



RÉVÉLATEUR D'INGÉNIEUR·E·S
DEPUIS 60 ANS

PRODITEC

Stage Assistant Ingénieur 2023/2024

Mise en place d'un serveur OPC UA


Z.I. PESSAC MAGELLAN
3, rue F. Chevreul
33600 PESSAC FRANCE
Tél. +33 557 891 720
Fax +33 557 891 060

Du 4/9/2023 au 4/1/2024

Tuteur entreprise : PERUZZETTO Enzo

Tuteur académique : NASREDDINE Kamal

Stagiaire :
ABDEL KADER
Omar

Remerciements

Je tiens à exprimer ma gratitude envers ceux qui ont contribué à la réussite de mon stage. En premier lieu, je souhaite remercier chaleureusement monsieur **Enzo PERUZZETTO**, mon tuteur tout au long de ma période de stage, ainsi que monsieur **Benjamin VIQUERAT**, chef de projet au sein de l'équipe R&D, et les autres membres de cette équipe, qui ont grandement facilité l'obtention de mon stage au sein de l'entreprise Proditec.

Je suis reconnaissant envers Monsieur **Christophe RIBOULET**, président de l'entreprise, pour la confiance qu'il m'a accordée tout au long de cette expérience professionnelle. Un sincère remerciement à Monsieur **Kamal NASREDDINE**, enseignant et directeur du département d'électronique à l'ENIB, pour son soutien constant et son encadrement attentif tout au long de mon stage en tant qu'assistant ingénieur. Sa collaboration a été d'une aide précieuse dans l'élaboration de ce rapport de stage.

Résumé/Abstract

Français

Au cours de mon stage de 4 mois en tant qu'assistant ingénieur chez Proditec, j'ai considérablement renforcé mes compétences à différents niveaux. Ce stage a été une opportunité cruciale pour mettre en pratique les connaissances acquises à l'école et approfondir mon expertise dans le domaine du génie logiciel. J'ai particulièrement enrichi mes compétences en programmation, notamment en C++/Qt, et j'ai utilisé plusieurs environnements de développement intégré tels que Visual Studio et Qt Creator. De plus, j'ai acquis une compréhension approfondie du protocole OPC UA et de ses fonctionnalités. Ce stage m'a aussi offert l'opportunité de renforcer mes compétences en résolution de problèmes, en communication, en collaboration d'équipe et en gestion de projet. Grâce à l'ambiance de travail dans l'entreprise, je me suis rapidement senti à l'aise. Cette ambiance favorable a facilité mon intégration totale, simplifiant ainsi la collaboration et l'entraide dans la réalisation des différents projets.

Proditec, entreprise spécialisée dans la fabrication de machines de tri pour gélules et pièces de monnaie, envisage d'intégrer un serveur OPC UA dans ses machines. Mon stage s'est déroulé en trois étapes. D'abord, le développement d'un serveur OPC UA à l'aide d'une SDK. Ensuite, la conception d'une Interface Homme-Machine (IHM) facilitant la configuration du serveur. Enfin, la création d'une bibliothèque dynamique (DLL) assurant la liaison efficace entre la machine et le serveur.

Je ressens une profonde gratitude envers mon maître de stage ainsi que tous les membres de l'entreprise pour la confiance qu'ils m'ont témoignée et le soutien qu'ils m'ont généreusement offert tout au long de mon stage. Cette expérience enrichissante m'a permis d'acquérir de précieuses connaissances et a orienté ma trajectoire vers le domaine de l'ingénierie.

English

During my 4-month internship as an assistant engineer at Proditec, I significantly enhanced my skills on various fronts. This internship was a crucial opportunity to apply the knowledge gained in school and deepen my expertise in software engineering. I particularly honed my programming skills, especially in C++/Qt, and utilized various integrated development environments such as Visual Studio and Qt Creator. Additionally, I gained a thorough understanding of the OPC UA protocol and its functionalities.

This internship also provided me with the opportunity to strengthen my problem-solving, communication, teamwork, and project management skills. Thanks to the positive work environment within the company, I quickly felt comfortable. This conducive atmosphere facilitated my seamless integration, simplifying collaboration and mutual assistance in the completion of various projects.

Proditec, a company specializing in the manufacturing of sorting machines for capsules and coins, is considering integrating an OPC UA server into its machines. My internship unfolded in three stages: firstly, the development of an OPC UA server using an SDK; secondly, the design of a Human-Machine Interface (HMI) to streamline server configuration; and finally, the creation of a dynamic link library (DLL) ensuring efficient communication between the machine and the server.

I am deeply grateful to my internship supervisor and all members of the company for the trust they placed in me and the support they generously provided throughout my internship. This enriching experience has equipped me with valuable knowledge and steered my path towards the field of engineering.

Table des matières

Remerciements	2
Résumé/Abstract	3
Français	3
English	4
Table des matières	5
Table de figures	6
Glossaire	7
Proditec	8
Présentation	8
L'histoire de l'entreprise	8
Ses produits	9
L'équipe	10
Organisation du temps	11
OPC	11
OPC Classique	12
OPC Data Access	12
OPC Alarm & Events	12
OPC Historical Data Access	13
OPC UA	13
Technologie DCOM	13
Dépendance à la Plate-forme Windows	13
Indépendance des protocoles	14
Problème de sécurité	14
Sujet du Stage	16
Serveur OPC UA	16
Etude de marché	16
QUaServer	17
Programmation du serveur	19
Fonctionnement du serveur	19
Problèmes / Solutions	26
Code	28
Installeur	30
IHM de configuration	34
Fonctionnement de l'IHM	34
Code	37
ModuleOPC (DLL)	39
Bilan du stage	42
Bibliographie/Webographie	43
Diagramme de Gantt	44

Table de figures

Figure 1: Logo de Proditec.....	8
Figure 2 Première machine pharmaceutique créée par Proditec	9
Figure 3 VISITAB_2L.....	10
Figure 4 Architecture de l'OPC (Client - Serveur)	11
Figure 5 Différence entre OPC Classique et OPC UA	14
Figure 6 Etapes de génération de open62541.lib	19
Figure 7 Fonctionnement de Exemples.pro	19
Figure 8 Configure l'environnement pour le compilateur C++	20
Figure 9 Exécution du serveur OPC.....	20
Figure 10 Insertion de l'URL dans UAExpert	20
Figure 11 Exploration de l'espace d'adressage du serveur.....	20
Figure 12 Accéder aux données du serveur.....	21
Figure 13 Déclenchement des alarmes	21
Figure 14 Appel de la fonction "triggerServerEvent"	22
Figure 15 Accéder aux événements envoyer	22
Figure 16 La valeur de New_Tag5 historisée dans la base de données.....	22
Figure 17 La valeur de New_Tag6 historisée dans la base de données.....	22
Figure 18 L'événement historisé dans la base de données.	22
Figure 19 Appel de la méthode "ClearDataBase"	23
Figure 20 BrowseName de "New_Tag5"	23
Figure 21 Appel de la méthode "ChangeDisplayName"	23
Figure 22 Résultats de l'appel de la méthode "ChangeDisplayName"	24
Figure 23 Le répertoire des certificats créé	25
Figure 24 Le certificat	25
Figure 25 Les mode de chiffrement valable dans le serveur	25
Figure 26 Propriétés de l'arrêt du system	26
Figure 27 Le nouveau fichier de configuration XML.....	27
Figure 28 UML du code du serveur.....	28
Figure 29 Fichier principale du serveur	29
Figure 30 choisir la langue de l'installateur.....	30
Figure 31 Choisir la machine	30
Figure 32 Choisir la destination.....	31
Figure 33 Partie du script de l'installateur.....	31
Figure 34 Code du fichier batch	32
Figure 35 Code du fichier Visual Basic.....	32
Figure 36 Schéma du fonctionnement de l'installateur.....	33
Figure 37 Premier widget de l'IHM (Server Settings)	34
Figure 38 Deuxième widget de l'IHM (Tags Settings)	35
Figure 39 Fenêtre de création des tags.....	35
Figure 40 Fenêtre de modification des tags.....	36
Figure 41 Troisième widget de l'IHM (Save Into XML)	36
Figure 42 UML du code de l'IHM.....	37
Figure 43 Fonction "Save" du premier widget.....	38
Figure 44 Illustration du LevelAlarmVector.....	39
Figure 45 UML du code de la bibliothèque dynamique (DLL).....	40
Figure 46 Appel de la Methode1	41
Figure 47 Diagramme de Gantt.....	44

Glossaire

A

AE: Alarms and Events

API: Application Programmable Interface

C

COM/DCOM: Component Object Model / Distributed Component Object Model

D

DA: Data Access

DLL: Dynamic-link Library

H

HDA: Historical Data Access

HTTP/HTTPS: Hypertext Transfer Protocol / Hypertext Transfer Protocol Secure

M

MSVC: Microsoft Visual C++

O

OPC: Open Platform Communication

OS: Operating System

S

SDK: Software Development Kit

T

TCP: Transmission Control Protocol

U

UA: Unified Architecture

URL: Uniform Resource Locator

Proditec

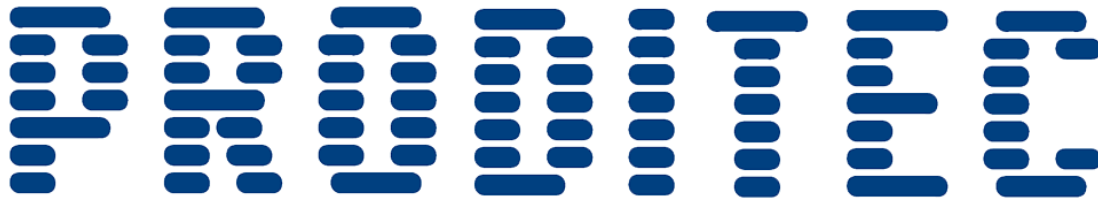


Figure 1: Logo de Proditec

Présentation

Proditec est une PME (petite à moyenne entreprise) française, basé à Pessac, qui fabrique des machines d'inspection automatique. Elle propose une large gamme de machines de tri de haute performance. Bien que l'essentiel de son activité se trouve dans le secteur pharmaceutique (qui constitue environ 90 % de son chiffre d'affaires), elle conçoit également des machines d'inspection de pièces de monnaie. Proditec est une entreprise d'une grande envergure, ayant une forte activité à l'international. Cela fait 30 ans que Proditec est leader européen sur le marché de fabrication de machines de tri pour diverses applications : le contrôle de comprimés liés à l'industrie pharmaceutique, mais également le contrôle de pièces de monnaies. En effet, 95% de son chiffre d'affaires est réalisé à l'étranger. Pour répondre aux demandes à l'international, Proditec s'est ensuite implantée aux États-Unis, à Singapour et en Inde. Sa présence en Asie, en Europe et en Amérique du Nord lui permet d'être à proximité de la plupart de ses clients. Cela favorise ainsi son activité commerciale, et simplifie la gestion du service après-vente.

L'histoire de l'entreprise

En 1987, M. Philippe RIBOULET crée l'entreprise Proditec, afin de proposer des services aux industriels locaux (ingénierie, robotique, pilotage de procédés, techniques de visionnage). En 1991, dans le cadre d'un projet avec « la monnaie de Pessac », Proditec conçoit sa première machine, la VISIA 100, pour répondre à une demande de tri des flans monétaires. Elle se spécialise ensuite dans ce domaine, qui sera son activité principale jusqu'en 1994. Aujourd'hui, plus de 150 machines de ce type sont utilisées dans le monde. En 1994, dans le cadre d'un projet avec SANOFI-Ambarès, Proditec développe la première machine de tri de comprimés « VISITAB ». C'est ainsi que Proditec commencera son activité dans l'industrie pharmaceutique.



Figure 2 Première machine pharmaceutique créée par Proditec

En 2005, M. Christophe RIBOULET (fils de Philippe RIBOULET) rachète l'entreprise de son père, et développe l'activité pharmaceutique de l'entreprise aux États-Unis, ainsi qu'en Asie. À partir de 2005, Proditec propose une large gamme de machines, et devient l'un des plus grands acteurs sur le marché des machines de tri de comprimés et de gélules. Aujourd'hui, le chiffre d'affaires de l'entreprise provient principalement du secteur pharmaceutique, grâce aux différentes machines VISITAB (nom de la gamme conçue pour l'industrie pharmaceutique). C'est un secteur dans lequel l'entreprise emploie plus d'une cinquantaine de personnes.

Ses produits

Du côté pharmaceutique :

- **VISITAB_3M** (ou simplement VT3M) est considérée comme étant la machine haute gamme de Proditec. Cette machine possède deux files de tri et elle est la machine la plus performante fabriquée par Proditec aujourd'hui. Cette machine utilise des postes de vision dits Kheops ou Khephren. Les deux postes Kheops et Khephren utilisent que des caméras matricielles (D'où le M dans le nom). Chaque poste possède 5 caméras qui vont permettre de prendre des images représentant les 5 vues (vues avant, arrière, gauche, droite, et centre) de l'objet. Le poste Kheops (Khephren) ne peut être utilisé que pour les comprimés (les gélules). L'atout et le point fort qui rend cette machine si performante est l'utilisation d'un convoyeur aspirant qui permet de monter en vitesse sans trop déstabiliser les objets. La VT3M peut atteindre une vitesse de 1m/s. Ainsi, en considérant un comprimé de 5mm, la machine est capable de trier plus de 700000 unités par heure.

- **VISITAB_3MH** (ou simplement VT3MH) est la version hybride de la VT3M qui va permettre de faire le tri des gélules et des comprimés avec une seule machine. C'est à dire, qu'elle contient simultanément les 2 postes Kheops et Khephren.
- **VISITAB_2M** (ou simplement VT2M) Cette machine est la machine soeur de la VT3M. La seule différence est que la VT2M utilise plutôt des convoyeurs normaux avec des roues libres pour retourner les objets. Le 2 présent dans le nom de cette machine signifie que cette machine est plus lente que la VT3M.
- **VISITAB_2L** (ou simplement VT2L) est aussi une machine avec 2 files mais qui n'utilise que des caméras linéaires (d'où le L dans la nomination) (Figure 3).

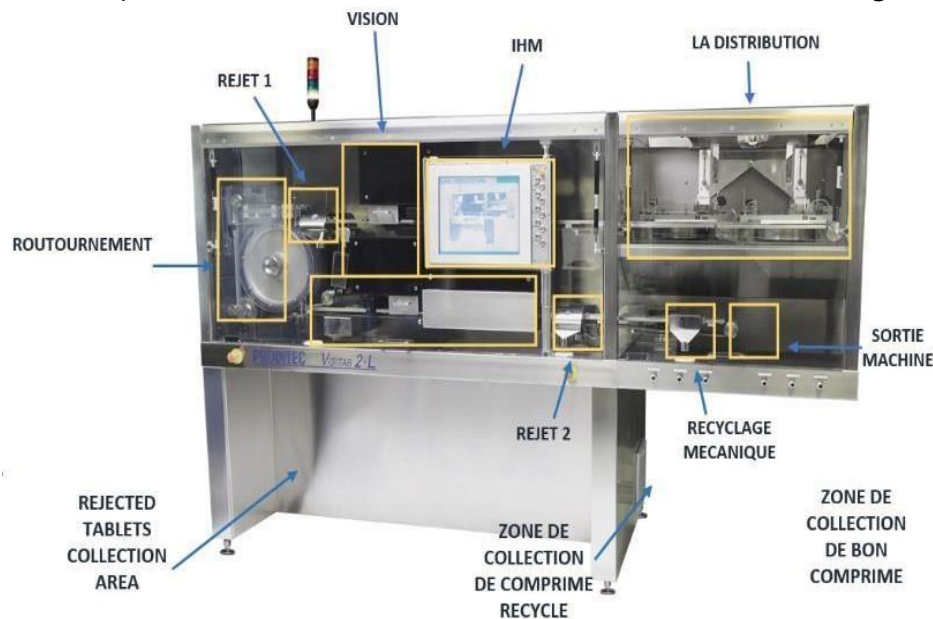


Figure 3 VISITAB_2L

Du côté monnaie :

- **Visia** : Contrôler la production monétaire en fin de production à haute cadence.
- **VisiCoin** : Contrôler les pièces fournies à 100% ou en reprise à haute cadence.
- **VisiFast** : Contrôler les couronnes à haute cadence.
- **Visilab** : Mesurer à haute précision les pièces et les flans.

