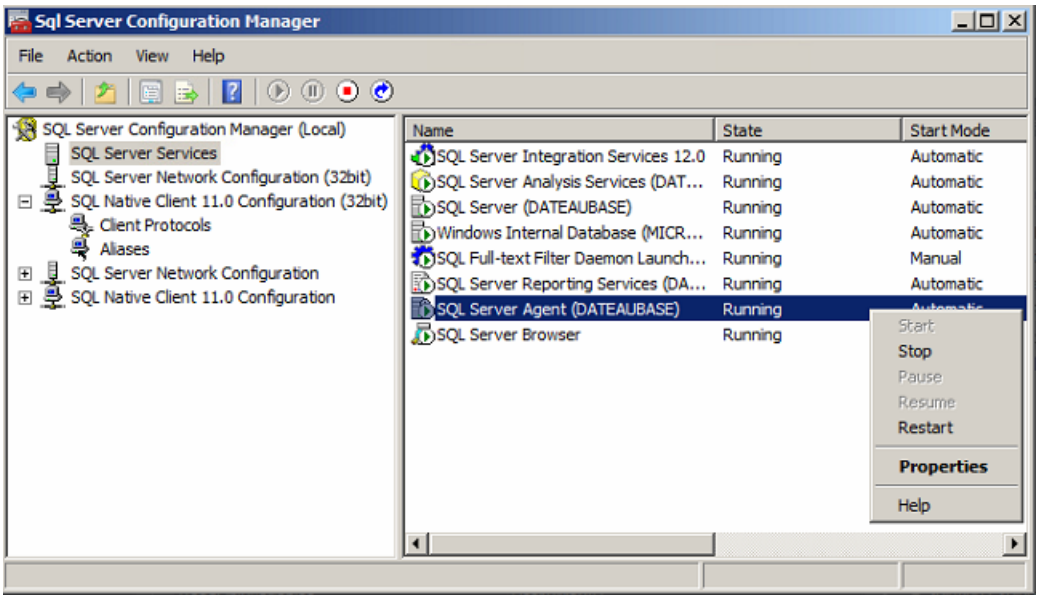

FIGURE 3.11 – Démarrage de l'Agent SQL Server

3.3.2 Créer une tâche planifiée

Dans le SQL Server Management Studio, pour créer une tâche planifiée (Job), il faut créer un nouveau "Job" pour l'Agent que l'on a démarré.

Configuration générale

Dans la fenêtre qui s'ouvre (figure 3.12), il faut remplir le nom de la tâche ainsi que sa description.

Nouvelle étape

Dans l'onglet "Step", il faut créer une nouvelle étape qui contiendra le fichier de notre package SSIS. Les données à entrer sont le nom de notre étape, son type, sous quels privilèges ("Run as") on veut lancer cette étape (laisser par défaut actuellement ; voir section suivante), la source de l'étape et son chemin sur l'ordinateur. Excepté pour les privilèges, la figure 3.13 est un bon exemple de comment remplir ces champs. Dans l'onglet "Step", il faut créer une

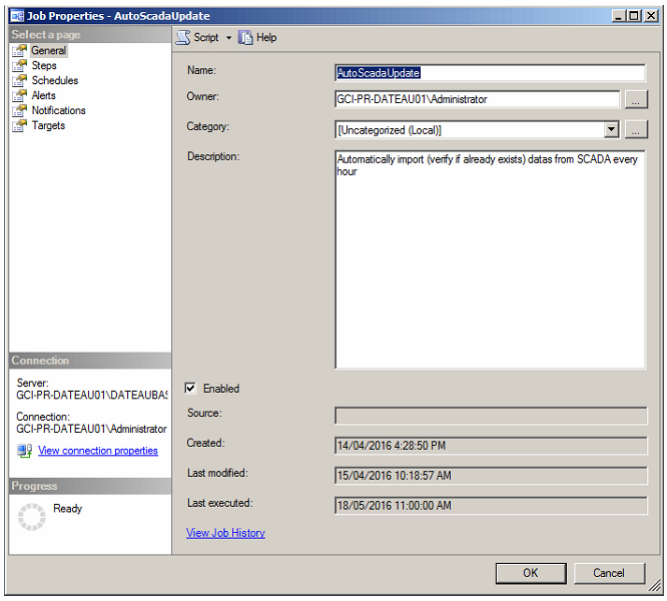

FIGURE 3.12 – Création d'une nouvelle tâche planifiée

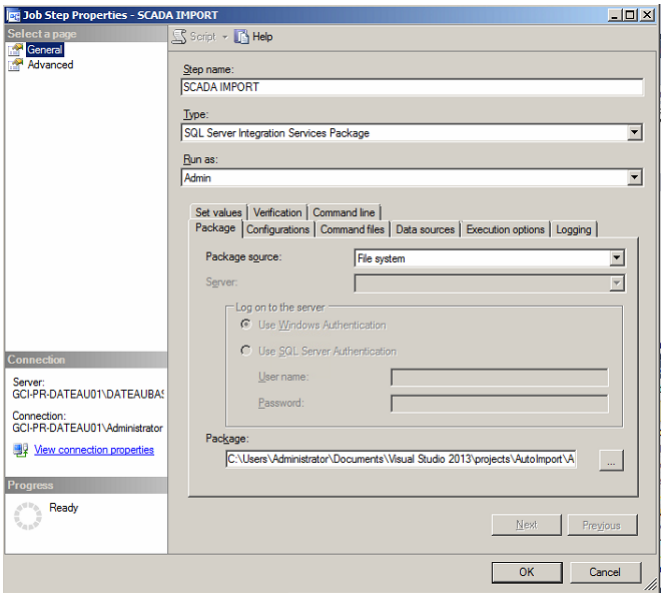

FIGURE 3.13 – Nouvelle étape pour une tâche planifiée

nouvelle étape qui contiendra le fichier de notre package SSIS. Les données à entrer sont le nom de notre étape, son type, sous quels privilèges ("Run as") on veut lancer cette étape (laisser par défaut actuellement ; voir section suivante), la source de l'étape et son chemin sur l'ordinateur. Excepté pour les privilèges, la figure 3.13 est un bon exemple de comment remplir ces champs.

Planifier la tâche

Dans l'onglet "Schedules", créer un nouveau planning correspondant aux désirs de l'utilisateur. La figure 3.14 est un exemple pour planifier la tâche de manière récurrente toutes les heures sans aucune limite de fin de planification.

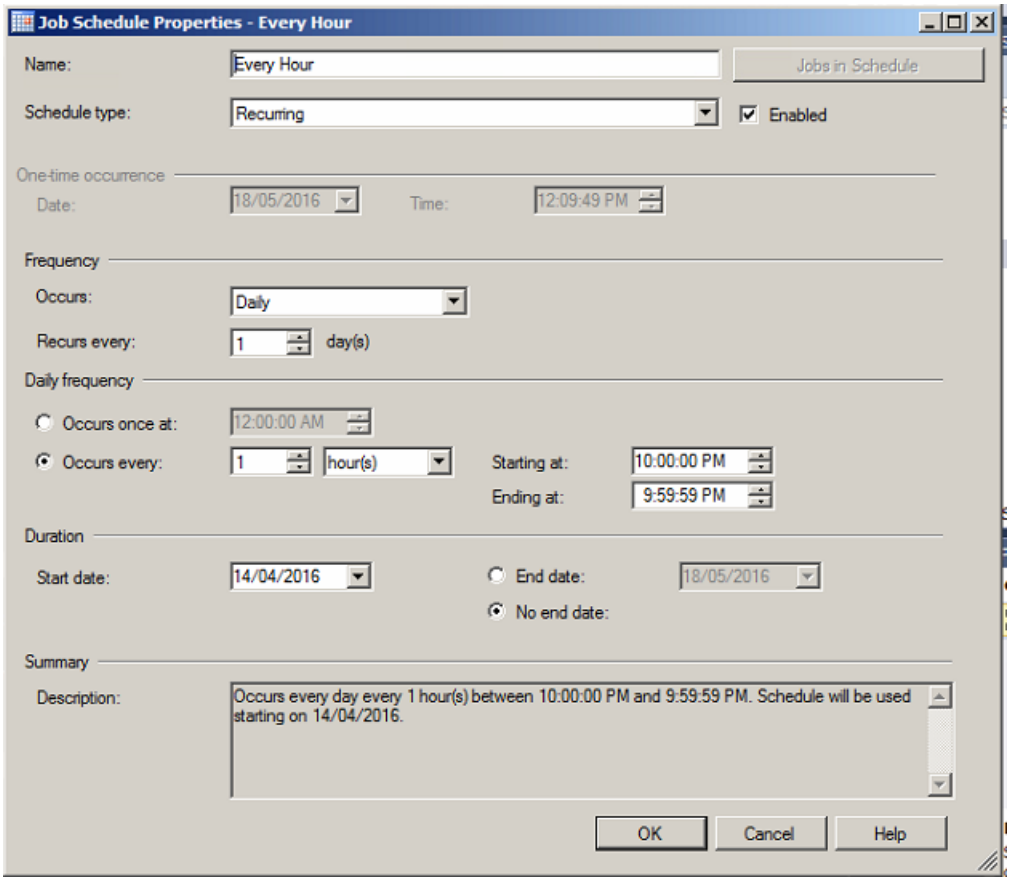

FIGURE 3.14 – Nouveau planning pour la tâche

Après cela, il suffit de valider le planning ainsi que le job pour clôturer sa création. Cependant, pour fonctionner correctement, celui-ci doit acquérir les droits d'administrateur. La section suivante développe comment les obtenir.

3.3.3 Lancer la tâche planifiée en tant qu'administrateur [Anonymous, 2016]

Permettre à la tâche de se lancer en tant qu'administrateur se fait en 3 étapes. D'abord, créer un "Credential" ; ensuite un "Proxy" et, enfin, configurer la tâche pour qu'elle utilise ce dernier.

Création d'un Credential

Dans l'onglet "Security" de SQL Server Management Studio, aller dans le sous-répertoire "Credentials" afin d'en créer un nouveau. Le nommer, le lier au compte Administrateur et entrer le mot de passe associé (figure 3.15).

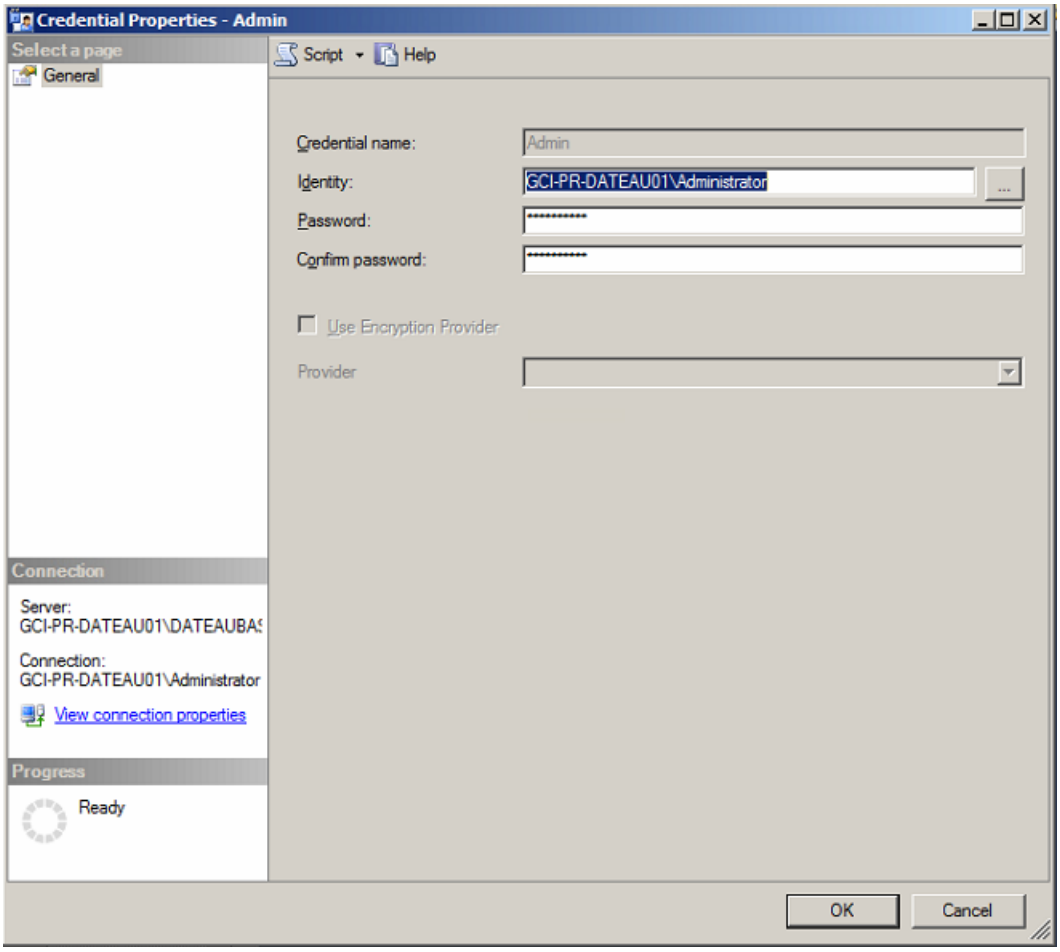

FIGURE 3.15 – Création de Credentials

Création d'un Proxy

Dans l'onglet "Proxies" de votre Agent SQL, aller dans le sous-répertoire SSIS Package Execution afin de créer un nouveau Proxy. La figure 3.16 indique comment remplir la fenêtre qui s'ouvre.

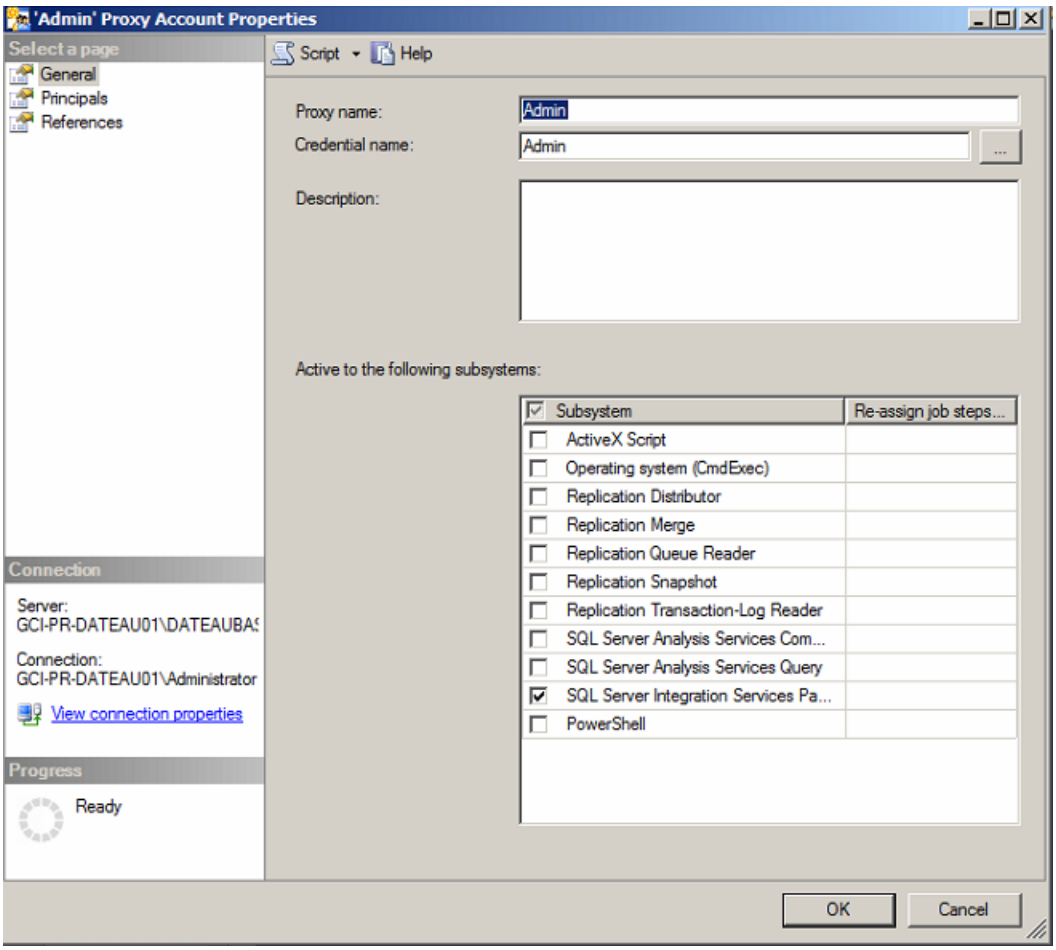

FIGURE 3.16 – Création d'un proxy

Utiliser le proxy dans la tâche

Dans l'option "Run As" de la figure 3.13, sélectionnez le proxy précédemment créé.

Chapitre 4

Import automatique depuis la station mon*EAU*

4.1 Import depuis les fichiers .tsdb

4.1.1 Control Flow

La figure 4.1 montre le flux de contrôle utilisé pour l'import des données venant de BaseStation. Chaque bloc représente une requête SQL qui permet de sélectionner le plus grand timestamp correspondant à un capteur et de le sauvegarder dans une variable. Le code et l'objectif de cette manipulation sont similaires à ce qui est développé à l'annexe C2.2 excepté que le numéro de méta-donnée est différent pour chaque capteur.

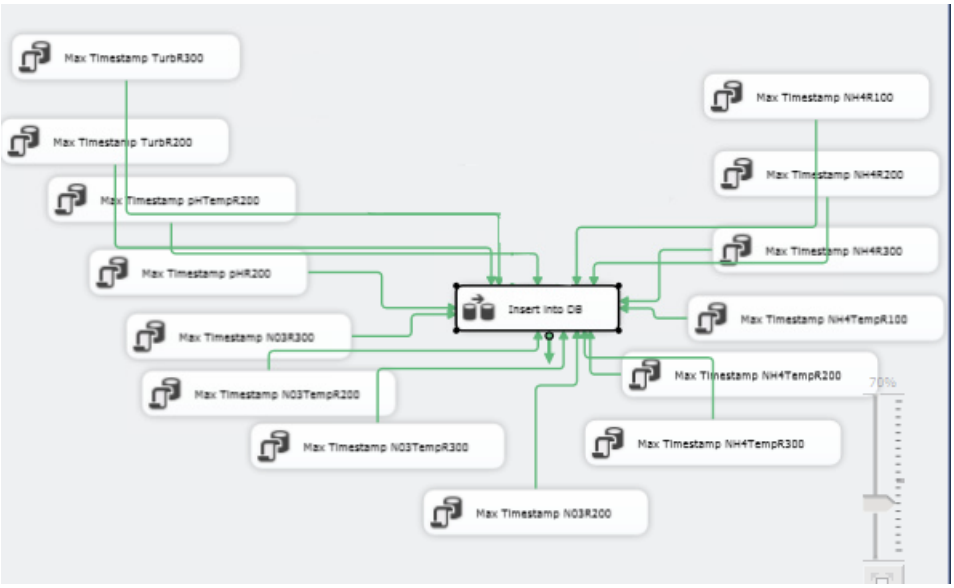

FIGURE 4.1 – Control Flow SSIS pour l'import des données de BaseStation

4.1.2 Data Flow

La figure 4.2 montre le flux de données utilisé pour l'import des données venant de BaseStation. Seul le premier bloc n'a pas été expliqué au chapitre précédent. Il permet de lire le fichier .tsdb.