L'équipe

À Proditec, on y trouve plusieurs départements majeurs :

- Un service recherche et développement (R&D) qui s'occupe du développement Vision et de la recherche de produits et technologies.
- Un service production qui réalise le montage, l'installation et le réglage des machines.
- Un service commercial qui s'occupe de la vente, des négociations avec les clients et du marketing.
- Un service après-vente (SAV) qui fait office de Hotline et dépanne les clients en cas de problèmes.

J'ai réalisé mon stage au sein du service R&D composé d'un chef de projet et de 4 ingénieurs. Dans ce département, on peut distinguer deux équipes différentes :

- Une équipe Mécatronique qui s'occupe de la partie mécanique et électrique des machines.
- Une équipe Vision qui s'occupe de tout ce qui est relation avec les images. Cela comprend la qualité de la prise d'image, l'éclairage, l'acquisition des données, les algorithmes de traitement ...

Organisation du temps

Pour améliorer mon rendement et éviter les pertes de temps lors de l'accomplissement des tâches qui m'ont été confiées au sein de l'entreprise, notre chef de projet élaborait un plan pour chaque ingénieur de notre équipe afin de suivre l'avancement individuel. Il m'a attribué des tâches spécifiques à accomplir chaque semaine, et tous les lundis matin, l'équipe se réunissait pour faire le point sur l'avancement de nos projets respectifs, y compris le mien. Vous pouvez retrouver les tâches que j'ai réalisées chaque semaine dans le diagramme de Gantt (Figure 47).

OPC

L'OPC, créé en 1996 par "OPC Foundation", est la norme d'interopérabilité pour l'échange sécurisé et fiable de données dans le domaine de l'automatisation industrielle et dans d'autres industries. Il est indépendant de la plate-forme et garantit un flux transparent d'informations entre les appareils de plusieurs fournisseurs [4]. Son objectif principal est de fournir aux clients toutes les informations qui sont liées à une machine. OPC est basée sur l'architecture client-serveur comme le montre la Figure 4.

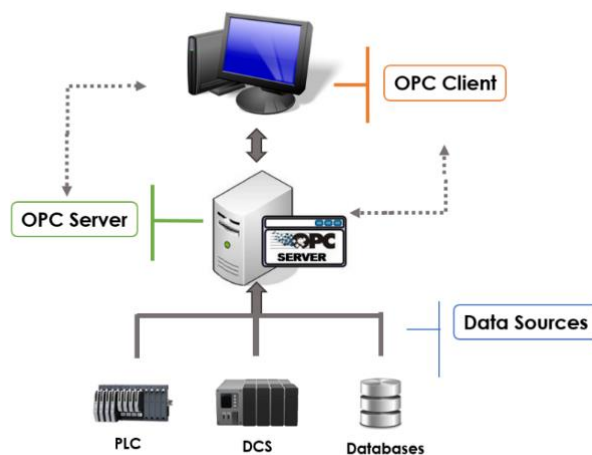


Figure 4 Architecture de l'OPC (Client - Serveur)

Notion de Tag : Un tag représente une variable associée à une information spécifique d'une machine (le nombre de comprimés bons et mauvais triés, les noms des utilisateurs qui sont connectés à l'OPC...) et peut prendre n'importe quel type de données (booléen, entier, chaîne de caractères...)

Notion d'Alarmes : Les Alarmes sont des Tags spéciaux qui déclenchent une alerte chez le logiciel du client lorsque la valeur associée à ce Tag atteint une valeur précise. Par exemple, si la valeur d'un Tag lié à un capteur de température dépasse 60 degrés, une alarme sera déclenchée dans le logiciel du client (le logiciel utilisé par les clients est souvent :

UAEExpert).

Notion Événements : Les Événements sont des messages envoyés de la machine au client, par exemple, si la porte de la machine est ouverte.

On peut observer deux types d'OPC : l'OPC Classique et l'OPC UA.

OPC Classique

L'OPC classique est la première norme utilisée par l'industrie et inclut différentes spécifications et protocoles :

OPC Data Access ou DA : l'accès aux données en temps réel

OPC Alarm and Events ou AE : la gestion des alarmes et événements

OPC Historical Data Access ou HDA : la construction d'historique de données [6]

OPC Data Access

Un serveur OPC Data Access a pour fonction de :

- Collecter les données issues des périphériques matériels (lecture) ou leurs faire parvenir les mises à jour de données (écriture).
 - Assurer l'intégrité des données (qualité et fraîcheur)
 - Répondre aux requêtes de clients (fournir les données, écrire vers le matériel...)
 - Avertir les clients abonnés des changements d'état des variables par un événement
 - Informer le client des problèmes de sécurité (dysfonctionnement logiciel ou matériel)
- [6]

OPC Alarm & Events

Le deuxième protocole à être ajouté à la norme OPC était Alarmes et événements. Il permet comme son nom l'indique d'échanger des alarmes et événements entre le périphérique et le client OPC. Ce protocole est fondamentalement différent du protocole Data Access simplement parce que les événements n'ont pas de valeur. Cela signifie que ce protocole est toujours basé sur un abonnement où les clients obtiennent tous les événements entrants qu'ils veulent. En termes d'informations fournies avec l'événement, il n'y a pas de nom ni de qualité, mais bien sûr, un horodatage. [6]

OPC Historical Data Access

La différence entre DA, AE et HDA réside dans le fait que HDA contient des données historisées et que vous pouvez faire appel à une grande quantité de valeurs antérieures d'une donnée. Le protocole prend donc en charge l'enregistrement d'informations pour une ou plusieurs données. [6]

OPC UA

Malgré ses attraits, OPC en sa version classique et sa spécification principale OPC DA se devaient d'évoluer pour suivre les progrès technologiques et s'adapter aux réalités du terrain. La définition des communications et des interactions entre les périphériques n'est pas restée figée depuis la création du standard. La fondation OPC avait également identifié les problèmes de la version classique de la norme. Un groupe de travail créé en 2003 a été chargé de la définition de la nouvelle norme appelé OPC UA (Unified Architecture). Dans les sous-sections ci-dessous, les principaux problèmes résolus par la nouvelle norme. [6]

Technologie DCOM

DCOM a atteint ses limites dans l'industrie. Son principal défaut tient dans le fait que la technologie DCOM n'autorise pas le franchissement des firewalls. C'est pourquoi OPC était réservé jusqu'alors aux seuls réseaux locaux. Aujourd'hui, il est impératif qu'une application puisse traverser les firewalls de l'entreprise, d'autant plus que les usines multi-site sont de plus en plus nombreuses. Les industriels ont besoin de pouvoir surveiller des productions partout dans le monde. D'autant plus que Microsoft s'est désengagé des technologies COM et DCOM, au profit des Web Services. C'est donc l'essence même d'OPC qui n'est plus maintenu.

Dépendance à la Plate-forme Windows

Les membres de la fondation OPC souhaitaient que le format d'échange de données fonctionne indépendamment de la plate-forme sur laquelle l'application est installée. Leur travail a concerné l'échange des données d'un logiciel à un autre comme pour l'OPC classique, mais aussi d'un système d'exploitation à un autre. Auparavant, seuls des PC sous Windows pouvaient devenir des serveurs OPC, puisque ces derniers étaient écrits avec les technologies COM et DCOM, spécifiques à Microsoft. L'objectif avec une architecture OPC

UA, est que clients et serveurs pourront être installés indifféremment sur des systèmes d'exploitation Windows classiques, mais aussi des OS libres (comme Linux), pour terminaux portables (comme Windows CE) ou même temps réel (comme VxWorks) et dialoguer sans se préoccuper du matériel sur lequel ils sont déployés. Il fallait donc en finir avec DCOM pour

passer à un fonctionnement plus performant et plus facile à migrer sur des plates-formes différentes comme le montre la Figure 5. [6]

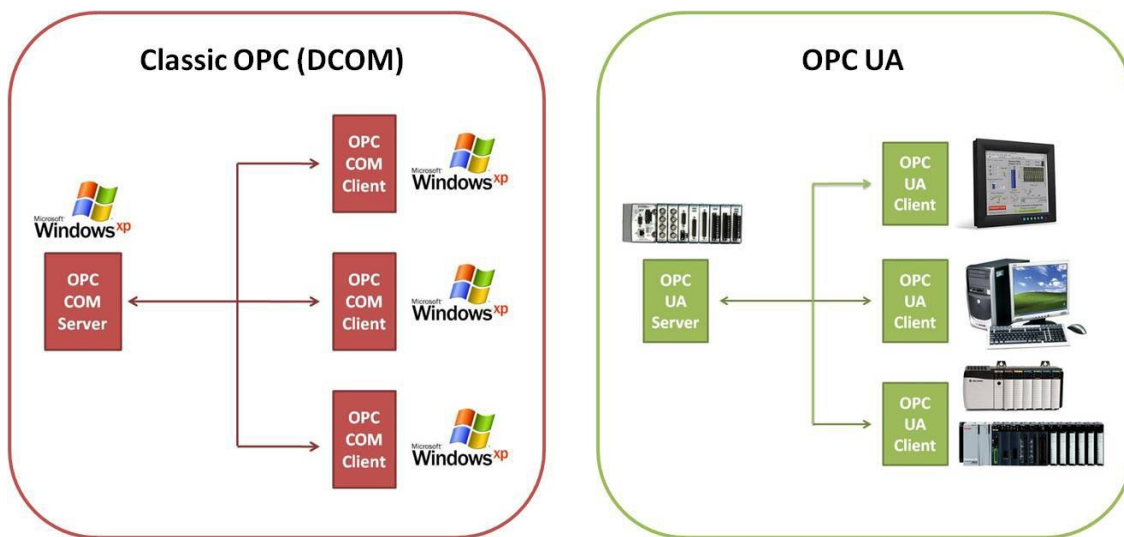


Figure 5 Différence entre OPC Classique et OPC UA

Indépendance des protocoles

OPC dans sa version classique se composait d'un ensemble de protocoles et de fonctions dissociées les unes des autres et proposées dans des modules différents (DA, AE, HDA, ...), il était ainsi impossible pour un client d'avoir accès à un autre serveur relié à un autre client. Les informations ne pouvaient pas être croisées facilement. Prenons l'exemple d'une application consistant à surveiller une cuve : imaginons qu'elle a un serveur AE remontant l'information du capteur " <alerte niveau bas>" et relié à un client dédié à la gestion des alarmes, et qu'elle a aussi un serveur DA remontant l'information du capteur "niveau de la cuve" qui est relié à un client qui concentre les valeurs de niveaux de toutes les cuves. Avec OPC classique, il était impossible pour l'opérateur surveillant les alarmes d'accéder au niveau réel de la cuve lorsque l'alarme se déclenche. Dans le cas d'OPC UA, un modèle regroupant plusieurs TAG peut être créé et correspondra à la cuve dans sa globalité. Ainsi, en se connectant au serveur lié à la cuve l'opérateur sera capable d'accéder à toutes les informations la concernant grâce à la mutualisation de tous les protocoles dans l'OPC UA d'où le UA pour UNIFIED ARCHITECTURE. [6]

Problème de sécurité

OPC UA apporte aussi de la sécurité au transfert d'informations distantes qui pouvait poser des problèmes avec les technologies COM/DCOM puisque les firewalls devaient être ouverts pour permettre un échange multi-site ce qui ouvrait des portes pour du piratage sur le réseau de l'usine. En effet, la technologie Web Services utilisée par l'UA pour remplacer ces composants intègre la sécurité de manière native et elle est déjà éprouvée à la base de la plupart des applications multi-site actuelles. En OPC UA, la communication entre serveur et

client est basée sur le protocole TCP/IP standard en utilisant SSL, HTTP ou HTTPS. Deux couches ont été implémenté au-dessus des couches de transport TCP/IP standard, une qui gère la session et une qui établit un canal sécurisé entre le client et le serveur. La communication en OPC UA sécurise le canal de communication non seulement en cryptant les données, mais également en sécurisant l'authentification, de sorte que les points finaux ne puissent pas être infiltrés et modifiés. Ceci est basé sur un système de certificats (X.509) de confiance partagée entre le serveur et le client. [6]

Sujet du Stage





Les machines d'inspection automatiques développées par Proditec sont essentiellement pilotées par l'IHM (ProdiSoft), une interface utilisateur qui offre aux utilisateurs la possibilité de lancer des productions, d'obtenir des informations en temps réel sur l'état des machines, ainsi que d'accéder aux recettes nécessaires pour garantir la qualité et l'efficacité des opérations. Cette interface avait été conçue » à l'origine en utilisant LabView, un environnement de développement bien établi dans l'industrie. Cependant, plusieurs facteurs critiques ont motivé une réflexion approfondie et finalement la décision de migrer vers une nouvelle technologie, en l'occurrence Qt. Les limitations de LabVIEW en ce qui concerne l'intégration avec les dernières technologies, ainsi que son incapacité à s'intégrer de manière transparente dans les réseaux existants des clients, a conduit l'équipe Vision de Proditec à prendre la décision stratégique de migrer vers Qt. Qt est reconnu pour être une technologie moderne et polyvalente offrant une prise en charge native du langage C++ et des capacités de développement d'interfaces utilisateur fluides, performantes et intuitives. Le protocole OPC actuellement utilisé dans leurs machines est l'**OPC DA**, ce qui signifie que les clients n'ont accès qu'aux données du serveur, excluant les alarmes et événements (OPC AE) ainsi que l'historisation (OPC HDA), ce serveur est fourni par l'entreprise Matrikon. Pendant cette transition significative des composants informatiques des machines, et en vue de satisfaire les exigences évolutives des clients, j'avais pour responsabilité de déployer un serveur OPC UA complet, intégrant l'ensemble de ses fonctionnalités.

Serveur OPC UA

Etude de marché

Chez Proditec, j'ai commencé mon stage en étudiant l'OPC UA et ses fonctionnalités et aussi le serveur DA qui est installé chez Proditec et son architecture. Ensuite, j'ai réalisé une étude de marché sur les serveurs disponibles afin de déterminer lequel serait le plus adapté à nos besoins. Après 2 semaines de recherche, j'ai identifié quatre serveurs, dont deux sont prêts à être déployés, tandis que les deux autres sont des SDK nécessitant un développement ultérieur.

Pour choisir le serveur sur lequel nous allons travailler, j'ai effectué des tests sur chacun d'eux, notant les avantages et les inconvénients de chacun pour les présenter à l'équipe comme le montre le tableau ci-dessous.