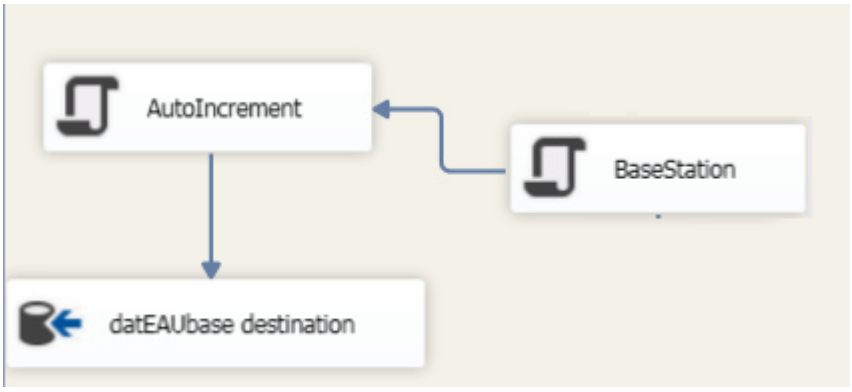

FIGURE 4.2 – Data Flow SSIS pour l'import des données de BaseStation

Fichier .tsdb

Le fichier .tsdb créé à partir de BaseStation est un fichier binaire qui se lit par 56 bytes. Ces bytes sont regroupés sous différentes formes : deux Long (représentant des dates dont seule la première nous intéresse) suivis de 4 Bytes, suivis d'un Double représentant la valeur de la mesure. Chaque fichier correspond à un capteur pour une année.

Lecture du fichier Afin de lire le fichier, il faut utiliser un script permettant la lecture byte par byte et de les regrouper afin de les interpréter selon leur type. Malheureusement, Python gère difficilement les Doubles tels que codés dans le fichier .tsdb. Nous utilisons donc un script Visual Basic (VB) dont la base de travail a été fournie par Primodal. Ce script est inséré dans le Data Flow (premier bloc de la figure 4.2) vu au chapitre précédent. Le code et les variables utilisées se trouvent en Annexe G.

Automatisation

Ici, l'automatisation est déjà mise en place car les blocs ont été ajoutés au même package que pour l'import MSSQL (voir chapitre 3). En effet, plusieurs morceaux de code étant communs, le temps est rentabilisé en utilisant ce qui existe déjà.
Si, par contre, un nouveau package avait été créé, il aurait fallu reproduire les étapes de la section 3.3 (excepté la création de "Credential" et de "Proxy" si ceux-ci ont déjà été faits).

4.2 Import automatique depuis fichiers .par

L'import des données enregistrées par Ana::Pro dans les fichiers .par est réalisé grâce à un script Python.
Les fichiers .par sont des fichiers lisibles avec un éditeur de texte. Ils contiennent une ligne regroupant certaines informations comme le dossier de destination ou le matériel utilisé. Ensuite, une ligne avec le nom des colonnes contenant les données et, enfin, une ligne, toutes les 30 secondes, contenant la date et l'heure de la mesure ainsi que les résultats de celle-ci pour huit paramètres (pH, COD, MeS, etc.). Certaines colonnes ne renvoient pas d'information pertinente et ne seront pas utilisées par la suite.

4.2.1 Principe du code

Le code Python (disponible en Annexe G) permet de se connecter à la dat*EAU* base afin de récolter les informations dont on a besoin (la date de dernière insertion de données p.e.) ainsi que pour insérer de nouvelles données dans cette BDD.
Le script va parcourir tous les fichiers .par du dossier dans lequel ils sont enregistrés. Lorsqu'il détecte, parmi ceux-ci, un fichier (que nous nommerons X) plus récent que la dernière donnée d'Ana::Pro enregistrée dans la dat*EAU* base, il met à jour une variable (index) pour que le programme sache par quel fichier commencer les imports.
Cet index pointera le fichier X-1 (ce qui veut dire le fichier créé juste avant X) car il risque de contenir des valeurs plus récentes que la dernière enregistrée (voir section 5.2). Si aucun fichier n'est plus récent que la dernière mesure de la dat*EAU* base, cela veut dire qu'aucun nouveau fichier n'a été créé depuis le dernier import et que l'on peut utiliser le dernier en date. L'index pointe, par défaut, ce fichier-là. Ceci est très utile car s'il fallait, chaque heure, scanner toutes les valeurs de tous les fichiers .par pour trouver les 120 nouvelles (1 mesure toutes les 30 secondes), il en découlerait une perte d'efficacité et une utilisation inutile du processeur.
Une fois l'index trouvé et le fichier correspondant ciblé, le script parcourt chaque ligne (excepté les deux premières qui ne sont pas pertinentes) de chaque fichier à partir de celui-ci.
Chacune de ces lignes est formatée afin d'en extraire les données pertinentes et, après comparaison entre la date de la donnée et celle de la dernière entrée similaire dans la dat*EAU* base afin de ne prendre que les données postérieures à cet étalon, elles sont introduites dans la base de données.
Certaines valeurs subissent une conversion (de mg/L à kg/m^3) pour correspondre aux unités SI, et l'incrémentation automatique est gérée par le script.

4.2.2 Explication de l'index

Imaginons qu'il est 12:11:45 et que la dernière valeur enregistrée dans la dat*EAU* base date de 11:11:00 (figure 4.3). L'index pointe, par défaut, le dernier fichier (3). Le script va parcourir chaque fichier dans l'ordre jusqu'à en trouver un créé après le dernier enregistrement dans la

BDD. Ainsi, le premier fichier créé après 11:11:00 est le fichier 3 (créé à 11:30:00). Cependant, le fichier 2 contient encore des valeurs qui n'ont pas été importées. C'est pourquoi l'index pointera "X-1", c'est à dire le fichier 2.

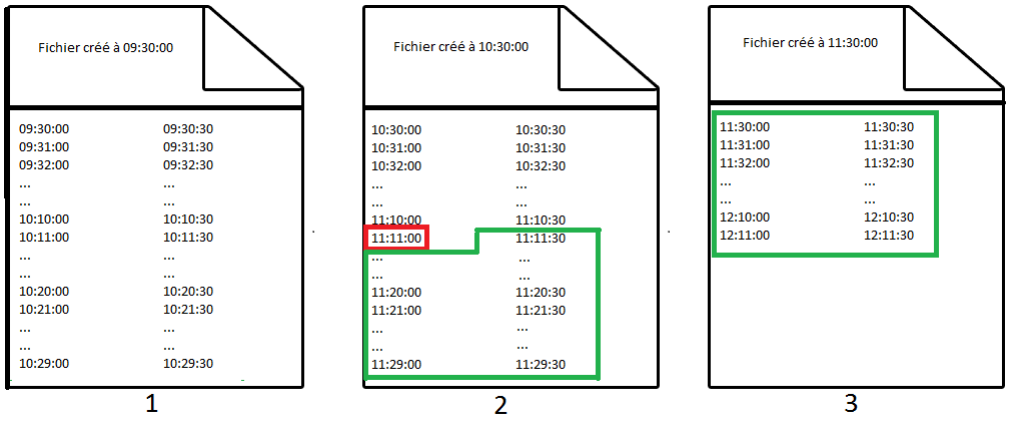

FIGURE 4.3 – Représentation schématique de l'index avec changement de fichier

Imaginons, cette fois-ci, être dans les mêmes conditions excepté que la dernière valeur enregistrée est celle de 11:31:30 (figure 4.4). L'index pointe toujours, par défaut, le fichier 3. Le script va parcourir tous les fichiers mais n'en trouvera aucun créé plus récemment que 11:31:30. L'index ne sera donc pas modifié et le fichier 3 sera le seul parcouru afin de trouver de nouvelles valeurs.

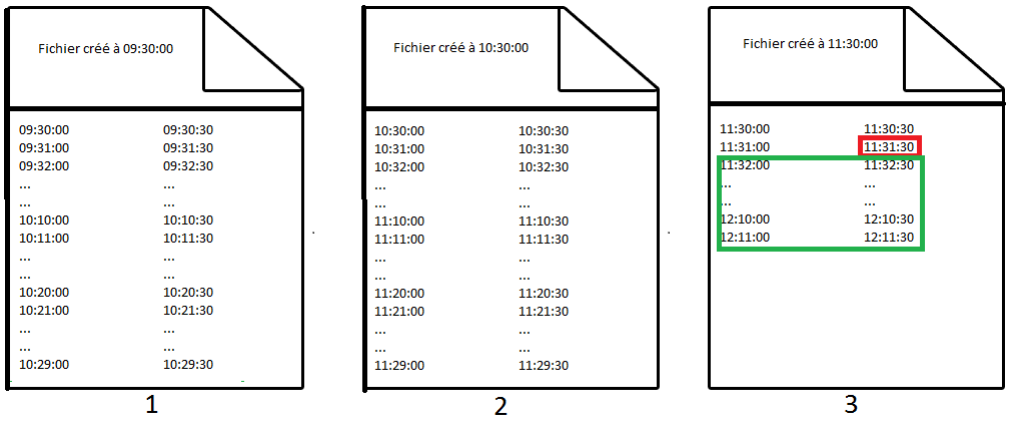

FIGURE 4.4 – Représentation schématique de l'index sans changement de fichier

4.2.3 Automatisation

La gestion de l'automatisation se fait grâce au planificateur de tâches Windows (Windows Task Scheduler).
Une fois cet outil démarré, il faut créer une nouvelle tâche. Une fenêtre (figure 4.5) apparaît, dans laquelle il faut inscrire le nom désiré et une description.
Il est préférable de cocher l'option pour lancer la tâche avec les plus hauts privilèges (Run with highest privileges).

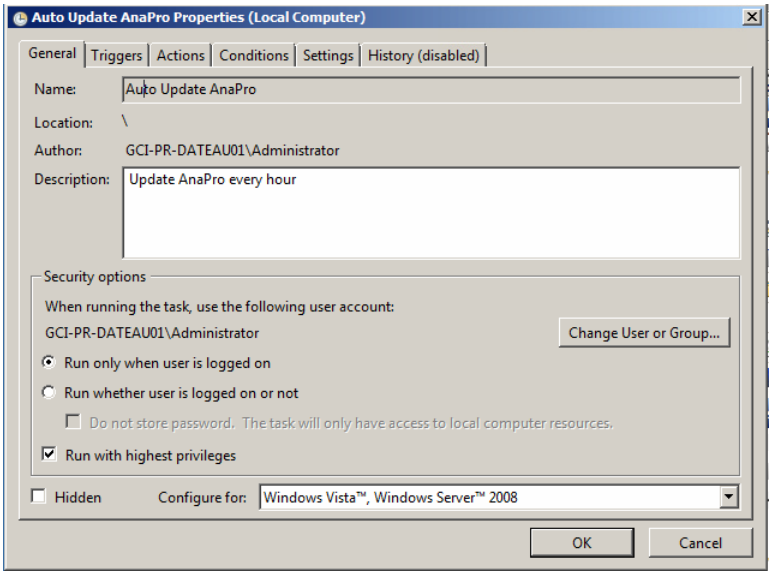

FIGURE 4.5 – Création d'une nouvelle tâche planifiée

Dans l'onglet "Déclencheurs" (Triggers), il faut en créer un nouveau et y inscrire la configuration désirée. La figure 4.6 montre l'exemple d'un déclencheur périodique s'activant, selon un planning (On a schedule) toutes les heures, sans limite dans le temps.
Il est important de ne pas oublier de cocher la case Enabled (Activé en français) pour que le trigger soit opérationnel.

Il faut enfin aller dans l'onglet "Actions" et en créer une nouvelle (figure 4.7). Cette action doit lancer un programme/ un script dont on inscrit le chemin d'accès (ici, l'environnement Python 2.7) avec d'éventuels arguments (ici, le nom du script Python permettant l'import) ainsi que le chemin depuis lequel lancer ce script (le dossier dans lequel il se trouve).
Les autres onglets gardent leur configuration par défaut.

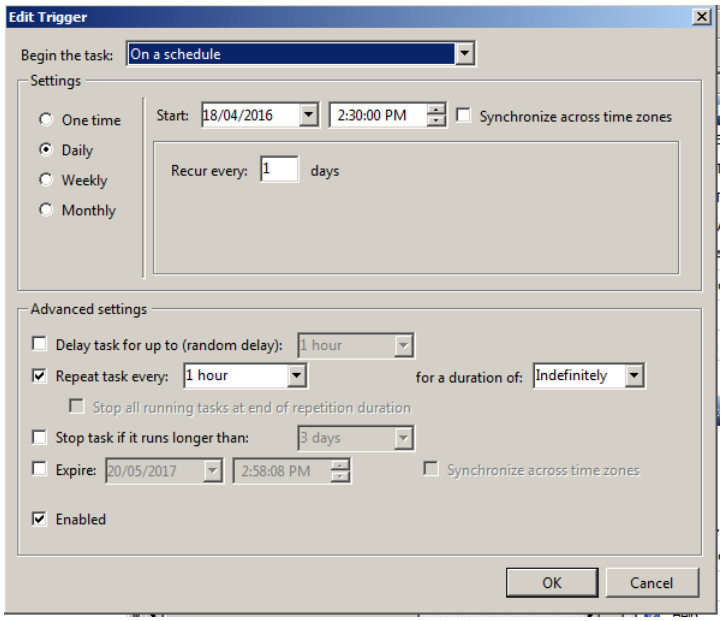

FIGURE 4.6 – Planification de la tâche

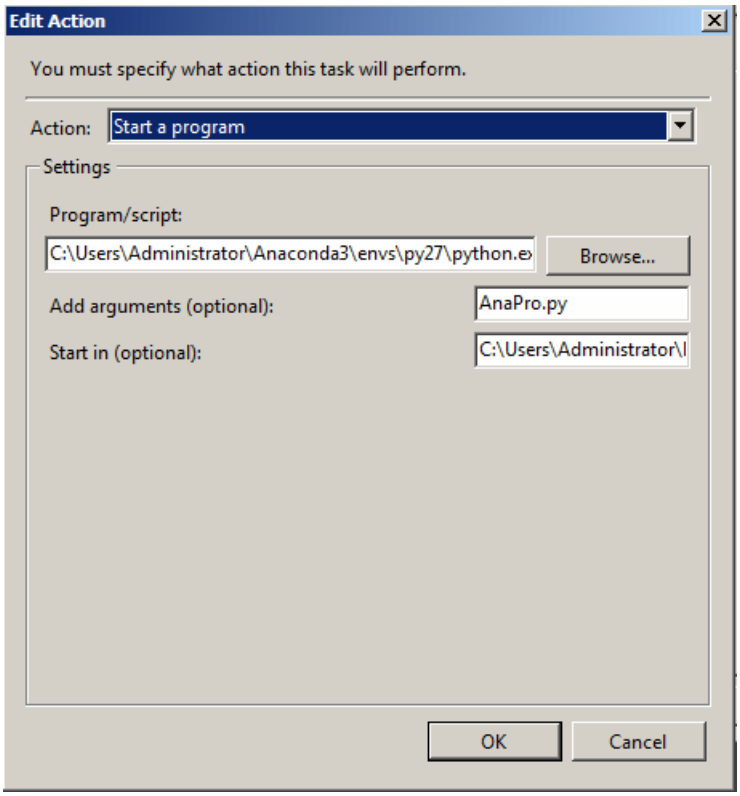

FIGURE 4.7 – Action à réaliser par la tâche planifiée

Conclusion

On remarque que l'intégration de différentes bases de données, sous plusieurs formes, est possible de manière formalisée sous la même structure finale. Les outils à utiliser varient mais aboutissent toujours (pour les cas étudiés) à un résultat formaté selon nos besoins.
Les informations majeures qui ont dirigé la recherche pour la conversion et l'import sont :

- La possibilité ou non de faire un import à chaud quel que soit le programme utilisé ;
- La manière de lire les fichiers (p.ex. le fichier .par est lisible dans un traitement de texte alors que le fichier .tsdb était à lire grâce à un lecteur binaire) ; et
- La structure du fichier afin de savoir quelle information en retirer.

Il reste cependant certaines pistes à exploiter pour améliorer le projet et faciliter son utilisation. Voici une liste non-exhaustive de celles-ci :

- Une interface permettant l'import des données de laboratoire ainsi que l'export des données existantes (ou la représentation graphique de celles-ci) devrait être développée afin de faciliter l'utilisation de la dat*EAU*base.
- La gestion automatique des conditions météorologiques en cherchant les données automatiquement sur internet pourrait faciliter la sauvegarde de cette information.
- Le code Visual Basic pourrait être optimisé pour être plus rapide à l'exécution en calculant combien de bytes il faudrait sauter avant de se trouver à la date désirée.
- Un script pourrait gérer les modifications à appliquer dans les différents codes lorsque l'on change un capteur et, par conséquent, d'identifiant de métadonnée.
- La gestion automatique des backups pourrait être implémentée.
- Etc.

Troisième partie

ANNEXES

Annexe A

Installation d'OpenVPN

L'installation d'OpenVPN se fait en plusieurs étapes. On considère, dans cette annexe, que le programme se trouve déjà sur votre ordinateur et que vous avez ouvert une invite de commande dont le chemin se trouve être celui du dossier easy-rsa d'OpenVPN.

A.1 Autorité de certification

A.1.1 Initialisation

La première chose à faire est de créer son autorité de certification avec sa clé privée et son certificat.
Pour ce faire, il faut initialiser la configuration grâce à la commande :

```
init-config
```

Ensuite, il faut modifier le fichier vars.bar en double cliquant dessus ou en utilisant la commande :

```
notepad vars.bat
```

Seuls les champs suivants sont à modifier selon votre cas :

```
set KEY_COUNTRY=VOTRE\_PAYS
set KEY_PROVINCE=VOTRE\_PROVINCE
set KEY_CITY=VOTRE\_VILLE
set KEY_ORG=VOTRE\_ORGANISATION
set KEY_EMAIL=VOTRE\_EMAIL
```

Après sauvegarde du fichier, supprimer toutes les anciennes clés éventuelles de votre dossier (à faire en connaissance de cause) et initialiser les nouvelles valeurs grâce aux commandes :

```
clean-all
vars
```

A.1.2 Création de l'autorité de certification

Après l'initialisation, la création de l'autorité de certification se fait grâce à la commande :

```
build-ca
```

Il suffit de remplir les champs demandés (normalement pré-remplis avec les valeurs modifiées dans vars.bat).

A.2 Création des différentes clés

A.2.1 Création des clés du serveur

La création des clés du serveur se fait grâce à la commande :

```
build-key-server NOM\_DU\_SERVEUR
```

Il suffit ensuite de remplir les champs demandés dans l'invite de commande et de répondre oui aux questions posées (cela sert à signer le certificat).

A.2.2 Création des clés des clients

La création des clés des clients se fait, pour chaque client, grâce à la commande :

```
build-key-server NOM\_DU\_CLIENT
```

A.2.3 Création de la clé de Diffie-Hellman

La clé de Diffie-Hellman est générée grâce à la commande :

```
build-dh
```

A.3 Configuration serveur et client

A.3.1 Préparation

Tout d'abord, il faut récupérer les clés créées (pour le serveur : dh.pem, ca.crt, NOM_DU_SERVEUR.crt et NOM_DU_SERVEUR.key ; pour les clients : ca.crt, NOM_DU_CLIENT.crt et NOM_DU_CLIENT.key) et les placer respectivement pour le serveur et pour les clients dans le répertoire souhaité de l'ordinateur concerné (OpenVPN/config).
Une fois cela fait, il faut récupérer, pour le serveur et pour les clients, le fichier de configuration correspondant dans "OpenVPN/OpenVPN Sample Configuration Files" et respectivement les placer dans le dossier OpenVPN/config de l'ordinateur concerné.

A.3.2 Configuration du serveur

Il faut éditer le fichier server.conf afin de changer les lignes suivantes :

```
ca ca.crt
cert server.crt
key server.key
dh dh1024.pem
```

Celles-ci doivent être modifiées pour contenir le chemin d'accès aux différents certificats et aux clés. Voici un exemple :

```
ca "C:\\Program Files\\OpenVPN\\config\\ca.crt"
cert "C:\\Program Files\\OpenVPN\\config\\server.crt"
key "C:\\Program Files\\OpenVPN\\config\\server.key"
dh "C:\\Program Files\\OpenVPN\\config\\dh1024.pem"
```

A.3.3 Configuration des clients

La configuration est similaire à celle du serveur excepté qu'une ligne supplémentaire doit être éditée. Ainsi :

```
remote my-server-1 1194
```

doit devenir :

```
remote ADRESSE_IP_DU_SERVEUR PORT_UTILISE
```

Annexe B

Création, affectation et utilisation de variables

Les variables, qu'elles soient d'entrée ou de sortie, en lecture ou en lecture/écriture, seront beaucoup utilisées dans SSIS. Il est donc important de bien comprendre comment les créer et les utiliser.