	Avantages	Inconvénients
 KEP IoT KEPServerEX6 (Fourni par KEPFrance)	- Serveur Professionnelle contenant tous les fonctionnalités nécessaires - Entreprise française fournissant un bon service au client	- L'exploitation du serveur engendrera des coûts, comprenant des frais annuels de 1788 euros pour les drivers Modbus ainsi qu'un contrat de maintenance annuel obligatoire s'élevant à 255 euros. En activant l'historisation et les alarmes, les coûts augmenteront, pouvant atteindre jusqu'à 12000 euros.
 EMERSON Connex 4.2 Editor (Fourni par Emerson)	- Serveur Professionnelle contenant tous les fonctionnalités nécessaires	- Le coût est déterminé en fonction du nombre de tags (des frais s'appliqueront au-delà de 100 tags), et je n'ai pas réussi à obtenir son certificat pour qu'il soit considéré comme "fiable". - Je n'ai pas pu le tester correctement en raison de l'absence de ce certificat. - L'équipe en question n'a pas répondu à mes e-mails ni à mes appels téléphoniques.
 S² OPC safe & secure S2OPC UA (SDK) (Fourni par S2OPC)	- Serveur gratuit	- Ecrit en langage C - J'ai pu compiler le code mais le fichier exécutable du serveur n'a pas été généré.
 QUaServer (SDK) (Open source)	- Serveur Gratuit - Implémenté en C++/Qt - Contient toutes les fonctionnalités	- Il nécessite une version de Qt qui dépasse la 6, alors que l'équipe chez Proditec travaille actuellement avec la version 5.9.9.

Après la présentation de ces quatre serveurs à l'équipe, nous avons conclu que le serveur **QUaServer** répond le mieux à nos besoins par rapport aux autres serveurs disponibles sur le marché.

QUaServer

QUaServer est une bibliothèque basée sur Qt qui fournit un "Wrapper" C++ pour la bibliothèque "open62541". Cette dernière, développée par la "OPC Foundation", est considérée comme la bibliothèque principale de tous les serveurs OPC. "QUaServer" fournit également une abstraction pour l'API du serveur OPC UA. L'objectif principal de cette bibliothèque est de simplifier le développement en fournissant une API orientée objet, évitant ainsi l'utilisation directe de la bibliothèque "open62541", qui peut demander beaucoup de temps à maîtriser.

Cette SDK requiert l'installation de : Visual Studio 17 2022 Community avec le compilateur MSVC 2019 64-bit, Qt 6.5.2, SQLite et MySQL2.

Le logiciel utilisé comme serveur client pour tester mon serveur est « **UAExpert** ».

Le projet inclut une explication sur plusieurs parties. Les parties qui nous intéressent sont :

- **Inclure** : Comment inclure le "QUaServer" dans notre projet.
- **Basiques** : Comment créer des Tags dans le serveur
- **Méthodes** : Comment créer des méthodes accessibles par les clients pour les appeler
- **Server** : Comment personnaliser le serveur et créer des certificats
- **Utilisateurs** : Comment créer des comptes pour les clients et modifier leurs autorisations sur le serveur (en lecture et écriture)
- **Chiffrement** : Comment activer le chiffrement entre le serveur et le client
- **Événements (OPC AE)** : Comment envoyer des événements aux clients.
- **Sérialisation** : Comment sauvegarder les tags créés sur le serveur dans un fichier XML et les récupérer.
- **Historisation (OPC HDA)** : Comment historiser et stocker des données de tags dans une base de données "SQLite".
- **Alarmes (OPC AE)** : Comment créer des alarmes dans un serveur. Dans "QUaServer", on peut trouver deux types d'alarmes : "Normal Alarms" et "Level Alarms". Le "Normal Alarm", est un tag de type booléen qui déclenche une alarme si le tag est "true" et non si c'est "false". Le "Level Alarm" est un tag de type entier ou décimal qui déclenche une alarme si le tag dépasse un seuil prédéfini.

En fait, il est possible de distinguer quatre types de tags dans le "QUaServer" :

- *Properties* : Ils sont utilisés pour définir les caractéristiques de ce que représente leur parent, et leur valeur ne change pas fréquemment. Par exemple, une unité d'ingénierie ou un nom de marque.
- *Data Variables* : Elles sont utilisées pour stocker des données qui peuvent changer fréquemment et qui peuvent avoir des enfants tels que des objets, des propriétés, ou d'autres variables de données de base. Un exemple serait la valeur actuelle d'un capteur de température.
- *Objets* : Ils peuvent avoir des enfants et sont utilisés pour organiser d'autres objets, propriétés, variables de données de base, etc. Leur objectif est de modéliser un appareil réel. Par exemple, un capteur de température pourrait avoir des propriétés telles que l'unité technique et le nom de la marque, ainsi qu'une variable représentant la valeur actuelle.
- *Folders* : Les dossiers d'objets sont des instances de QUaFolderObject qui existent toujours sur le serveur pour servir de conteneurs pour toutes les instances utilisateur.

[2]

Programmation du serveur

Fonctionnement du serveur

Dans le SDK, des exemples de code ont été fournis pour chaque point expliqué sur "Github". Mon objectif est de créer un seul code intégrant tous les éléments nécessaires. Pour ce faire, j'ai créé une branche dédiée à mon projet sur "GitLab" et j'ai commencé à tester chaque exemple du SDK. Chaque fois que je comprends la fonctionnalité d'un exemple et le trouve utile pour mon objectif, je l'intègre dans mon code et je "commit" sur "git". Avant de commencer, j'ai procédé à l'installation des dépendances requises par "QUaServer". Ensuite, j'ai fait un "Clone" de "QUaServer" sur mon PC et exécuté la commande suivante pour installer les projets "open62541" et "Mbedtls". Cette dernière sera responsable du chiffrement entre le serveur et le client. Ces bibliothèques sont incluses dans GitHub comme des "submodules".

```
git submodule update --init --recursive
```

Les "submodules" sont simplement une référence à un autre dépôt qui nous permet de les installer dans notre projet en utilisant la commande ci-dessus. Les submodules sont utilisés pour éviter d'inclure d'autres projets lourds d'un autre dépôt dans notre projet. Ensuite, j'ai procédé à la compilation du fichier "amalgamation.pro" (Figure 6), responsable de compiler le projet open62541 et de fournir sa bibliothèque, laquelle est nommée "open62541.lib"

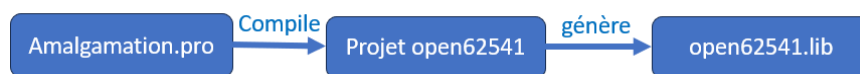


Figure 6 Etapes de génération de open62541.lib

Ensuite, pour inclure le QUaServer dans mon projet, il faut inclure le "quaserver.pri" dans le fichier ".pro". Enfin, je compile le projet "examples.pro" (Figure 7) qui va compiler tous les exemples dans le SDK.

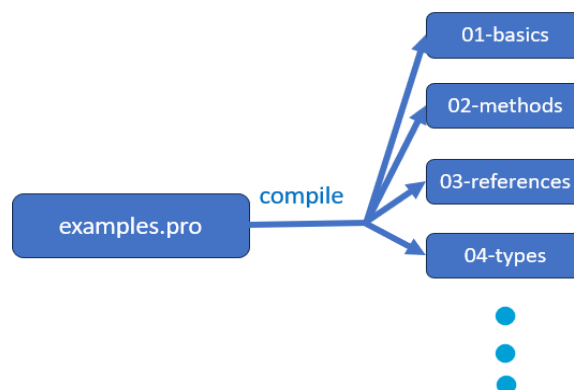


Figure 7 Fonctionnement de Examples.pro

J'ai sélectionné l'exemple "05-server" pour commencer le développement, en utilisant initialement son certificat de test avant de créer le mien. Mon travail s'est déroulé sur Visual Studio 2022. Pour lancer le serveur, il est nécessaire d'appeler le fichier "vcvars64.bat" qui sert

à configurer l'environnement pour le compilateur C++, comme le montre la Figure 8.

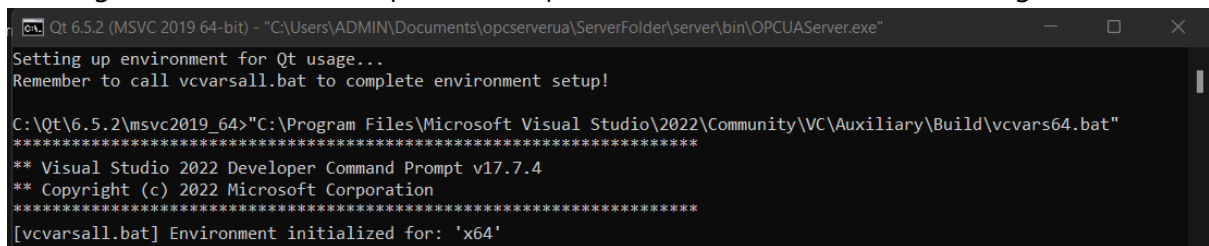


Figure 8 Configure l'environnement pour le compilateur C++

Ensuite, on lance l'exécution du serveur (Figure 9) :

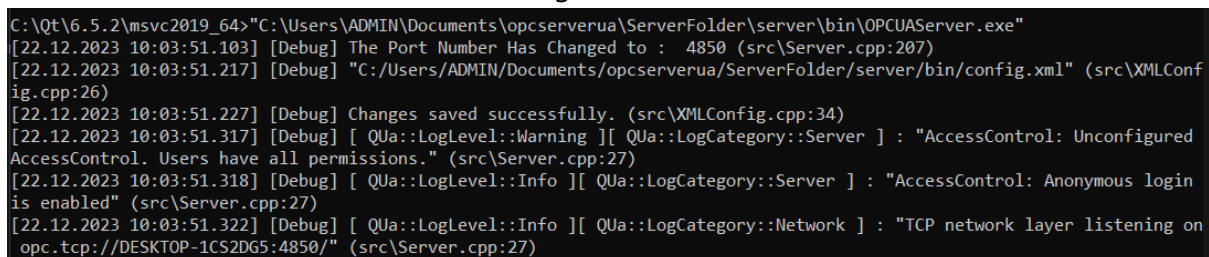


Figure 9 Exécution du serveur OPC

Le serveur a automatiquement pris le nom de l'ordinateur pour construire son URL. Maintenant, nous pouvons nous connecter au serveur en utilisant cette URL dans UAExpert comme le montre la Figure 10, en utilisant le protocole TCP.

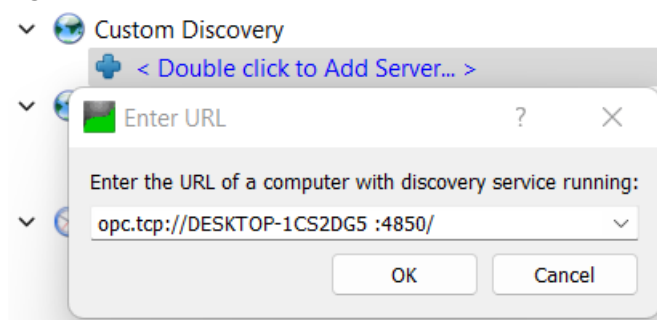


Figure 10 Insertion de l'URL dans UAExpert

Lors de la connexion, il est possible d'explorer tous les tags présents dans le serveur à partir de l'espace d'adressage (Figure 11).

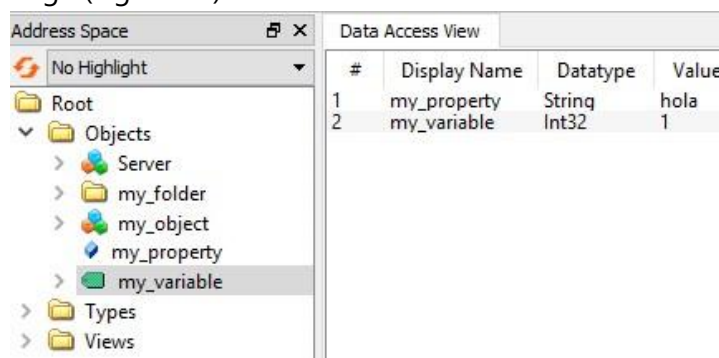
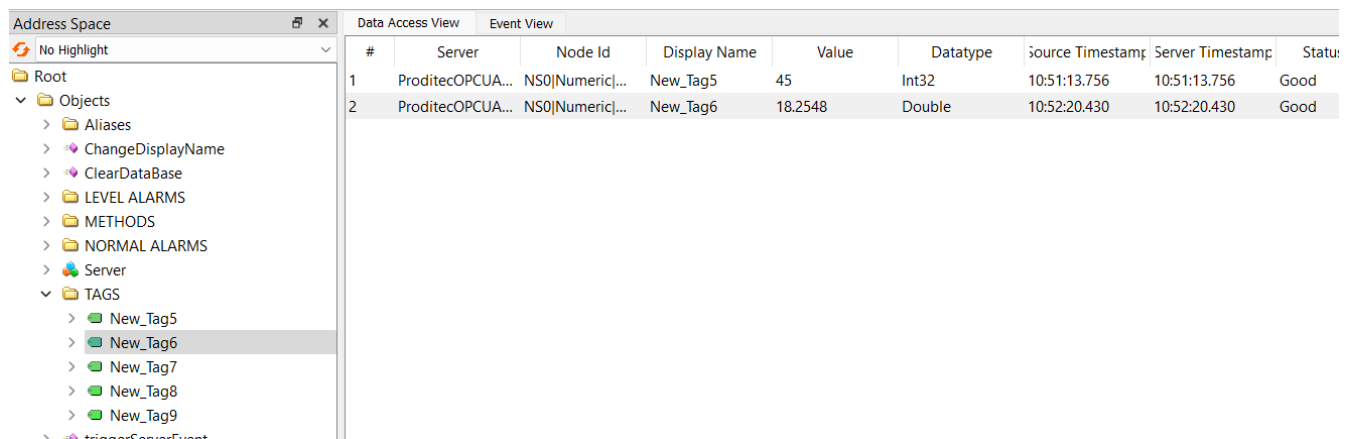


Figure 11 Exploration de l'espace d'adressage du serveur

Pour initier la programmation du serveur, j'ai activé simultanément les alarmes, l'historisation, les événements, et le chiffrement. Afin de les activer en une seule étape, j'ai recompilé le projet "open62541" en activant les alarmes et les événements. De plus, j'ai ajouté les paramètres "ua_alarms_conditions", "ua_historizing", "ua_encryption", et "ua_events" dans la configuration de "amalgamation.pro" et "examples.pro". En activant l'option "ua_encryption", le processus de compilation intégrera automatiquement le projet "Mbedtls" pour générer la bibliothèque "mbedtls.lib". Cependant, le chiffrement ne se produira qu'après la génération des certificats.