B.0.1 Création et affectation de variables

Pour les requêtes SQL

Dans le bloc permettant de réaliser une tâche SQL, dans l'onglet général, il faut choisir d'obtenir le résultat de la requête sous forme de "single row" (figre B.1)

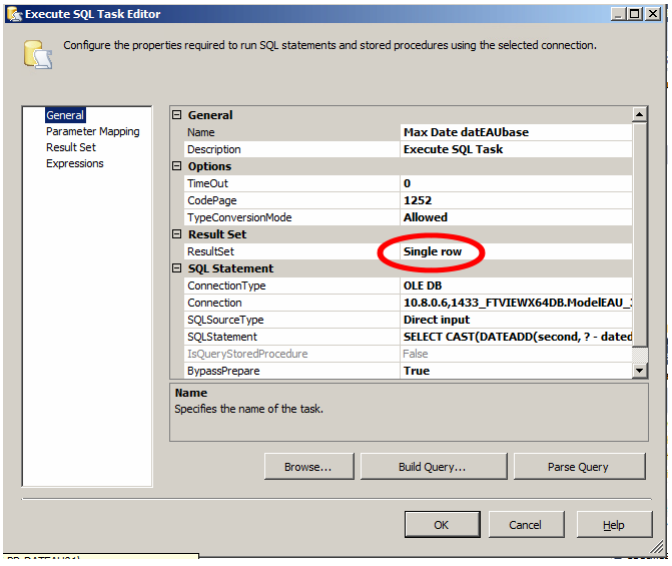

FIGURE B.1 – Sortie de requête SQL sous forme de ligne unique

Dans l'onglet "Result Set", il faut ajouter un nouvel ensemble de résultats (afin d'affecter la variable que l'on utilisera), le nommer et définir la variable à utiliser. Si celle-ci n'existe pas, il faut la créer (figures B.2 et B.3).// Une fois réalisé, la variable sera disponible dans tout le reste du programme.

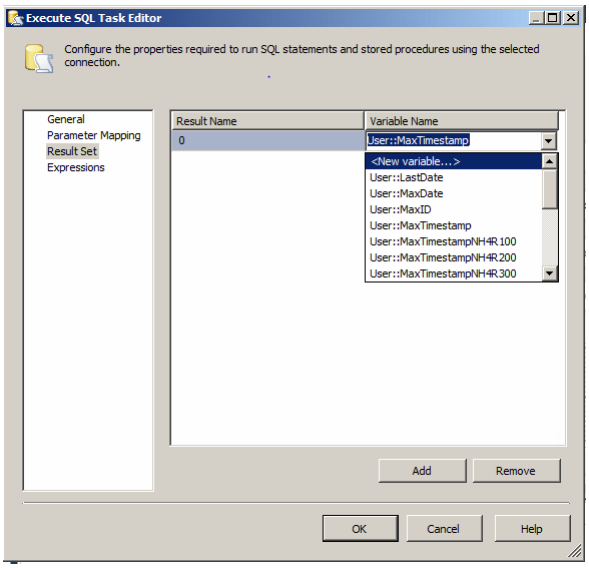

FIGURE B.2 – Création d'un nouveau Result Set

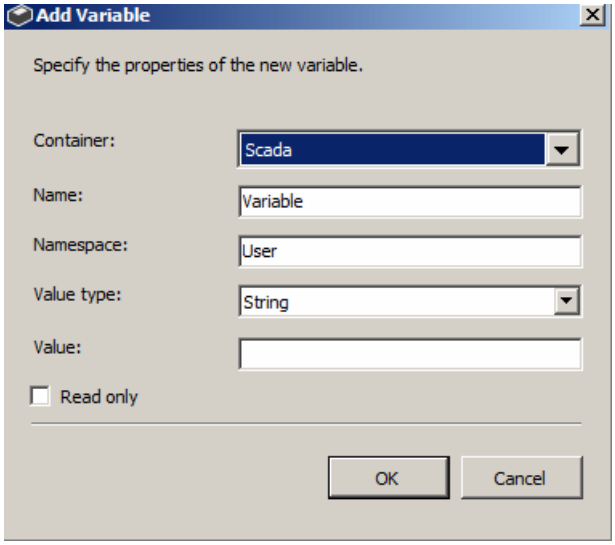

FIGURE B.3 – Création d'une nouvelle variable pour une requête SQL

Pour un script

Pour un script, la création de variable se fait avant la programmation tandis que son affectation se trouve dans le code de manière standard.
Afin de créer une variable, il faut se rendre dans l'onglet "Inputs and Outputs" du bloc de configuration du script, de développer les sorties (Outputs) puis d'ajouter une colonne (Add Column). Enfin, il suffit de donner un nom à cette colonne ainsi que de préciser son type comme indiqué à la figure B.4.
L'affectation de la variable dans le code se fait en faisant appel à "Row.LA_VARIABLE".

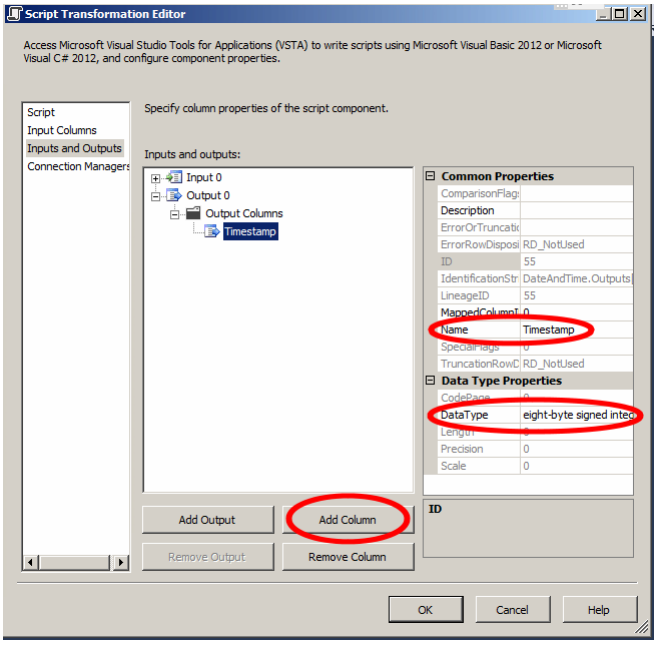

FIGURE B.4 – Création d'une nouvelle variable pour un script

B.0.2 Utilisation de variables

Pour une requête SQL

Dans la requête SQL, si l'on utilise une variable, celle-ci sera représentée par un point d'interrogation (voir Annexe C2.3). Pour que le programme puisse interpréter ce à quoi correspond le paramètre, il faut faire un mapping dans l'onglet correspondant comme représenté à la figure B.5.

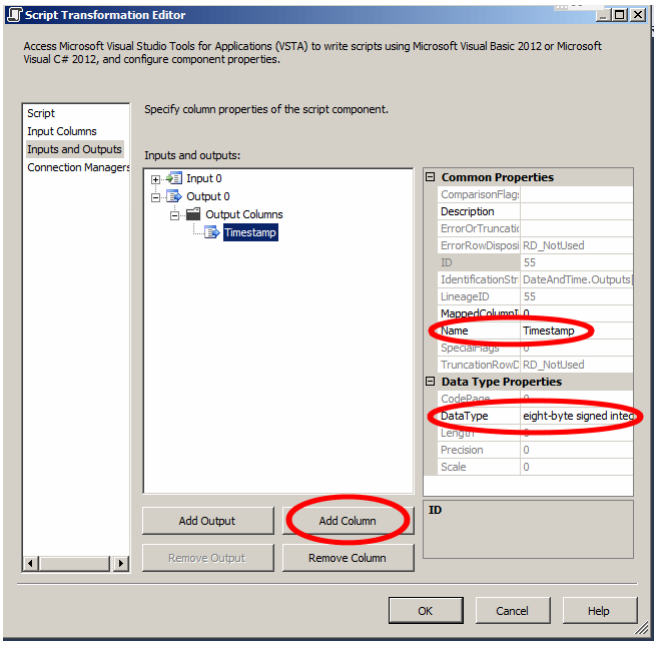

FIGURE B.5 – Mapping de variables pour une requête SQL

Pour un script

Deux possibilités peuvent subsister.

Soit on veut utiliser une variable d'entrée venant du flux, pour laquelle il faut faire le mapping comme montré à la figure B.6.
Dans ce cas, ne pas oublier qu'il faut changer le ReadOnly (lecture seule) en Read and Write (lecture et écriture) si l'on veut modifier la variable.
L'utilisation de cette variable dans le code se fait en appelant "Row.LA_VARIABLE".

Soit on utilise une des variables créées grâce à nos requêtes SQL. Leur ajout est différent des autres variables utilisées jusque maintenant. Celles-ci doivent être ajoutées dans la fenêtre générale du script comme indiqué à la figure B.7.
Pour les utiliser dans le code, il faut faire appel à "Variables.LA_VARIABLE".

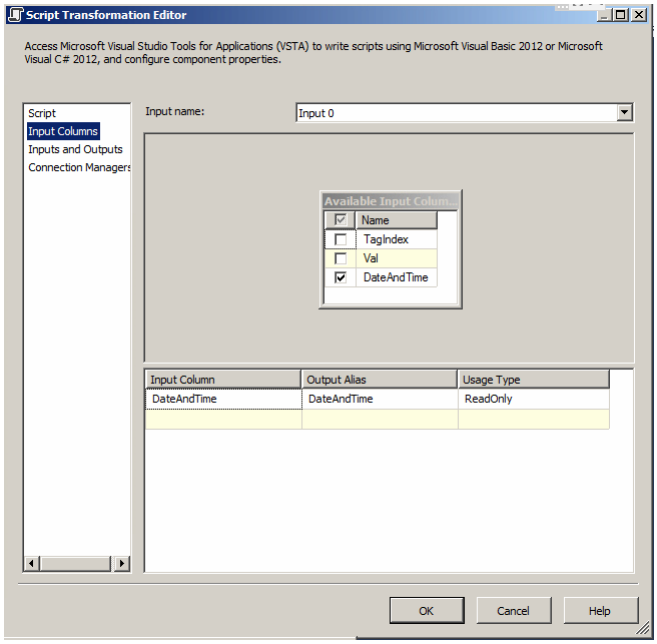

FIGURE B.6 – Choix des variables d'entrée pour un script

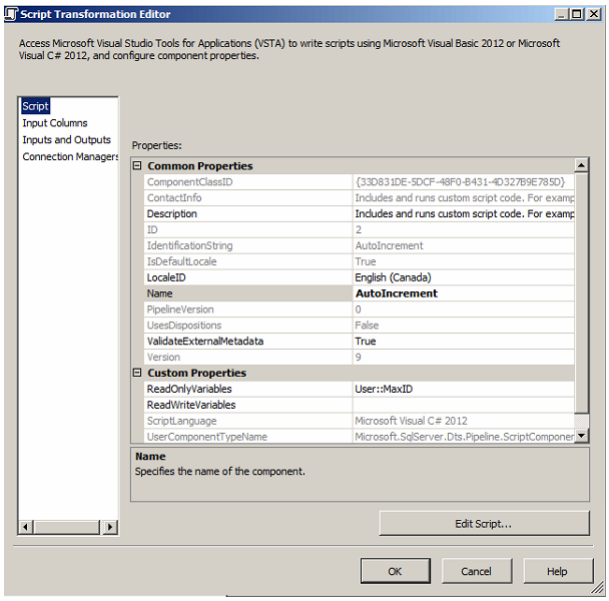

FIGURE B.7 – Choix des variables externes au flux pour un script

Annexe C

Requêtes SQL

C.1 Code

C.1.1 ID maximum

Avoir l'ID Maximum des données de la dat*EAU*base permet, par la suite, de l'utiliser afin d'incrémenter automatiquement et éviter d'avoir 2 fois le même identifiant. Voici la requête SQL exécutée :

```
SELECT ISNULL(MAX(VALUE_ID),0) FROM dateaubase.dbo.value
```

Cette requête permet, si aucune valeur n'est trouvée (si on lance le script pour la première fois), de dire que l'ID max est égal à 0.

C.1.2 Timestamp maximum

Connaître le timestamp de la dernière valeur de la dat*EAU*base permet de le transformer en date et heure afin, par la suite, de n'extraire du SCADA que les données postérieures à celle-ci. Voici la requête SQL exécutée :

```
SELECT ISNULL(MAX(Timestamp),0)
FROM dateaubase.dbo.value
WHERE Metadata_ID = 9
```

Cette requête recherche le timestamp maximum pour lequel l'identifiant de métadonnée vaut 9 (premier capteur du SCADA dans des conditions standard). Si aucune valeur n'est trouvée, la réponse sera 0.

C.1.3 Date et heure à partir du timestamp

Le but de cette requête est de transformer le timestamp précédemment enregistré en DateTime (date et heure en une seule variable) afin, par la suite, de n'extraire que les données postérieures à celle-ci. Voici la requête SQL exécutée :

```
SELECT CAST(DATEADD(second, ? - datediff(s, GETDATE(), GETUTCDATE()) ,
'19700101' )AS DATETIME)
```

Cette requête fait la différence en secondes entre le paramètre (représenté par un point d'interrogation) qu'est le timestamp, corrigé en fonction du fuseau horaire, et le premier janvier 1970. Cette différence est ensuite sauvegardée comme un DateTime.

Annexe D

Conversion de dates et heures en timestamp

D.1 Variables et paramètres

Il y a, pour ce script, une variable en entrée et une variable en sortie. La première est le DateTime récupéré grâce à la connexion au serveur distant. La deuxième est un entier signé de 8 bytes (8-byte signed integer) contenant le timestamp.

D.2 Code

Le reste du code étant généré automatiquement, seule la fonction Input0_ProcessInputRow sera développée.

Ce script se découpe en différentes parties. D'abord, l'initialisation des variables. Ensuite la créaction de la date "epoch" (1er janvier 1970). Puis la création des différents fuseaux horaires (GMT comme heure de référence pour le timestamp, EDT pour la conversion à l'heure d'été et EST pour la conversion à l'heure d'hiver). Par après, on utilise la fonction "IsDaylightSavingTime" sur le DateTime en entrée afin de savoir s'il s'agit d'une date/heure à l'heure d'hiver ou non. En fonction du résultat de cette fonction, on applique la conversion de EDT à GMT ou de EST à GMT. Enfin, on sauvegarde en sortie le timestamp correspondant.

```
public override void Input0\_ProcessInputRow(Input0Buffer Row)
{
    // Initialize varibles
    TimeZoneInfo timeZoneGMT;
    TimeZoneInfo timeZoneEST;
    TimeZoneInfo timeZoneEDT;
    DateTime Converted_dateTime;
    DateTime epoch;

    // epoch = 1 January 1970
    epoch = new DateTime(1970, 1, 1, 0, 0, 0);

    // Define EDT = GMT - 4
    timeZoneEDT = TimeZoneInfo.FindSystemTimeZoneById
    ("Atlantic Standard Time");
    // Define GMT
    timeZoneGMT = TimeZoneInfo.FindSystemTimeZoneById
    ("GMT Standard Time");
    // Define EST = GMT - 5
    timeZoneEST = TimeZoneInfo.FindSystemTimeZoneById
    ("Eastern Standard Time");

    // Check if Daylight saving time
    if (TimeZoneInfo.Local.IsDaylightSavingTime(Row.DateAndTime
        ↪ )){
        // if it is : Convert time from EDT
        Converted_dateTime = TimeZoneInfo.ConvertTime
        (Row.DateAndTime, timeZoneEDT, timeZoneGMT);
    }
    else {
        // if it is not : Convert time from EST
        Converted_dateTime = TimeZoneInfo.ConvertTime
        (Row.DateAndTime, timeZoneEST, timeZoneGMT);
    }
    // Timestamp = (time after conversion - epoch) in seconds
    Row.Timestamp=(long)(Converted_dateTime-epoch).TotalSeconds
        ↪ ;
}
```

Annexe E

Lien avec les métadonnées et conversion des unités

E.1 Variables

Les variables utilisées en entrée sont : le TagIndex (identifiant du capteur) et la valeur de mesure (Val). La première sert à être liée à une métadonnée, la deuxième à être convertie en fonction de son unité et de l'unité SI correspondante. Cette dernière étant modifiée lors du script, elle doit être en lecture/écriture.
Les variables de sortie sont un identifiant de métadonnée (MetaID) et le nombre d'expériences (NumberExp) sous forme d'entier signé de 4 bytes (four-byte signed integer). La valeur de mesure étant en entrée en lecture/écriture, elle sera modifiée mais ne doit pas apparaître dans les sorties.

E.2 Code

```
public override void Input0_ProcessInputRow(Input0Buffer Row)
    {
        // Add Number of experiments = 1 (because of on-line data)
        Row.NumberExp = 1;
        switch (Row.TagIndex){
            case 0:
                // AIT 110 | uS/cm => S/m => 10 000/1
                Row.MetaID = 9;
                Row.Val = Row.Val / 10000;
                break;
            case 1:
                // AIT 240 | mg/L => g/m^{3} => 1 000/1
```

```
            Row.MetaID = 10;
            break;
        case 3:
            // AIT 241 | mg/L => g/m^{3} => 1 000/1
            Row.MetaID = 11;
            break;
        case 4:
            // AIT 260 | mg/L => g/m^{3} => 1 000/1
            Row.MetaID = 12;
            break;
        case 5:
            // AIT 340 | mg/L => g/m^{3} => 1 000/1
            Row.MetaID = 13;
            break;
        case 6:
            // AIT 341 | mg/L => g/m^{3} => 1 000/1
            Row.MetaID = 14;
            break;
        case 7:
            // AIT 360 | mg/L => g/m^{3} => 1 000/1
            Row.MetaID = 15;
            break;
        case 8:
            //TIT 111 |  \degre C
            Row.MetaID = 16;
            break;
        case 9:
            //TIT 211 | \degre C
            Row.MetaID = 17;
            break;
        case 10:
            //TIT 212 | \degre C
            Row.MetaID = 18;
            break;
        case 11:
            //TIT 241 | \degre C
            Row.MetaID = 19;
            break;
        case 12:
```

```
            //TIT 311 | \degre C
            Row.MetaID = 20;
            break;
        case 13:
            //TIT 312 | \degre C
            Row.MetaID = 21;
            break;
        case 14:
            //TIT 341 | \degre C
            Row.MetaID = 22;
            break;
        case 15:
            //FCV 410 | L/min => m^{3}/s => 60 000/1
            Row.MetaID = 23;
            Row.Val = Row.Val / 60000;
            break;
        case 16:
            //FCV 420 | L/min => m^{3}/s => 60 000/1
            Row.MetaID = 24;
            break;
        case 17:
            //FCV 430 | L/min => m^{3}/s => 60 000/1
            Row.MetaID = 25;
            Row.Val = Row.Val / 60000;
            break;
        case 18:
            //FCV 440 | L/min => m^{3}/s => 60 000/1
            Row.MetaID = 26;
            Row.Val = Row.Val / 60000;
            break;
        case 19:
            //FCV 450 | L/min => m^{3}/s => 60 000/1
            Row.MetaID = 27;
            Row.Val = Row.Val / 60000;
            break;
        case 20:
            //FCV 460 | L/min => m^{3}/s => 60 000/1
            Row.MetaID = 28;
            Row.Val = Row.Val / 60000;
```

```
        break;
    case 21:
        //FIT 011 | m^{3}/h => m^{3}/s => 3 600/1
        Row.MetaID = 29;
        Row.Val = Row.Val / 3600;
        break;
    case 22:
        //FIT 021 | m^{3}/h => m^{3}/s => 3 600/1
        Row.MetaID = 30;
        Row.Val = Row.Val / 3600;
        break;
    case 23:
        //FIT 100 | m^{3}/h => m^{3}/s => 3 600/1
        Row.MetaID = 31;
        Row.Val = Row.Val / 3600;
        break;
    case 24:
        //FIT 110 | m^{3}/h => m^{3}/s => 3 600/1
        Row.MetaID = 32;
        Row.Val = Row.Val / 3600;
        break;
    case 25:
        //FIT 120 | m^{3}/h => m^{3}/s => 3 600/1
        Row.MetaID = 33;
        Row.Val = Row.Val / 3600;
        break;
    case 26:
        //FIT 250 | m^{3}/h => m^{3}/s => 3 600/1
        Row.MetaID = 34;
        Row.Val = Row.Val / 3600;
        break;
    case 27:
        //FIT 260 | m^{3}/h => m^{3}/s => 3 600/1
        Row.MetaID = 35;
        break;
    case 28:
        //FIT 350 | m^{3}/h => m^{3}/s => 3 600/1
        Row.MetaID = 36;
        Row.Val = Row.Val / 3600;
```

```
            break;
        case 29:
            //FIT 360 | m^{3}/h => m^{3}/s => 3 600/1
            Row.MetaID = 37;
            Row.Val = Row.Val / 3600;
            break;
        case 30:
            // FIT 410 | L/min => m^{3}/s => 60 000/1
            Row.MetaID = 38;
            Row.Val = Row.Val / 60000;
            break;
        case 31:
            // FIT 420 | L/min => m^{3}/s => 60 000/1
            Row.MetaID = 39;
            Row.Val = Row.Val / 60000;
            break;
        case 32:
            // FIT 430 | L/min => m^{3}/s => 60 000/1
            Row.MetaID = 40;
            Row.Val = Row.Val / 60000;
            break;
        case 33:
            // FIT 440 | L/min => m^{3}/s => 60 000/1
            Row.MetaID = 41;
            Row.Val = Row.Val / 60000;
            break;
        case 34:
            // FIT 450 | L/min => m^{3}/s => 60 000/1
            Row.MetaID = 42;
            Row.Val = Row.Val / 60000;
            break;
        case 35:
            // FIT 460 | L/min => m^{3}/s => 60 000/1
            Row.MetaID = 43;
            Row.Val = Row.Val / 60000;
            break;
        case 36:
            // LIT 100 | m
            Row.MetaID = 44;
```

```
                    break;
            }
        }
```