Maintenant, je suis capable de :

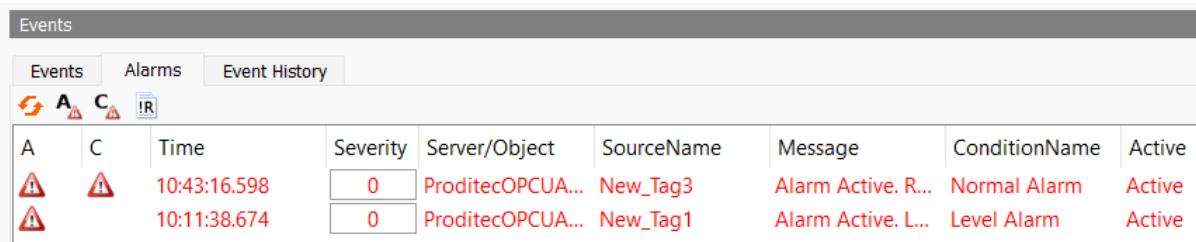
- Accéder aux données depuis UAExpert (OPC DA). (Figure 12)



#	Server	Node Id	Display Name	Value	Datatype	Source Timestamp	Server Timestamp	Status
1	ProditecOPCUA...	NS0[Numeric]...	New_Tag5	45	Int32	10:51:13.756	10:51:13.756	Good
2	ProditecOPCUA...	NS0[Numeric]...	New_Tag6	18.2548	Double	10:52:20.430	10:52:20.430	Good

Figure 12 Accéder aux données du serveur

- De créer des tags qui sont des alarmes et de tester si l'alarme se déclenche (Figure 13) lorsqu'on modifie la valeur de ces tags au-delà de leur seuil (OPC AE).



A	C	Time	Severity	Server/Object	SourceName	Message	ConditionName	Active
!	!	10:43:16.598	0	ProditecOPCUA...	New_Tag3	Alarm Active. R...	Normal Alarm	Active
!	!	10:11:38.674	0	ProditecOPCUA...	New_Tag1	Alarm Active. L...	Level Alarm	Active

Figure 13 Déclenchement des alarmes

- Envoyer des événements : j'ai créé une méthode accessible aux clients. Cette méthode prend en paramètre le message à envoyer ainsi que la gravité (Figure 14 et Figure 15). Cependant, cette méthode sera utilisée par la bibliothèque (DLL) que je vais développer ultérieurement (OPC AE).

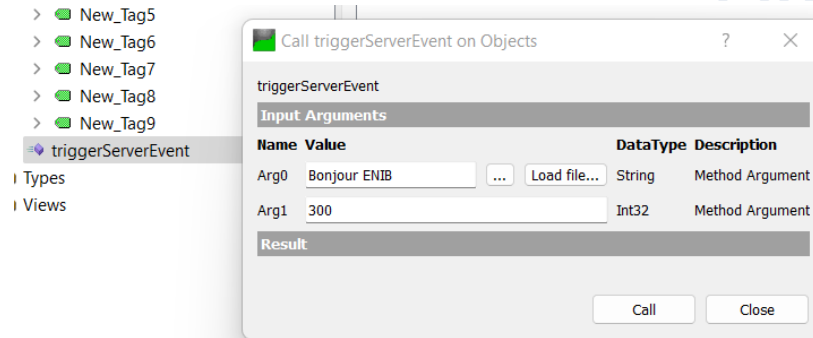


Figure 14 Appel de la fonction "triggerServerEvent"

Events								
Events Alarms Event History								
A	C	Time	Severity	Server/Object	SourceName	Message	EventType	Active
		10:54:32.110	300	ProditecOPCUA...	Server	Bonjour ENIB	MyEvent	

Figure 15 Accéder aux événements envoyés

- Historiser les données des tags (Figure 16 et Figure 17) et des événements (Figure 18) dans une base de données SQLite. Une base de données SQLite sera créée en démarrant le serveur pour la première fois après l'activation de l'historisation. Sur chaque tag, nous aurons le choix de décider si nous voulons historiser ses données ou non (OPC HDA).

Table : ns=0;i=54016			
ns=0;i=54016	Time	Value	Status
Filtre	Filtre	Filtre	Filtre
1	727	1703238740430	18.2548 0

Figure 16 La valeur de New_Tag5 historisée dans la base de données.

Table : ns=0;i=54015			
ns=0;i=54015	Time	Value	Status
Filtre	Filtre	Filtre	Filtre
1	18	1703238673756	45 0

Figure 17 La valeur de New_Tag6 historisée dans la base de données.

Table : ns=0;i=53998											
ns=0;i=53998	EventId	EventNodeId	EventType	LocalTime	Message	OriginNodeId	ReceiveTime	Severity	SourceName	SourceNode	Time
Filtre	Filtre	Filtre	Filtre	Filtre	Filtre	Filtre	Filtre	Filtre	Filtre	Filtre	Filtre
1	466	BLOB	ns=0;i=53999	ns=0;i=53998	NULL	Bonjour ENIB	ns=0;i=2253	-11644473600000	300	Server	ns=0;i=2253 1703238872110

Figure 18 L'événement historisé dans la base de données.

Pour que le client puisse vider la base de données lorsque celle-ci devient importante, j'ai également créé une méthode pour faciliter cette opération comme le montre la Figure 19 :

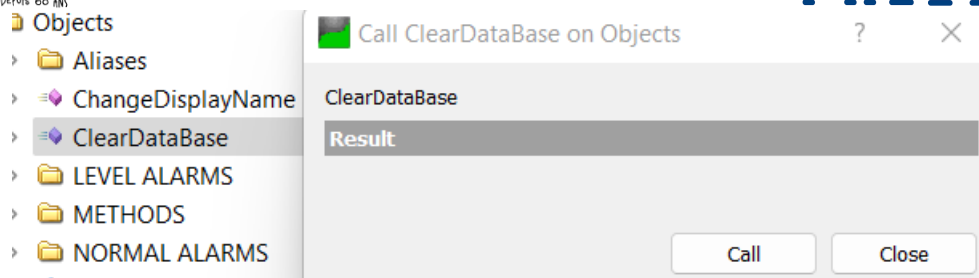


Figure 19 Appel de la méthode "ClearDataBase"

Ainsi, les clients de Proditec ont demandé l'autorisation de changer le nom d'affichage (DisplayName) des tags à leur guise. Par conséquent, j'ai également créé une autre méthode pour les aider à accomplir cela. Cette méthode prend en paramètre le BrowseName du tag et le nouveau nom d'affichage qu'ils souhaitent utiliser. Par exemple, le tag "New_Tag5" a un BrowseName "NewTag5-Proditec" (Figure 20). Le BrowseName servira d'identifiant pour les Tags et les alarmes et, par conséquent, ils ne doivent pas être modifiables.

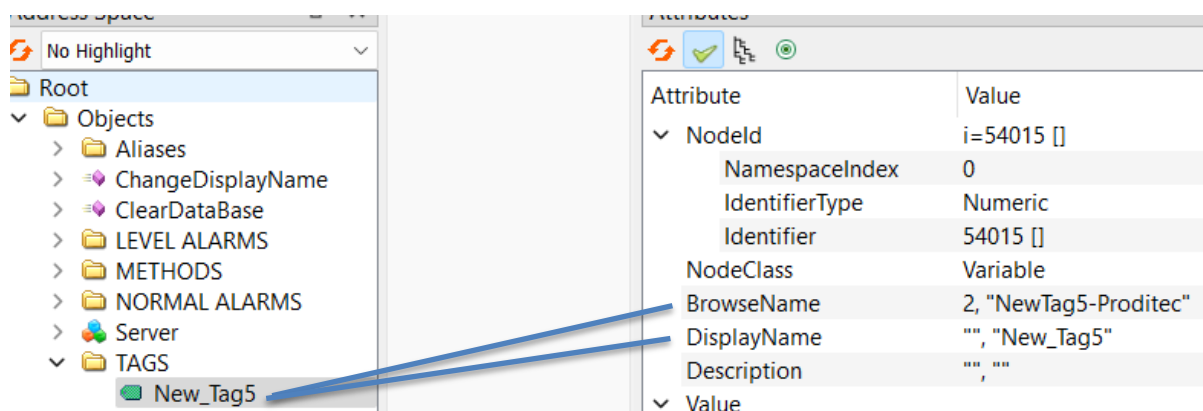


Figure 20 BrowseName de "New_Tag5"

Pour changer son DisplayName on appelle la méthode comme le montre la Figure 21:

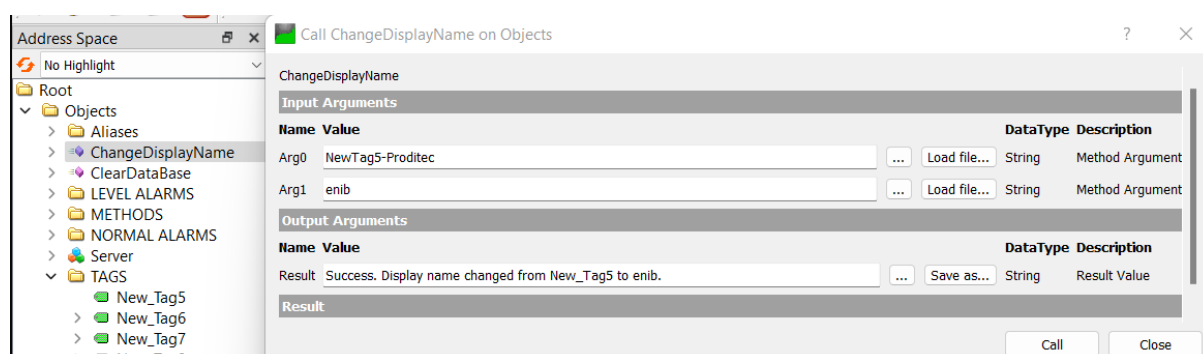


Figure 21 Appel de la méthode "ChangeDisplayName"

Résultat (Figure 22) :

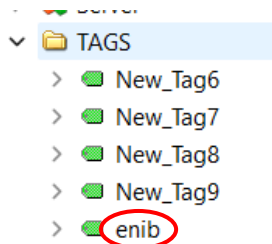


Figure 22 Résultats de l'appel de la méthode "ChangeDisplayName"

Par rapport à la sécurité, j'ai créé un certificat en utilisant "OpenSSL" (qui est une boîte à outils de chiffrement comportant deux bibliothèques, libcrypto et libssl, fournissant respectivement une implémentation des algorithmes cryptographiques et du protocole de communication SSL/TLS) afin que UAExpert fasse confiance à notre serveur. Pour créer ce certificat, j'avais besoin d'installer "MySys2", qui nous fournit une ligne de commande utilisant "OpenSSL".

La première étape consiste à créer une Autorité de Certification (CA). La CA jouera le rôle d'un intégrateur système chargé d'installer des serveurs OPC dans une usine. La CA doit :

- Créer sa propre paire de clés publique et privée.
- Créer son propre certificat auto-signé.
- Créer sa propre liste de révocation de certificats (CRL).

Les clés peuvent être générées et transformées en divers formats, mais la plupart des applications OPC UA utilisent ultimement le format "DER". Ensuite, pour chaque serveur que l'intégrateur système souhaite installer, les étapes suivantes doivent être appliquées :

- Créer sa propre paire de clés publique et privée.
- Créer un fichier "exts.txt" contenant les extensions de certificat requises par la norme OPC UA. Le fichier "exts" nécessite l'inclusion du nom de l'ordinateur sur lequel le serveur doit être installé, ce qui implique que chaque machine chez Proditec disposera de son propre certificat.
- Créer son propre certificat non signé et une demande de signature de certificat.
- Transmettre la demande de signature de certificat à la CA pour signature.

Une fois terminé, on peut voir tous les fichiers créés comme le montre la Figure 23 et si on ouvre le certificat "server.crt" on obtiendra un certificat comme le montre la Figure 24.

ca.crl	11/21/2023 2:06 PM	Certificate Revoca...	1 KB
ca.crt	11/21/2023 2:06 PM	Security Certificate	2 KB
ca.crt.der	11/21/2023 2:06 PM	Security Certificate	1 KB
ca.der.crl	11/21/2023 2:09 PM	Certificate Revoca...	1 KB
ca.key	11/21/2023 2:05 PM	KEY File	2 KB
ca.srl	11/21/2023 2:09 PM	SRL File	1 KB
exts.txt	11/21/2023 1:51 PM	Text Document	1 KB
server.crt	11/21/2023 2:09 PM	Security Certificate	2 KB
server.crt.der	11/21/2023 2:09 PM	Security Certificate	1 KB
server.csr	11/21/2023 2:09 PM	CSR File	1 KB
server.key	11/21/2023 2:09 PM	KEY File	2 KB
server.key.der	11/21/2023 2:09 PM	Security Certificate	2 KB

Figure 23 Le répertoire des certificats créé

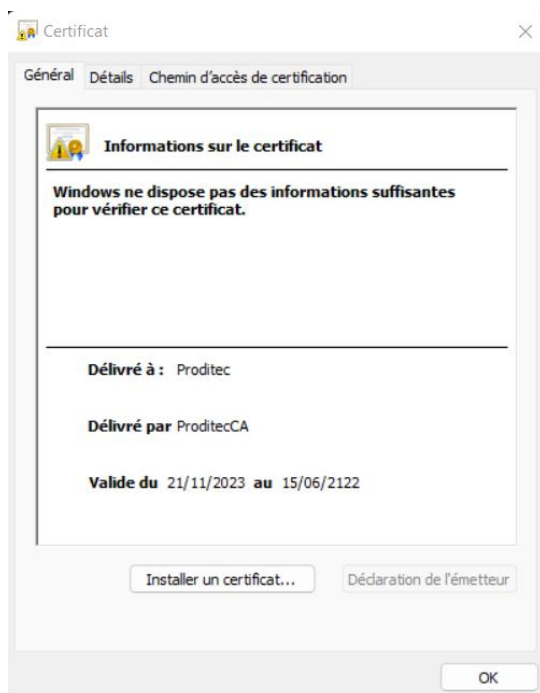


Figure 24 Le certificat

Après la génération des certificats, les fichiers "ca.crt.der" et "ca.der.crl" doivent être envoyés aux clients pour les intégrer dans "UAExpert", afin qu'ils aient confiance envers le serveur. Deux autres fichiers seront utilisés par le serveur : "server.crt.der" et "server.key.der". Ce dernier est utilisé pour intégrer le chiffrement dans le serveur (Figure 25) après avoir activé "Mbedtls". Pour créer un certificat, j'aurai besoin du nom de l'ordinateur où j'installe le serveur. Par conséquent, chaque machine aura son propre certificat.

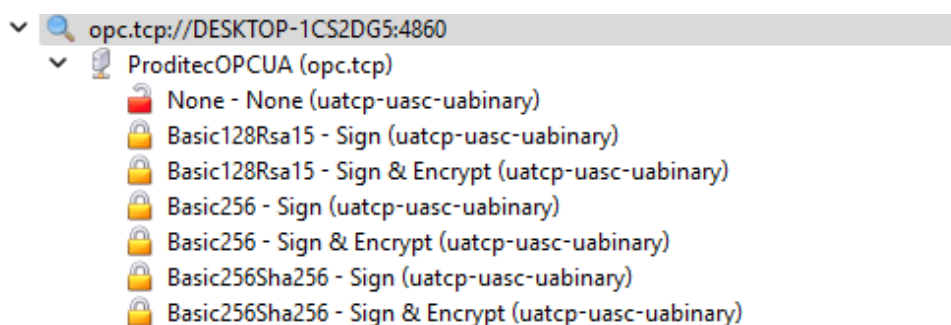


Figure 25 Les mode de chiffrement valable dans le serveur

Tant que les projets "open62541" et "Mbedtls" sont volumineux, je souhaitais également les intégrer dans mon projet "GitLab" en tant que "submodules". Pour ce faire, j'ai utilisé les commandes suivantes :

```
git submodule add -b master <URL_Du_Depot> ServerFolder/depends/mbed.git
```

```
cd ServerFolder/depends/mbed.git
```

```
git checkout <commit_hash>
```

```
cd ../../..
```

```
git add ServerFolder/depends/mbed.git
```

Enfin, j'ai effectué un nettoyage de mon projet sur Git en supprimant les fichiers inutiles, ainsi que tous les exemples fournis par le SDK, ne laissant que l'exemple sur lequel j'étais en train de travailler. Ensuite, j'ai supprimé le fichier du projet "exemples.pro".

Problèmes / Solutions

- Le SDK propose une méthode de sérialisation qui génère un fichier XML pour sauvegarder l'espace d'adressage du serveur, comprenant les tags et les alarmes, ainsi que leurs propriétés. De plus, il offre une méthode de désérialisation pour charger le fichier XML. Dans le but de déclencher automatiquement la méthode "serialize" à l'arrêt du système pour sauvegarder automatiquement l'espace d'adressage, j'ai développé un programme en Python qui se connecte en tant que client à mon serveur, appelle la méthode "serialize" et se déconnecte ensuite. J'étais en train de chercher des moyens pour que ce programme s'exécute automatiquement lors de l'arrêt de l'ordinateur. J'ai découvert une seule méthode, qui consiste à ajouter le programme aux propriétés de l'arrêt du système, comme indiqué dans la figure ci-dessous (Figure 26).

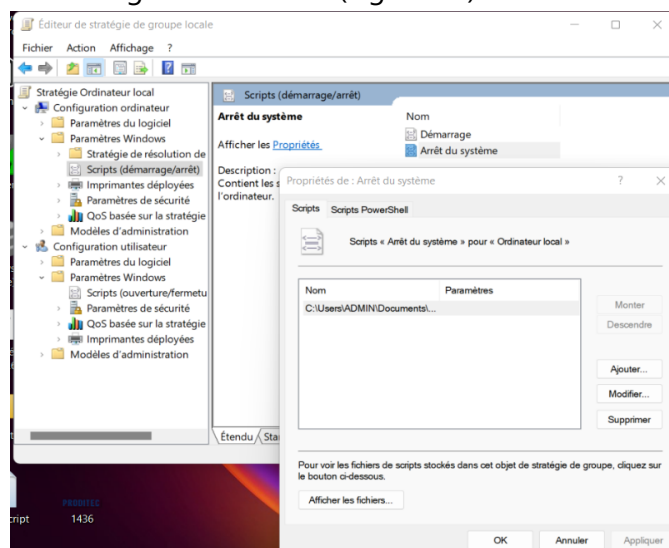


Figure 26 Propriétés de l'arrêt du system

Cependant, cette approche n'a pas fonctionné probablement parce que l'arrêt du système est plus rapide que l'exécution du programme. Ensuite j'ai créé un

fichier batch pour faire tourner mon serveur en boucle, et je l'ai ajouté au démarrage de l'ordinateur pour voir s'il démarre automatiquement. Ça a marché, et ensuite, j'ai continué à travailler sur mon serveur.

Cependant, j'ai découvert que la méthode "deserialize" provoque une erreur lorsqu'une alarme a été sérialisée et que j'essaie de la désérialiser (La documentation du QUaServer a mentionné que les alarmes sont en cours de développement et peuvent poser des problèmes). Pour remédier à cela, j'ai décidé de ne pas utiliser ces méthodes. J'ai créé mon propre fichier XML (Figure 27) dans lequel j'ai organisé les Tags, les "Normal Alarms" et les "Level Alarms". J'y ai ajouté le numéro de port, le chemin des fichiers log, le chemin de la base de données et si le serveur sera en mode anonyme ou pas. Dans le code de mon serveur, je parcours le fichier XML pour configurer le serveur et créer les Tags et les Alarmes. Le mode anonyme laisse les clients se connecter sans avoir des comptes.

```

<Root>
  Numéro de Port : <p port="4860"/>
  Anonymus : <a anonymous="true"/>
  Chemin des fichier log : <l logpath="C:/Users/ADMIN/Documents/opcserverua/ServerFolder/server/bin"/>
  Chemin Base de donnée : <d datahistory="C:/Users/ADMIN/Documents/opcserverua/ServerFolder/server/bin/db.sqlite"/>

  Les TAGS
  <Tags>
    <n browseName="ns=2;s=New_Tag5" dataType="int32_t" description="" displayName="New_Tag5" historizing="true" maxHistoryDataResponseSize="1000" minHistoryDataResponseSize="1000" value="0" valuerank="-2" writeaccess="true"/>
    <n browseName="ns=2;s=New_Tag6" dataType="QString" description="" displayName="New_Tag6" historizing="true" maxHistoryDataResponseSize="1000" minHistoryDataResponseSize="1000" value="" valuerank="-2" writeaccess="false"/>
  </Tags>

  Les LevelAlarms
  <LevelAlarms>
    <n browseName="ns=2;s=New_Tag2" dataType="double" description="" displayName="New_Tag2" highlimit="0.00" historizing="false" lowlimit="0.00" maximum="0.00" minimum="0.00" minimumSamplingInterval="100" value="" valuerank="-2" writeaccess="true"/>
    <n browseName="ns=2;s=New_Tag1" dataType="double" description="" displayName="New_Tag1" highlimit="0.00" historizing="false" lowlimit="0.00" maximum="0.00" minimum="0.00" minimumSamplingInterval="100" value="" valuerank="-2" writeaccess="false"/>
  </LevelAlarms>

  Les NormalAlarms
  <NormalAlarms>
    <n browseName="ns=2;s=New_Tag3" dataType="bool" description="" displayName="New_Tag3" historizing="false" maxHistoryDataResponseSize="1000" minHistoryDataResponseSize="1000" value="false" valuerank="-2" writeaccess="true"/>
    <n browseName="ns=2;s=New_Tag4" dataType="bool" description="" displayName="New_Tag4" historizing="false" maxHistoryDataResponseSize="1000" minHistoryDataResponseSize="1000" value="false" valuerank="-2" writeaccess="false"/>
  </NormalAlarms>
</Root>

```

Figure 27 Le nouveau fichier de configuration XML

- Le code du serveur fonctionnait sous Visual Studio 17 2022, tandis que l'équipe utilisait Qt Creator. J'ai rencontré une erreur inexplicable lors de la compilation du serveur sur QtCreator, mais après des recherches, j'ai découvert sur un forum GitHub qu'il était nécessaire d'inclure la bibliothèque "Advapi32" fournie par Windows Kits, ce qui a résolu le problème. [5]
- Au début du stage, en raison de la complexité du répertoire du serveur, il m'arrive parfois de commettre de légères erreurs, ce qui entraîne l'échec de la compilation du code. Dans ces cas, j'ai utilisé Git pour revenir au dernier commit, mais par la suite, il est devenu plus simple de déboguer les problèmes auxquels je faisais face.
- Lors de la génération des certificats, j'ai constaté un avertissement lors de la connexion sur UAExpert. Après avoir effectué des recherches, j'ai découvert qu'il est nécessaire d'ajouter "dataEncipherment" et "nonRepudiation" dans la section "KeyUsage" du fichier "exts.txt". [3]

- Pour créer le "submodule" de "Mbedtls", j'ai rencontré des problèmes car le projet "Mbedtls" que je souhaite utiliser correspond à un commit spécifique d'une branche particulière d'un projet, et tous les tutoriels sur Internet expliquent comment le faire sur la dernière commit de la branche master. Pour résoudre cela, je me suis rendue sur les stackoverflow pour trouver la méthode correcte. [7]
- On m'a demandé de créer un fichier log, et après l'avoir généré, j'ai constaté qu'il était nécessaire de le vider manuellement pour éviter qu'il ne devienne trop volumineux. Pour remédier à cela, je l'ai régénéré à l'aide d'une méthode récursive c.-à-d., 5 fichiers log seront générés et chacune aura une taille maximum de 1MB quand tous les fichiers log seront remplis, le fichier le plus ancien sera réécrit de nouveau pour éviter que le fichier log ne soit trop volumineux.

Code

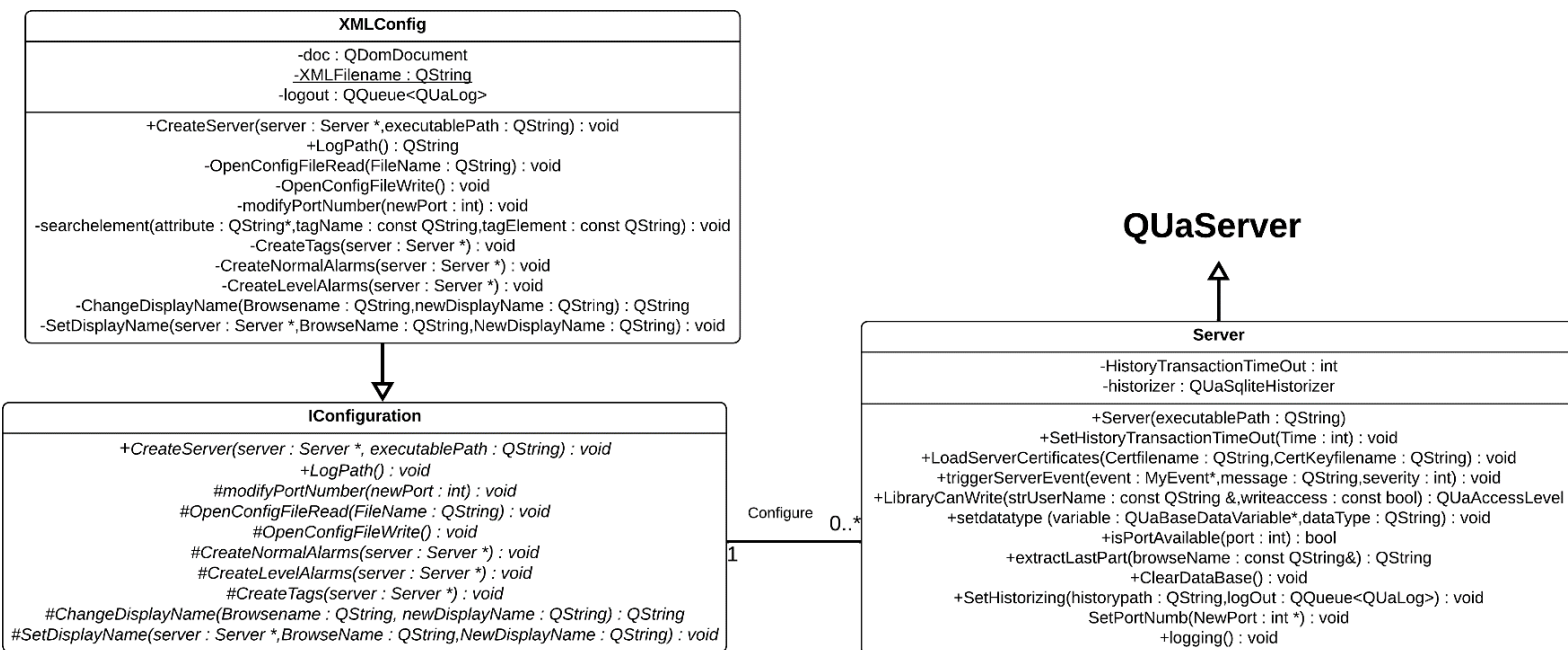


Figure 28 UML du code du serveur

Le code serveur comprend trois classes : IConfiguration, XMLConfig et Server (Figure 28).

- IConfiguration est une classe abstraite qui définit les méthodes de configuration du serveur.
- XMLConfig est une classe concrète qui hérite de IConfiguration et implémente ses méthodes. Elle est responsable de la configuration du serveur à partir d'un fichier XML.
- Server est une classe concrète qui représente le serveur lui-même et hérite de la classe QUaServer fourni par le SDK.

La classe "Server" a pour fonction d'initialiser mon serveur en incluant les certificats et en créant les méthodes "triggerServerEvent" et "ClearDataBase". J'ai également développé une

méthode qui accorde des autorisations d'écriture à un compte spécifique, ce compte étant destiné à la future bibliothèque que je vais créer (DLL). En ce qui concerne la classe XMLConfig, elle prend mon serveur et son chemin en paramètre (dans la méthode "CreateServer") pour le configurer. Cette classe parcourt le fichier XML, recherche le numéro de port, l'associe au serveur en utilisant la méthode définie dans la classe "Server", puis réécrit le numéro de port dans le fichier XML en cas de modification due à son invalidité. Ensuite, elle extrait le chemin de la base de données pour l'ouvrir ou la créer s'il n'existe pas, met le serveur en mode anonyme ou non, et crée la méthode "ChangeDisplayName" utilisée par le client. Enfin, elle crée les tags et les alarmes dans le serveur.

Fichier Main

```
#include "XMLConfig.h"

IConfiguration *config = nullptr;

void customMessageHandler(QtMsgType type, const QMessageLogContext& context,

int main(int argc, char* argv[]) {
    QApplication a(argc, argv);
    qInstallMessageHandler(customMessageHandler);
    QString executablePath = QApplication::applicationDirPath();
    config = new XMLConfig();
    Server server(executablePath);
    config->CreateServer(&server,executablePath);

    server.addUser("libraryuser", "pass12321");
    server.logging(); //if logging() were displayed in the constructor, it w
    server.setApplicationName("ProditecOPCUA");
    server.setApplicationUri("urn:unconfigured:application"); //important to
    server.setManufacturerName("Proditec");
    server.setProductName("OPC UA Server");

    server.start();
    return a.exec();
}
```

Figure 29 Fichier principale du serveur

Dans le fichier principal qui contient la fonction "main" (Figure 29), la fonction "customMessageHandler" est définie pour créer les fichiers journaux (logs), et à l'intérieur de cette fonction, nous extrayons le chemin des fichiers logs depuis le fichier XML. Dans la fonction « main », j'ai associé la fonction "customMessageHandler" à "qInstallMessageHandler", qui requiert une fonction statique en tant que paramètre. "qInstallMessageHandler" est utilisé pour appeler "customMessageHandler" chaque fois qu'un message de débogage, d'avertissement ou d'erreur est émis par l'application Qt. Ensuite, la variable "executablePath" représente le chemin du répertoire du fichier

exécutable, et elle est utilisée pour déterminer les chemins des fichiers journaux, de la base de données, des certificats, et du fichier XML de configuration. Après cela, j'initialise l'objet "config" en tant que classe "XMLConfig" et j'appelle la méthode "CreateServer" en passant l'objet "server" en paramètre pour le configurer. Enfin, je crée un utilisateur spécifique pour la bibliothèque et je configure les propriétés du serveur.