Encore une fois, seule la fonction modifiée a été copiée dans le rapport, le reste du code étant généré automatiquement.

Annexe F

Incrémentation automatique

F.1 Variables

Il y a, pour l'incrémentation automatique, appel à une variable d'entrée en lecture seule. Cette variable étant l'ID max récupéré dans le Control Flow, il faut l'ajouter comme expliqué dans l'annexe B.2.2 (figure B.7).

En sortie, seul l'ID sera donné, sous forme d'entier signé de 8 bytes (eight-byte signed integer).

F.2 Code

```
long i = 1;
public override void Input0_ProcessInputRow(Input0Buffer Row)
{
    Row.ID = this.Variables.MaxID + i;
    i = i + 1;
}
```

Encore une fois, seule la portion de code modifiée a été copiée dans le rapport, le reste du code étant généré automatiquement.

Annexe G

Import depuis BaseStation

G.1 Variables

Les variables d'entrée sont les timestamps maximum récupérés dans le Control Flow. Ce sont des variables en lecture seule à ajouter comme expliqué en E.1.
En sortie, on retrouve un timestamp sous forme d'entier signé de 8 bytes (eight-byte signed integer) ; la valeur de la mesure sous forme de nombre à virgule flottante de double précision (double-precision float) ainsi que le nombre d'expériences (toujours égal à 1 pour une mesure en ligne) et l'identifiant de métadonnée sous forme d'entier signé de 4 bytes (four-bytes signed integer).

G.2 Code

```
    'Initialise the variables
    Dim dateLong1 As Long
    Dim dateLong2 As Long
    Dim var3 As Byte
    Dim var4 As Byte
    Dim var5 As Byte
    Dim var6 As Byte
    Dim value As Double

    Dim date1 As Date
    Dim epoch As Date = New DateTime(1970, 1, 1, 0, 0, 0)

    Dim ts As Long

    Dim counter As Integer = 0
```

```
'Functiun to import Data
Public Sub ImportData(path As String, param As Double, meta As
    ↪ Integer)
    If (File.Exists(path)) Then
        Try
            'Read the file as a BinaryFile
            Using reader As BinaryReader = New BinaryReader(
                ↪ File.Open("path", FileMode.Open))
                'While there is something not read anymore :
                Do While (counter < reader.BaseStream.Length)
                    'read a long (8 bytes)
                    dateLong1 = reader.ReadInt64()
                    'read a long (8 bytes)
                    dateLong2 = reader.ReadInt64()
                    ' read a Byte
                    var3 = reader.ReadByte()
                    '...
                    var4 = reader.ReadByte()
                    var5 = reader.ReadByte()
                    var6 = reader.ReadByte()
                    value = reader.ReadDouble()
                    'cast dateLong1 as a Date
                    date1 = New Date(dateLong1)
                    'timestamp is created as a Long that equals
                        ↪ the date-epoch is seconds
                    ts = CLng((date1 - epoch).TotalSeconds)
                    'if the timestamp is higher than the last
                        ↪ data (for this parameter) in the
                        ↪ datEAUbase
                    If (ts > param) Then
                        'if the timestamp is higher than a
                            ↪ value (to reject false values)
                        If (ts > 1325376000) Then
                            'if the value isn't equal to -1 (if
                                ↪ the value exists)
                            If (value <> -1) Then
                                'add a Row as output with the
                                    ↪ value, the metadata, the
```

```
                    ↪ timestamp and the number
                    ↪ of experiments
                With Output0Buffer
                    .AddRow()
                    'if the metadata correspond
                        ↪  to a value in mg/L
                        ↪ => value/1000 to be
                        ↪ in kg/m^{3}
                    If (meta = 49 OrElse meta =
                        ↪  51 OrElse meta = 53
                        ↪ OrElse meta = 55
                        ↪ OrElse meta = 57)
                        ↪ Then
                        .Val = value / 1000
                    Else
                        .Val = value
                    End If
                    .MetaID = meta
                    .Timestamp = ts
                    .NumberExp = 1
                End With
            End If
        End If
    End If
    'value = -1 (to test at each iteration if
        ↪ the value exists
    value = -1
    'increment the counter
    counter = counter + 1
  Loop
 End Using
 Catch ex As Exception
 End Try
 End If
End Sub

'Main function
Public Overrides Sub CreateNewOutputRows()
    'use ImportData for each file
```

```
        ImportData("\\10.8.0.14\DB_Raw\100924-135506_TurbR200_2016.
            ↪ tsdb", Variables.MaxTimestampTurbR200, 45)
        ImportData("\\10.8.0.14\DB_Raw\100924-135506_TurbR300_2016.
            ↪ tsdb", Variables.MaxTimestampTurbR300, 46)
        ImportData("\\10.8.0.14\DB_Raw\100924-135506_pHR200_2016.
            ↪ tsdb", Variables.MaxTimestamppHR200, 47)
        ImportData("\\10.8.0.14\DB_Raw\100924-135506
            ↪ _pHTempR200_2016.tsdb", Variables.
            ↪ MaxTimestamppHTempR200, 48)
        ImportData("\\10.8.0.14\DB_Raw\100924-135506_NH4R200_2016.
            ↪ tsdb", Variables.MaxTimestampNH4R200, 49)
        ImportData("\\10.8.0.14\DB_Raw\100924-135506
            ↪ _NH4TempR200_2016.tsdb", Variables.
            ↪ MaxTimestampNH4TempR200, 50)
        ImportData("\\10.8.0.14\DB_Raw\100924-135506_NO3R200_2016.
            ↪ tsdb", Variables.MaxTimestampNO3R200, 51)
        ImportData("\\10.8.0.14\DB_Raw\100924-135506
            ↪ _NO3TempR200_2016.tsdb", Variables.
            ↪ MaxTimestampNO3TempR200, 52)
        ImportData("\\10.8.0.14\DB_Raw\100924-135506_NH4R100_2016.
            ↪ tsdb", Variables.MaxTimestampNH4R100, 53)
        ImportData("\\10.8.0.14\DB_Raw\100924-135506
            ↪ _NH4TempR100_2016.tsdb", Variables.
            ↪ MaxTimestampNH4TempR100, 54)
        ImportData("\\10.8.0.14\DB_Raw\100924-135506_NH4R300_2016.
            ↪ tsdb", Variables.MaxTimestampNH4R300, 55)
        ImportData("\\10.8.0.14\DB_Raw\100924-135506
            ↪ _NH4TempR300_2016.tsdb", Variables.
            ↪ MaxTimestampNH4TempR300, 56)
        ImportData("\\10.8.0.14\DB_Raw\100924-135506_NO3R300_2016.
            ↪ tsdb", Variables.MaxTimestampNO3R300, 57)
        ImportData("\\10.8.0.14\DB_Raw\100924-135506
            ↪ _NO3TempR300_2016.tsdb", Variables.
            ↪ MaxTimestampNO3TempR300, 58)
    End Sub
```

Le code se divise en trois grosses parties :

— L'initialisation des variables ;

— La fonction d'import et

— La fonction principale utilisant cette dernière.

La fonction d'import des données demande trois paramètres :

— Le chemin d'accès au fichier .tsdb ;

— La variable contenant le timestamp maximum de la dat*EAU* base pour ce capteur ; et

— L'identifiant de la métadonnée associée à la mesure.

La fonction vérifie si le chemin donné en paramètre donne bien accès à un fichier existant.
Si c'est le cas, une lecture binaire de ce fichier se fait en regroupant les bytes selon l'ordre indiqué (d'abord 2 Long de 8 bytes, ensuite 4 Bytes ensuite un Double). Ces valeurs sont sauvegardées à chaque itération dans les variables initialisées préalablement.
Le premier double est ensuite converti en Date puis en timestamp afin de pouvoir interpréter cette valeur ainsi que la comparer au timestamp max de ce capteur (entré en paramètre à la fonction) et à une valeur arbitraire permettant de vérifier qu'il s'agit bien d'une date réelle.
Enfin, on vérifie que l'on a bien une valeur de mesure (le fichier étant sans arrêt utilisé par BaseStation, il se peut que le programme soit en pleine écriture d'une ligne et que la valeur nous manque). Si c'est le cas, on ajoute une nouvelle ligne au flux contenant le nombre d'expériences, la métadonnée associée, le timestamp et la valeur de la mesure. Si celle-ci nécessite une conversion (comme pour les ID 49 ou 51 p.e.), celle-ci est effectuée (ici une division par 1000 pour passer de mg/L à kg/m^3).