Installeur

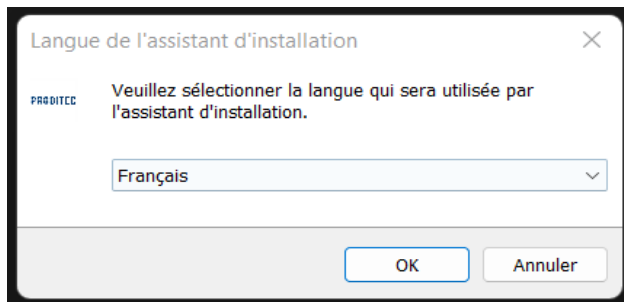


Figure 30 choisir la langue de l'installateur

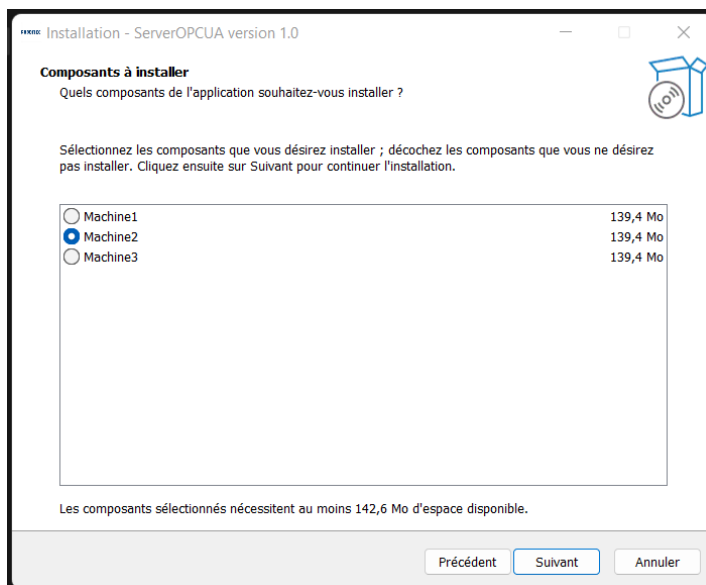


Figure 31 Choisir la machine

Après avoir terminé le développement du serveur, j'ai entrepris de le placer dans un installateur. Dans cet installateur, j'avais besoin d'une page permettant à l'utilisateur de sélectionner la machine sur laquelle le serveur serait installé. En fonction de ce choix, le fichier de configuration approprié serait automatiquement installé. Au cours de mes recherches sur Internet, j'ai examiné plusieurs installateurs tels que NSIS et InstallForge, mais j'ai trouvé qu'ils étaient soit difficiles à configurer en raison de leurs scripts complexes, soit ne disposaient pas de la fonctionnalité précitée. Finalement, j'ai découvert "Inno Setup" qui repose sur un script, mais qui était facile à utiliser et capable de gérer plusieurs composants, ainsi que d'installer des fichiers en fonction du composant choisi par l'utilisateur. En modifiant le script par défaut généré par Inno Setup, j'ai inclus le fichier exécutable du serveur ainsi que toutes ses dépendances telles que les DLLs et les fichiers de configuration. Le processus d'installation commence par permettre à l'utilisateur de choisir la langue (Figure 30), puis de spécifier la machine (Figure 31) et le répertoire d'installation (Figure 32).

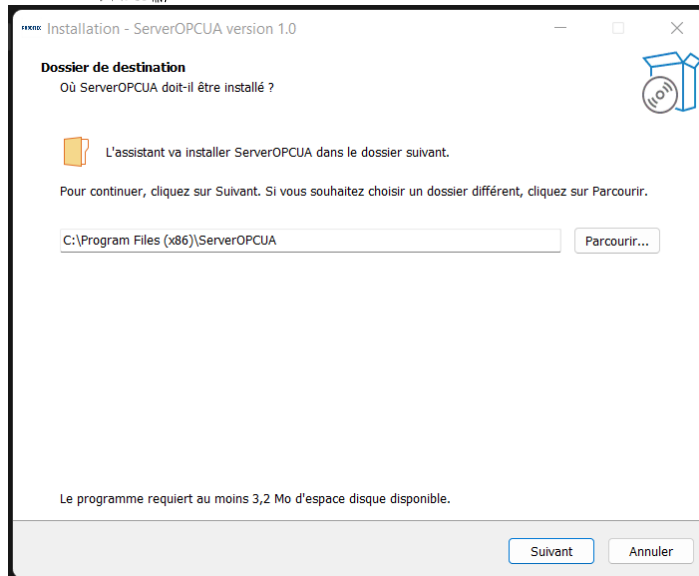


Figure 32 Choisir la destination

Pour importer toutes les DLL utilisées par mon serveur dans son répertoire, permettant ainsi son démarrage en ligne de commande (CMD) sans nécessiter l'utilisation de la ligne de commande de Qt, j'ai employé l'outil windeployqt de Qt :

`cd C:/Path/To/Executable/File`

`windeployqt --debug ServerOPC.exe`

En ce qui concerne le script, j'ai créé des composants représentant les différents types de machines chez Prodittec. Ensuite, j'ai inclus l'exécutable du serveur et ses dépendances comme des fichiers partagés entre tous les composants. Après cela, j'ai ajouté les fichiers de configuration et j'ai associé à chaque machine le fichier de configuration correspondant (Figure 33)

```
[Components]
Name: "Machine1"; Description: "Machine1"; Flags: exclusive; Types: Custom
Name: "Machine2"; Description: "Machine2"; Flags: exclusive; Types: Custom
Name: "Machine3"; Description: "Machine3"; Flags: exclusive; Types: Custom

[Types]
Name: "custom"; Description: "Select Machine"; Flags: iscustom

[Files]
Source: "C:\Users\ADMIN\Documents\opcserverua\ServerFolder\server\bin\{MyAppExeName}"; DestDir: "{app}"; Components: Machine1
Source: "C:\Users\ADMIN\Documents\opcserverua\ServerFolder\server\bin\*"; DestDir: "{app}"; Components: Machine1 Machine2 Machine3
Source: "C:\Users\ADMIN\Documents\opcserverua\ServerFolder\server\bin\networkinformation\*"; DestDir: "{app}\networkinformation"; Components: Machine1 Machine2 Machine3
Source: "C:\Users\ADMIN\Documents\opcserverua\ServerFolder\server\bin\sqlldrivers\*"; DestDir: "{app}\sqlldrivers"; Components: Machine1 Machine2 Machine3
Source: "C:\Users\ADMIN\Documents\opcserverua\ServerFolder\server\bin\tls\*"; DestDir: "{app}\tls"; Components: Machine1 Machine2 Machine3
Source: "C:\Users\ADMIN\Documents\opcserverua\ServerFolder\server\bin\translations\*"; DestDir: "{app}\translations"; Components: Machine1 Machine2 Machine3
Source: "C:\Users\Omar ABDEL KADER\Desktop\config1\config.xml"; DestDir: "{app}"; Components: Machine1; Flags: ignoreversion
Source: "C:\Users\Omar ABDEL KADER\Desktop\config2\config.xml"; DestDir: "{app}"; Components: Machine2; Flags: ignoreversion
Source: "C:\Users\Omar ABDEL KADER\Desktop\config3\config.xml"; DestDir: "{app}"; Components: Machine3; Flags: ignoreversion
: NOTE: Don't use "Flags: ignoreversion" on any shared system files
```

Figure 33 Partie du script de l'installateur

En fin de compte, j'ai dû réfléchir à la manière de permettre au serveur de démarrer automatiquement lors du démarrage de l'ordinateur, en mode invisible, immédiatement après l'installation. Pour ce faire, j'ai créé un fichier batch (.bat) qui tourne en boucle pour vérifier si le serveur est en cours d'exécution, le redémarrant si ce n'est pas le cas. Cependant, le défi résidait dans le fait que je ne pouvais pas connaître le chemin d'installation du serveur

choisi par l'utilisateur, nécessaire pour spécifier le chemin de l'exécutable du serveur dans le fichier batch. J'ai également exploré la possibilité de faire en sorte que l'installateur place le fichier batch dans le dossier de démarrage automatique du PC. Finalement, j'ai découvert qu'avec InnoSetup, je pouvais transmettre des arguments en tant que paramètres à un fichier Visual Basic. Cela m'a permis de résoudre le problème en envoyant le chemin d'installation du serveur en tant qu'argument au fichier VB. En outre, j'ai constaté qu'il était possible d'envoyer le fichier VB au dossier de démarrage commun (Common Startup), ce qui permet au fichier d'être exécuté automatiquement au démarrage de l'ordinateur.

[Icons]

Name: "{commonstartup}\RunOpcUA"; Filename: "{app}\RunOpcUA.vbs"; Parameters: "{app}\runopc.bat"; Components: Machine1 Machine2 Machine3; WorkingDir: "{app}";

Dans cette ligne, l'indication « commonstartup » signifie que le fichier RunOpcUA sera envoyé dans le dossier « shell:common startup », et « {app} » représente le chemin du fichier exécutable du serveur que l'utilisateur a choisi. En tant que paramètre, je transmets le fichier batch runopc.bat. Ce mécanisme sera appliqué à tous les composants de l'installateur.

```
@echo off

:mainLoop

tasklist /fi "imagename eq OPCUAServer.exe" | findstr /i "OPCUAServer.exe" >nul
if errorlevel 1 (
    start "" "OPCUAServer.exe"
)

timeout /nobreak /t 5 >nul

goto mainLoop
```

Figure 34 Code du fichier batch

```
Dim WshShell
Set WshShell = CreateObject("WScript.Shell")

Dim runopcPath
runopcPath = WScript.Arguments(0)

Dim command
command = "cmd /c "" & runopcPath & """"

WshShell.Run command, 0

Set WshShell = Nothing
```

Dans le fichier batch, nous vérifions si le serveur est déjà en cours d'exécution. Si ce n'est pas le cas, nous le démarrons, et le programme s'exécute dans une boucle avec un délai de 5 secondes entre chaque itération.

Figure 35 Code du fichier Visual Basic

Dans le code Visual Basic (Figure 35), je récupère le chemin du fichier batch envoyé par l'installateur en tant que paramètre, puis je lance la commande pour démarrer le fichier batch qui va ensuite lancer mon serveur. Lorsque le serveur démarre, son URL sera basée sur le nom du PC sur lequel il est installé (par exemple : `opc.tcp://DESKTOP-RHH4JIQ:4801`).

Pour la création du certificat, j'aurai besoin de ce nom "DESKTOP-RHH4JIQ", ce qui signifie qu'un certificat spécifique sera installé sur chaque machine, car chaque machine à un PC différent et donc une URL différente.

Afin de faciliter la génération des certificats, j'ai inclus l'installateur de MySys2 dans l'installateur de mon serveur ainsi qu'un fichier README qui inclus le tutorial nécessaire pour la création des certificats.

Cela permet à l'équipe de générer les certificats directement après l'installation du serveur sur une machine.

Finalement, j'ai testé l'installateur sur le système d'exploitation de la machine, qui est Windows 10 LTS 2019. Cependant, cela n'a pas fonctionné pour deux raisons. Tout d'abord, ce système exigeait des DLL fournies par Windows pour permettre le démarrage du serveur. Ensuite, la première batch que j'avais créée utilisait une commande pour vérifier si le serveur était en cours d'exécution, une commande qui fonctionnait correctement sur Windows 11 mais pas sur Windows 10 LTS 2019. Pour résoudre ces problèmes, j'ai recherché les DLL manquantes dans le répertoire Windows Kits et j'ai également modifié le code de la batch, comme illustré dans la Figure 34, ce qui a permis de résoudre ces problèmes. Dans la Figure 36, on peut observer une illustration représentant le fonctionnement de l'installateur.

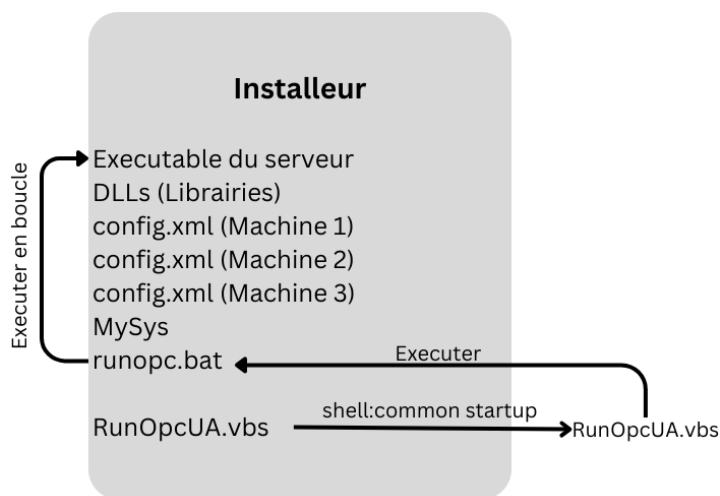


Figure 36 Schéma du fonctionnement de l'installateur

IHM de configuration

Fonctionnement de l'IHM

Pour faciliter la modification du fichier XML de configuration, j'ai créé une interface graphique qui s'en charge.

Au départ, j'ai développé une interface utilisateur permettant la création de comptes clients. Elle enregistrerait des informations telles que le numéro de port, le mode anonyme, le chemin du fichier journal et la base de données des tags dans une base de données. Ensuite, le serveur accédait à cette base de données pour lire ces coordonnées. De plus, le programme effectuait des opérations d'édition sur un ancien fichier XML généré à l'aide de la méthode "serialize" de QUaServer, comprenant des actions telles que la création et la suppression de tags. Cependant, avant de finaliser cette interface, j'ai découvert les problèmes liés à l'ancien fichier XML, ce qui m'a conduit à créer le mien (Figure 27). J'ai donc refait entièrement mon interface. De plus, mon tuteur m'a informé que la création des comptes serait gérée par une DLL en cours de développement par un stagiaire, mais qui n'était pas encore terminée. Donc, dans ma nouvelle interface utilisateur, ma responsabilité se limitera à la configuration du fichier XML. Cette interface se compose d'une fenêtre avec 3 Widgets : le premier (Figure 37) permet de configurer le serveur (port, mode anonyme, emplacement des fichiers logs et de la base de données), le deuxième (Figure 38) permet de gérer les tags et les alarmes (créer, modifier et supprimer), et le troisième (Figure 41) permet d'enregistrer les changements dans le fichier XML.

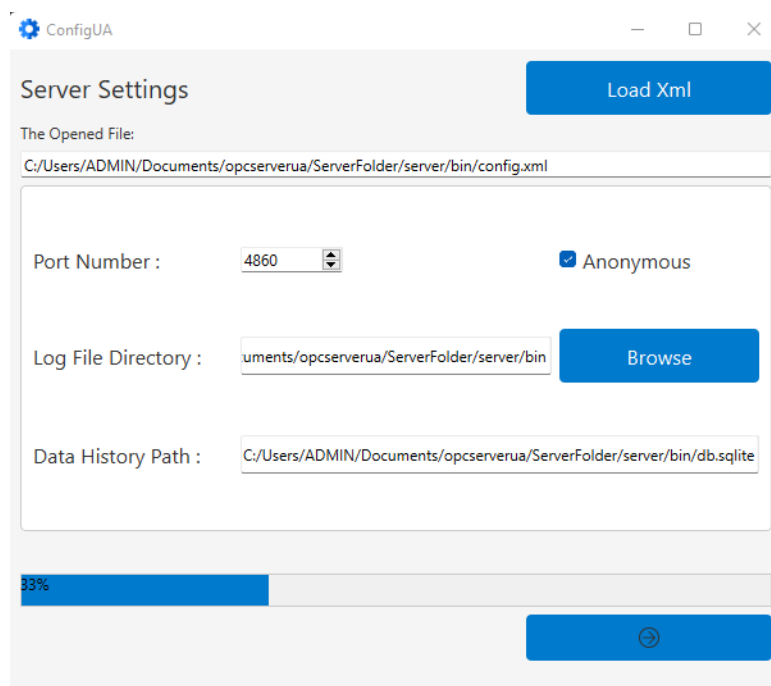
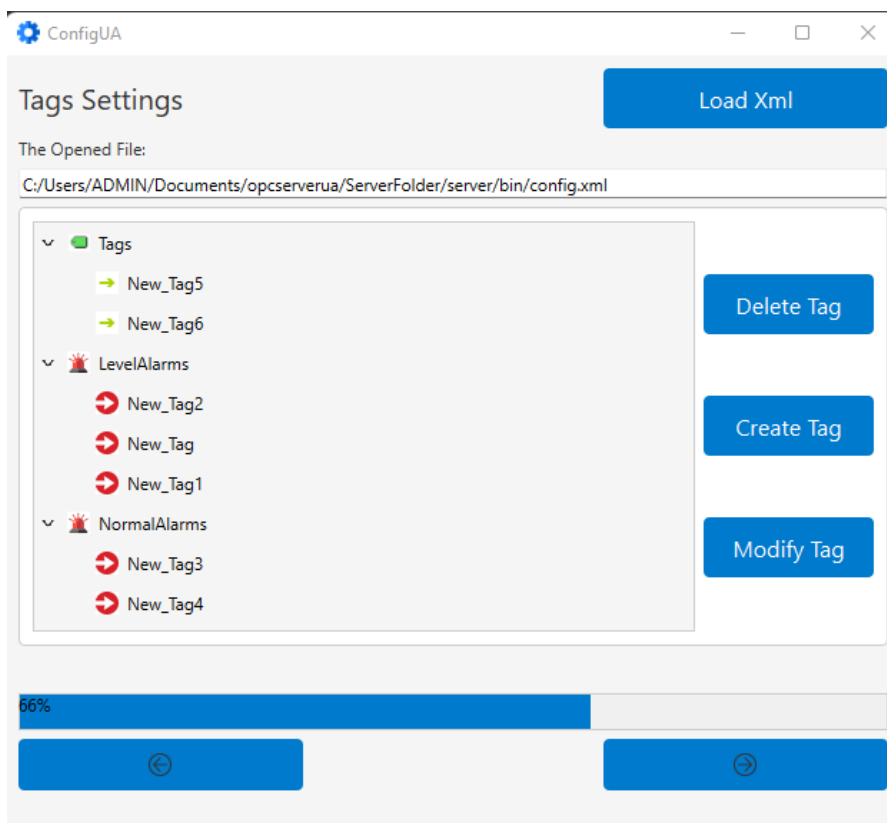


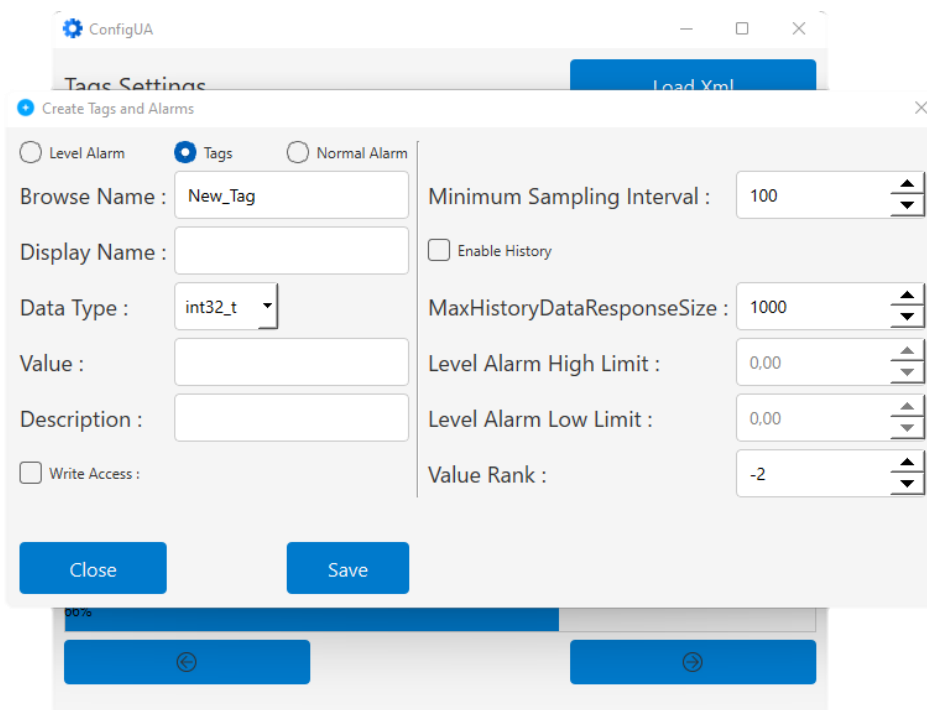
Figure 37 Premier widget de l'IHM (Server Settings)

Tout d'abord, pour modifier un fichier de configuration XML existant, on importe le fichier XML que l'on souhaite configurer. Toutes les données présentes dans le fichier XML occupent alors leurs places respectives dans les champs correspondants de l'interface homme-machine (IHM). Si l'objectif est de créer un nouveau fichier, il suffit de remplir les champs, de créer les tags et les alarmes souhaités. Enfin, lors de la sauvegarde, un nouveau fichier XML est créé à côté du fichier exécutable de l'IHM.



Une fois la modification des paramètres du serveur dans le premier widget terminée, l'utilisateur passe au deuxième widget. Ici, il peut visualiser tous les tags et les alarmes déjà présents dans le fichier de configuration. Pour supprimer un tag, il lui suffit de cliquer sur le tag qu'il souhaite supprimer, puis de cliquer sur le bouton "Supprimer le Tag".

Figure 38 Deuxième widget de l'IHM (Tags Settings)



Ensuite, pour créer un tag ou alarme (Figure 39), l'utilisateur clique sur le bouton "Créer Tag", et une nouvelle fenêtre apparaît. Il remplit les champs requis, puis clique sur "Enregistrer" (ou "Save").

Figure 39 Fenêtre de création des tags

Enfin, pour modifier un tag ou une alarme (Figure 40), l'utilisateur clique sur le tag ou l'alarme qu'il souhaite modifier, puis appuie sur le bouton "Modifier Tag". La même fenêtre

apparaît, mais avec les propriétés du tag à modifier déjà remplies dans leurs champs respectifs. Il peut alors effectuer les modifications nécessaires et appuyer sur "Modifier" pour les sauvegarder.

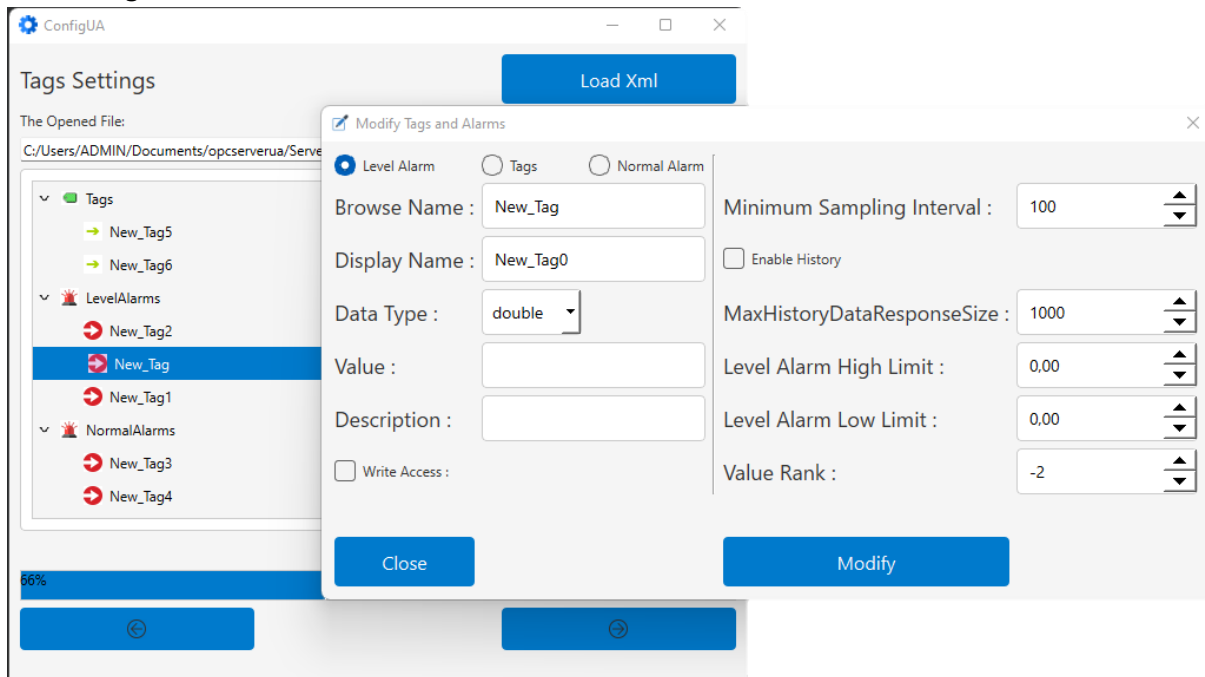
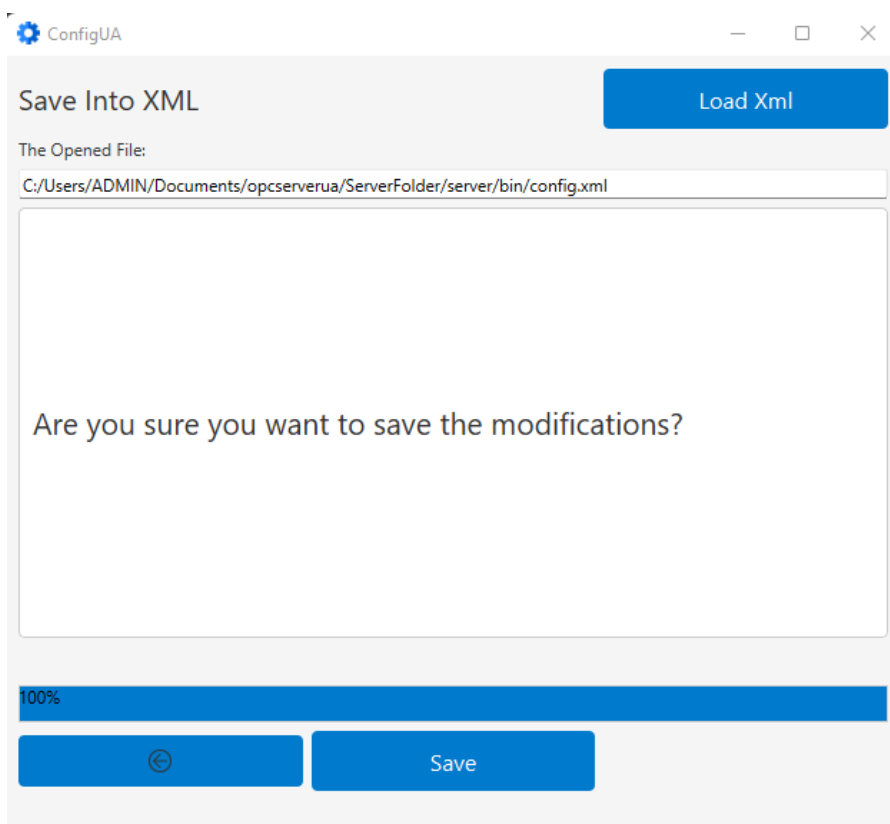


Figure 40 Fenêtre de modification des tags



Enfin, toutes les modifications effectuées dans les deux premiers widgets, seront sauvegardées dans le fichier XML en cliquant sur le bouton "Save" qui apparaît dans le troisième widget.

Figure 41 Troisième widget de l'IHM (Save Into XML)

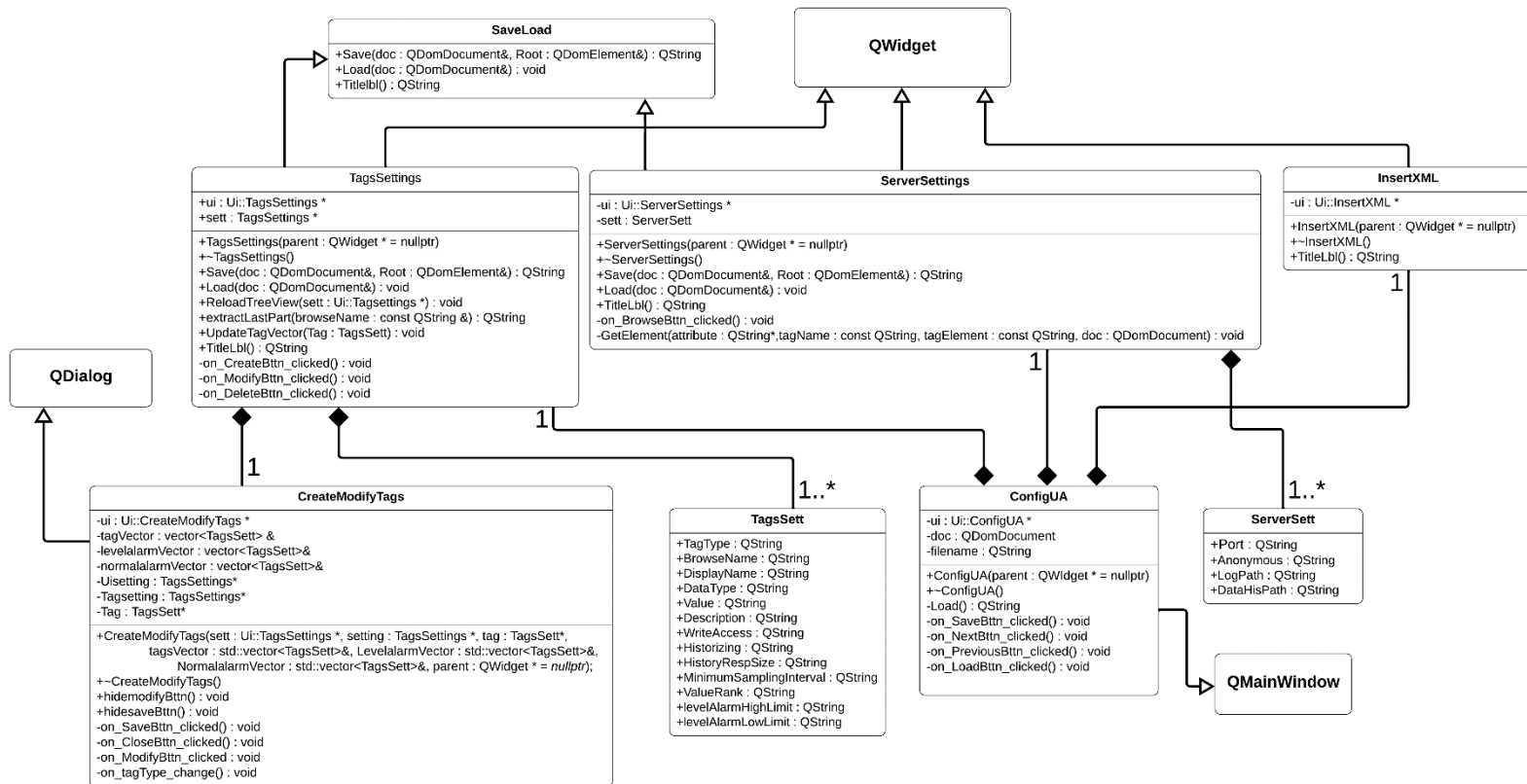


Figure 42 UML du code de l'IHM

Dans ce schéma UML (Figure 42), on observe la classe "ConfigUA" qui représente la fenêtre principale de mon IHM et qui hérite de "QMainWindow". Cette classe contient les trois widgets représentés par les classes suivantes :

- "ServerSettings" : la classe du premier widget (Figure 37)
- "TagsSettings" : la classe du deuxième widget (Figure 38)
- "InsertXML" : la classe du troisième widget (Figure 41)

Les deux premières classes héritent de la classe abstraite "SaveLoad" afin d'implémenter les fonctions nécessaires pour charger le fichier XML. La classe "ServerSett" est utilisée pour porter les modifications que l'utilisateur a effectuées dans le premier widget, tandis que la classe "TagsSett" est utilisée pour porter les modifications que l'utilisateur a apportées dans le deuxième widget. Enfin, la classe "CreateModifyTags" représente la fenêtre où l'on crée et modifie les tags et les alarmes (Figure 39 et Figure 40). Elle est contenue dans la classe "TagsSettings" et hérite de "QDialog".