Annexe H

Import depuis Ana : :Pro

H.1 Code

```
import os
import glob
import pymssql
import re
import time
import datetime

# Path to the directory with .par files (anaPro)
path = "//10.8.0.14/inflpc/"

# List the .par files
listFile = glob.glob(path+'*.par')

# Sort the list from the oldest to the last
listFile.sort

# Index of the oldest file with new data (will change during the
    ↪ execution but the last file always has new data)
index = len(listFile)-1

# Connect to the DB
conn = pymssql.connect(server = 'GCI-PR-DATEAU01\DATEAUBASE',
    ↪ database ='dateaubase')
cursor = conn.cursor()
```

```
# Get the timestamp of the last data inserted in the DB
cursor.execute('SELECT ISNULL(MAX(Timestamp),0) FROM dbo.value
    ↪ WHERE Metadata_ID=1')
row = cursor.fetchone()
lastTS = row[0]

# Convert the Timestamp in DateTime format
lastData = str(datetime.datetime.fromtimestamp(lastTS))

# loop to check all the files
for idx,file in enumerate(listFile):

        # Delete the path from the name
        actualFile = file.replace(path,'')

        # Get the date/time of the first data in the .par file
        dateName = re.split(r'[._-]+', actualFile)
        dateFile = (dateName[0] + '-' +dateName[1] + '-' +dateName
            ↪ [2] + ' ' +dateName[3] + ':' +dateName[4] + ':' +
            ↪ dateName[5] )

        # if a file is more recent than the last data => change the
            ↪  index to begin with the file just before this one
        if dateFile>lastData:
                index = max(idx-1,0)
                break

for file in listFile[index:]:
        # Open, save the content and close the file
        f = open(file,'r')
        lignes = f.readlines()[2:]
        f.close()

        # Find the highest ID in the Database to increment from it
        cursor.execute('SELECT ISNULL(MAX(VALUE_ID),0) FROM
            ↪ dateaubase.dbo.value')
        row = cursor.fetchone()
        lastID = row[0]
        i = 1
```

```
# For each line on the .par file, split the datas and get
    ↪ pertinent ones. If value = NaN : Value = 0
for ligne in lignes :
        datas = ligne.split("\t")
        # Split date and time
        datetime = datas[0].split("  ")
        # Format date to the DB format
        Date = datetime[0].replace(".","-")
        Time = datetime[1]
        # If last data in the DB oldest than the the actual
            ↪ treated data : insert it
        DateTime = Date + " " + Time
        UTCTimestamp=time.mktime(time.strptime(DateTime,"%Y
            ↪ -%m-%d %H:%M:%S"))
        if lastData<DateTime:
                TSS = datas[2]
                if TSS=="NaN":
                        TSS = 0
                NO3N = datas[4]
                if NO3N=="NaN":
                        NO3N = 0
                COD = datas[6]
                if COD=="NaN":
                        COD = 0
                CODf = datas[8]
                if CODf=="NaN":
                        CODf = 0
                NH4N = datas[10]
                if NH4N=="NaN":
                        NH4N = 0
                K = datas[12]
                if K=="NaN":
                        K = 0
                pH = datas[14]
                if pH=="NaN":
                        pH = 0
                Temp = datas[16]
                if Temp=="NaN":
```

```
                    Temp = 0

                # Increment the ID
                ID = lastID + i

                # Insert the datas in the DB
                cursor.executemany(
                        "INSERT INTO dbo.value(Value_ID,
                            ↪ Timestamp,Value,
                            ↪ Number_of_experiment,
                            ↪ Metadata_ID) VALUES (%d,%d,%d
                            ↪ ,%d,%d)",
                        [(ID, int(UTCTimestamp), float(TSS
                            ↪ /1000), 1,5),
                        (ID+1, int(UTCTimestamp), float(
                            ↪ NO3N/1000), 1,6),
                        (ID+2, int(UTCTimestamp), float(COD
                            ↪ /1000), 1,7),
                        (ID+3, int(UTCTimestamp), float(
                            ↪ CODf/1000), 1,8),
                        (ID+4, int(UTCTimestamp), float(
                            ↪ NH4N), 1,1),
                        (ID+5, int(UTCTimestamp), float(K),
                            ↪  1,2),
                        (ID+6, int(UTCTimestamp), float(pH)
                            ↪ , 1,3),
                        (ID+7, int(UTCTimestamp), float(
                            ↪ Temp), 1,4)])
                        conn.commit()
                i=i+8
# Close the connexion to the DB
conn.close()
```

Pour l'explication générale et la philosophie du code : voir chapitre 5. Les détails sont développés ci-dessous.

Tout d'abord, on importe les packages nécessaires pour le script. Leur utilité a été développée au chapitre 2.
Ensuite, on instancie le chemin d'accès vers le dossier contenant les fichiers .par, on en dégage une liste des fichiers présents sous cette extension et on trie cette liste.
On crée notre index que l'on déclare valoir la taille de notre liste - 1 afin de pointer le dernier élément de la liste (le premier élément valant 0, il faut décrémenter la longueur de cette liste de 1).
On se connecte à la datEAUbase et l'on crée le curseur (objet permettant de parcourir la BDD).
On sélectionne la plus haute valeur de timestamp d'Ana::Pro dans la base de données, on l'enregistre dans la variable "lastTS" que l'on convertit en DateTime (lastData).
On parcourt chaque fichier de la liste et l'on formate son nom afin d'en récupérer la date de création (celle-ci étant sous forme AAAA-MM-JJ_HH-MM-SS.par, il est aisé d'en récupérer l'année, le mois, le jour, l'heure, les minutes et les secondes).
Dès que l'on trouve un fichier dont la date de création est plus récente que la dernière donnée de la BDD, on modifie l'index à X-1 (voir section 5.2) et l'on sort de la boucle (on arrête de parcourir chaque fichier).
On parcourt maintenant chaque fichier à partir de cet index. Pour chacun d'eux, on lit le fichier (excepté les 2 premières lignes non pertinentes) et le sauvegarde en mémoire.
On recherche ensuite le dernier ID de la BDD afin de pouvoir incrémenter automatiquement à partir de cet identifiant.
Enfin, chaque ligne est formatée afin d'en dégager les informations importantes (date, heure et différentes valeurs des mesures), chaque valeur est convertie (soit de date et heure vers timestamp, soit d'unité fonctionnelle vers unité SI) et l'on sauvegarde ces informations dans la BDD.

Bibliographie

Anonymous. How to schedule ssis package to run as something other than sql agent service account. 2016. URL http://stackoverflow.com/a/6717195.

P. Vanrolleghem. Presentation. pileaute : modeleau's new pilot plant. 08 2015.

Q. Plana. Automated data collection and management at enhanced lagoons for wastewater treatment. 2013. URL http://modeleau.fsg.ulaval.ca/no_cache/publications/.

Q. Plana. Efficient on-line monitoring of river water quality using automated measuring stations. master's thesis, universitat politecnica de catalunya. barcelona, spain. 2015. URL http://modeleau.fsg.ulaval.ca/no_cache/publications/.

Microsoft. Mysql to sql server migration : How to use ssma. 2016. URL https://blogs.msdn.microsoft.com/ssma/2011/02/07/mysql-to-sql-server-migration-how-to-use-ssma.

L. Rieger , P. Vanrolleghem. moneau : a platform for water quality monitoring networks. *Water Science and Technology*, 57(7) :1079–1086, 2008. URL http://modeleau.fsg.ulaval.ca/fileadmin/modeleau/documents/Publications/pvr765.pdf.

Washington State Department Of Transportation. Water quality monitoring database user's guide. 2008.

D. Camhy, V. Gamerith, D. Steffelbauer, D. Muschalla, G. Gruber. Scientific data management with open source tools – an urban drainage example. *Proceedings IWA/IAHR 9th International Conference on Urban Drainage Modelling, Belgrade, Serbia, September 4-6*, 2012.

M.E. Holmes and G.C. Poole. Management of a long-term water quality database : Flatdat for the flathead lake biological station. 1 :111, 1996.

Hydrolab. 2016. URL http://www.ott.com/download/hydrolab-metadata-article/.

Janelcy Alferes, John Copp, Peter A. Vanrolleghem, Stefan Weijers. Innovative water quality monitoring : Automation of data assessment in practical scenarios. *Proceedings IWA World Water Congress 2014*, 2014. URL http://modeleau.fsg.ulaval.ca/fileadmin/modeleau/documents/Publications/pvr1138.pdf.

Y. Amerlinck, L. Benedetti, J.J.M. De Klein, T. Flameling, G.L. Langeveld, I. Nopens, A. Van Nieuwenhuijzen, O. Van Zanten, S. Weijers. Impact based integrated real time control for improvement of the dommel river water quality. *Urban Water Journal*, 2013.

M. Ecker , N. Kreuzinger, A. Pressl, N. Fleischmann, N. Gruber, and S. Winkler. Innovative technology for integrated water quality measurement. *Proceedings International Conference on Automation in Water Quality Monitoring*, 2002.

Ecole Nationale Supérieure des Mines de Saint-Etienne. La décantation. 2016. URL http://www.emse.fr/~brodhag/TRAITEME/fich4_4.htm.

G. Aubry. Enlèvement de l'azote des eaux usées par un procédé à culture fixée immergée. 2003. URL http://theses.ulaval.ca/archimede/fichiers/21279/ch01.html.

ToutWindows. Windows server 2008 r2 - installation. 2016. URL http://www.toutwindows.com/ws_sommaire.shtml.

Debian Handbook. Réseau privé virtuel. 2016. URL https://debian-handbook.info/browse/fr-FR/stable/sect.virtual-private-network.html.