Les deux premiers widgets implémentent la fonction "Save", qui prend en paramètre un objet QDomDocument représentant le fichier XML et un QDomElement, servant de balise racine dans ce fichier. Dans le premier widget, la fonction "save" récupère tous les champs

remplis par l'utilisateur, crée leurs balises en utilisant l'objet QDomDocument, puis les insère dans l'objet QDomElement. De manière similaire, dans le deuxième widget, cette fonction récupère tous les tags et alarmes créés par l'utilisateur, crée leurs balises en utilisant l'objet QDomDocument, puis les insère dans l'objet QDomElement comme le montre la Figure 43.

```
QString ServerSettings::Save(QDomDocument& doc, QDomElement& Root)
{
    QString Error;
    Recuperation des donnees
    sett.Port = ui->PortNumbSpinbox->text();
    sett.Anonymous = ui->AnonymloginBox->isChecked()?"true":"false";
    sett.DataHisPath=ui->DatahistorypathTxt->text();
    sett.LogPath=ui->LogpathTxt->text();
    Creation des balises
    QDomElement port = doc.createElement("p");
    port.setAttribute("port",sett.Port);
    QDomElement anonymous = doc.createElement("a");
    anonymous.setAttribute("anonymous",sett.Anonymous);
    QDomElement logpath = doc.createElement("l");
    logpath.setAttribute("logpath",sett.LogPath);
    QDomElement datahistory = doc.createElement("d");
    datahistory.setAttribute("datahistory",sett.DataHisPath);
    Insertion dans la balise "Root"
    Root.appendChild(port);
    Root.appendChild(anonymous);
    Root.appendChild(logpath);
    Root.appendChild(datahistory);

    if (sett.LogPath.isEmpty() || sett.DataHisPath.isEmpty())
    {
        Error = "Path is empty";
    }
    return Error;
}
```

Figure 43 Fonction "Save" du premier widget

Dans le troisième widget, lorsqu'on appuie sur le bouton "Save", la procédure consiste à supprimer le contenu du fichier XML déjà existant. Ensuite, un nouvel objet QDomDocument appelé "doc" est créé, ainsi qu'un objet QDomElement appelé "Root". Enfin, les fonctions "Save" des deux premiers widgets sont appelées en leur passant les objets par référence en tant que paramètres. Les données résultantes sont ensuite insérées à nouveau dans le fichier XML.

Dans le contexte du deuxième widget, j'ai mis en place la configuration des Tags et des alarmes en créant trois vecteurs de type "TagsSett" : le premier, appelé TagsVector, le deuxième, appelé LevelAlarmVector, et le troisième, appelé NormalAlarmVector. Lors du chargement du fichier XML, je parcours tous les tags et alarmes présents dans le fichier, puis je remplis les trois vecteurs en conséquence. Ensuite, lors de la création, de la modification ou de la suppression de tags ou d'alarmes dans l'interface utilisateur, ces opérations sont effectuées directement sur les vecteurs. Enfin, lors de l'utilisation de la fonction "Save", je parcours ces vecteurs pour créer les balises correspondantes à l'aide de QDomDocument.

Dans la Figure 44, vous trouverez une illustration du vecteur « LevelAlarmVector ». Lorsque je souhaite modifier un tag, j'utilise les mêmes vecteurs en les passant en paramètre au constructeur de la classe CreateModifyTags. Ensuite, je recherche le tag ou l'alarme que je souhaite modifier en utilisant ces vecteurs. Une fois trouvé, je remplis les données correspondantes dans les champs de l'interface homme-machine (IHM) de la fenêtre de modification des tags pour effectuer les modifications nécessaires.



Figure 44 Illustration du LevelAlarmVector

ModuleOPC (DLL)

Après avoir terminé la conception de mon Interface Homme-Machine (IHM), on m'a demandé de créer une librairie dynamique (DLL) chargée de faciliter la communication entre la machine et le serveur. Cette librairie doit être en mesure de se connecter à mon serveur et d'accomplir les tâches suivantes :

- Écrire dans les Tags
- Écrire dans les Level Alarms
- Écrire dans les Normal Alarms
- Envoyer des Événements
- Fournir un signal pour vérifier la connectivité avec le serveur

Pour cela, j'ai dû installer un autre projet open62541 qui a une version différente que celui du QUaServer et le compiler afin d'obtenir la bibliothèque open62541.lib. Dans cette librairie, j'ai choisi d'utiliser directement la bibliothèque open62541 et de ne pas utiliser le QUaServer, car la DLL sera créée avec Qt 5.9.9, qui n'est pas compatible avec QUaServer. Lorsque j'ai inclus open62541.lib dans mon projet ModuleOPC, j'ai rencontré une erreur étrange qui empêchait le bon fonctionnement. Par la suite, en consultant des forums sur GitHub [9], j'ai découvert qu'il était également nécessaire d'inclure des bibliothèques Windows pour que cela fonctionne.

`LIBS += -lws2_32 -lIPHLPAPI -ladvapi32`

Pour savoir comment utiliser cette bibliothèque, je me suis référé aux documentations officielles. [1]

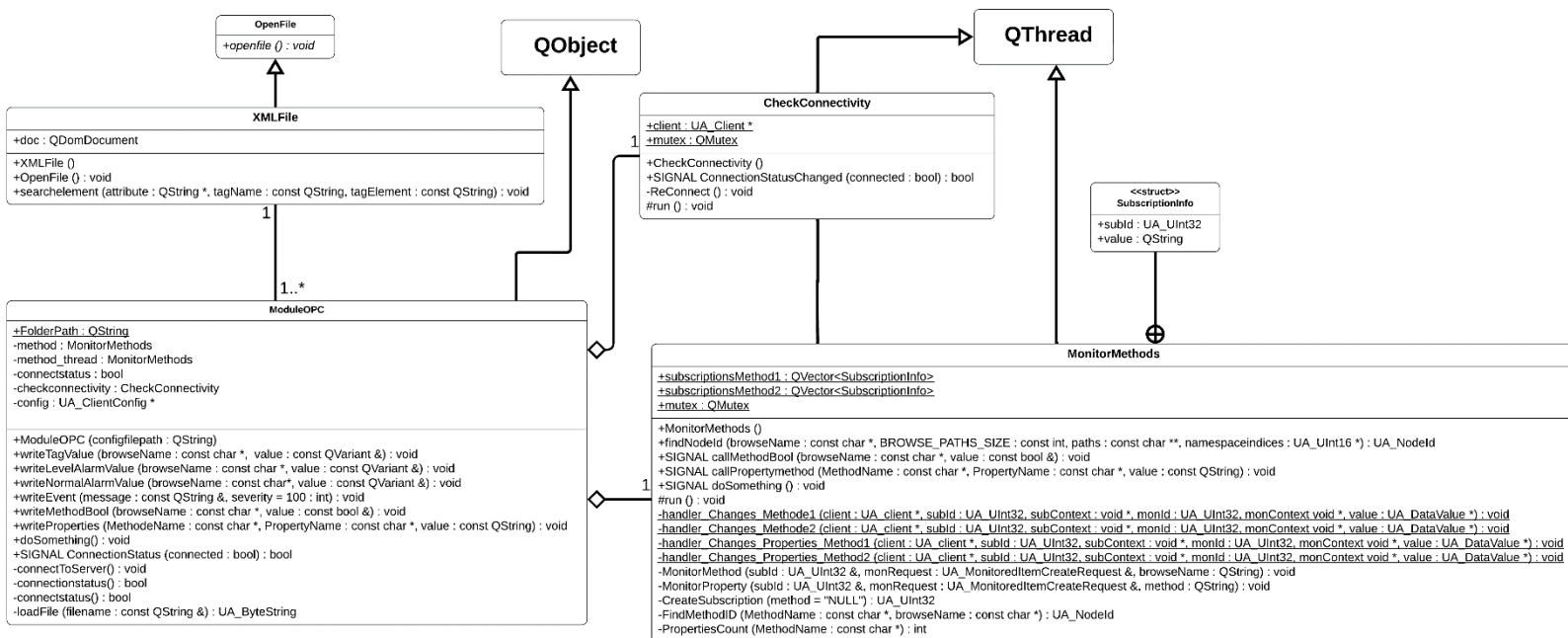


Figure 45 UML du code de la bibliothèque dynamique (DLL)

Dans mon programme, j'ai défini cinq classes (Figure 45). La classe "OpenFile" est une classe abstraite, tandis que la classe "XMLFile" hérite de "OpenFile" et est chargée d'ouvrir un fichier XML pour extraire le numéro de port du serveur en vue d'une connexion. La classe "ModuleOPC" est la classe principale, responsable de la connexion, de l'écriture dans les tags et les alarmes, ainsi que de l'envoi d'événements au serveur. Pour garantir la vérification de la connectivité entre la bibliothèque et le serveur indépendamment du programme initial, j'ai créé la classe "CheckConnectivity" qui est un Thread et implémente la fonction protégée "run" [8], qui tourne en boucle "while", émettant un signal chaque seconde. Une fois ces fonctionnalités mises en place, j'ai été chargé d'explorer la possibilité pour un client depuis UAExpert de pouvoir appeler une méthode de la bibliothèque (DLL), permettant ainsi de donner des ordres à la machine à distance via OPC UA. N'ayant pas pu le faire directement en raison de l'architecture client-serveur, j'ai envisagé la création d'un tag de type booléen. J'ai également mis en place une méthode accessible au client UAExpert. Cette méthode a pour rôle de modifier l'état du tag, qui est de type booléen. (Figure 46)

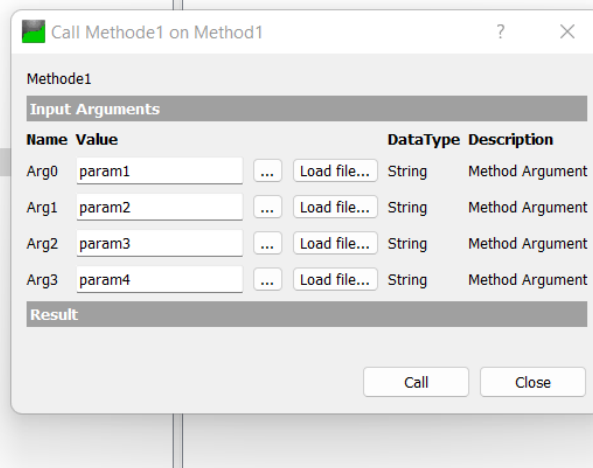


Figure 46 Appel de la Methode1

La bibliothèque surveille en permanence l'état de ce tag, et lorsqu'il devient "true" (lorsque la méthode est appelée), la bibliothèque déclenche une fonction spécifique. Pour rendre la méthode plus flexible, j'ai ajouté des paramètres afin que le client puisse fournir des valeurs précises à la machine si nécessaire.

La classe "MonitorMethods" est un thread chargé de surveiller tous les tags de toutes les méthodes. En cas de changement d'état d'un tag de type booléen, il enverra un signal à la classe "ModuleOPC" pour appeler une méthode spécifique, et les paramètres seront stockés dans un vecteur public accessible par "ModuleOPC" pour une utilisation ultérieure. Enfin, après que le client a appelé une méthode, le tag de type booléen redeviendra faux, et les "properties" redeviendront nulles.

J'ai ensuite dû sécuriser la ressource partagée par tous les threads, qui est l'objet "client" représentant la connexion de la librairie au serveur. Pour cela, j'ai utilisé le "QMutex" pour empêcher que plusieurs threads accèdent simultanément aux ressources partagées, ce qui pourrait provoquer des bugs pendant l'exécution du programme. Enfin, ma librairie se connectait en mode anonyme au serveur et je devais lui donner la permission d'écrire sur tous les tags et les alarmes. Pour cela, j'ai créé dans le serveur, dans le "main", un utilisateur "libraryuser", et je lui ai donné toutes les autorisations nécessaires. Cependant, je n'ai pas pu intégrer ce compte dans la librairie, car la fonction qui permet de se connecter au serveur avec des comptes n'est pas activée.

Pour activer cette fonction, j'ai dû installer le projet OpenSSL, le compiler, puis recompiler le projet open62541 en intégrant les bibliothèques d'OpenSSL.

Bilan du stage

En conclusion, j'ai réussi à répondre à toutes les attentes de Proditec. J'ai mis en place un serveur OPC UA complet, incluant OPC DA, OPC HDA et OPC AE, fonctionnant avec le protocole TCP plutôt que COM/DCOM. Il utilise des certificats et des modes de chiffrement pour se connecter avec le client, et indépendant de Windows et des protocoles. Ensuite, j'ai fourni un outil facilitant la configuration du serveur ainsi que la gestion des tags et des alarmes. Enfin, j'ai développé une DLL capable de se connecter au serveur, d'écrire dans les tags et les alarmes, et d'envoyer des événements aux clients. Cette DLL devra ensuite être connectée au logiciel de la machine (ProdiSoft) pour permettre à la machine d'écrire dans les tags et les alarmes, ainsi que d'envoyer des événements. Actuellement, cette possibilité est entravée par la nécessité de supprimer LabVIEW, ce qui n'est pas réalisable pour le moment. Ainsi, j'ai pu accomplir tout ce qui était demandé au maximum.

Ce stage m'a permis de découvrir et d'explorer la technologie de l'OPC, mettant en lumière son importance dans le domaine de l'industrie. J'ai eu l'opportunité de travailler avec des protocoles de sécurité tels que MbedTLS et OpenSSL. En outre, j'ai renforcé mes compétences en planification de projets et mis en œuvre l'outil de gestion de projet Git, que j'avais déjà eu l'occasion d'apprendre à l'ENIB. J'ai également eu l'opportunité de mettre en pratique l'un des langages les plus répandus, le C++, que j'avais également étudié à l'ENIB, en utilisant le cadre très utilisé de notre époque, le Qt. Ce stage m'a ouvert les yeux sur l'environnement de travail au sein des entreprises professionnelles, me permettant de comprendre les attentes d'une entreprise envers un ingénieur ou un stagiaire. Travailler au sein d'une équipe professionnelle m'a considérablement aidé à changer ma mentalité, en adoptant une approche plus professionnelle. De plus, ce stage m'a fait prendre conscience de l'importance des enseignements dispensés à l'ENIB et a modifié ma perspective sur l'éducation.

Désormais, je ressens une forte motivation pour approfondir mes connaissances davantage. Dans l'ensemble, mon expérience chez Proditec a été très satisfaisante, d'un point de vue personnel et professionnel.

Bibliographie/Webographie

1. (n.d.). Retrieved from open62541: <https://www.open62541.org/doc/master/>
2. Burgos, J. G. (2019-2020). *QUaServer*. Retrieved from QUaServer: <https://github.com/QUaServer/QUaServer>
3. *Creating a CSR and SSL Certificate with SAN Extensions.md*. (n.d.). Retrieved from Github Gist: <https://gist.github.com/GangGreenTemperTatum/38aafed258feb1e048575db5a6e7130b>
4. Foundation, O. (n.d.). *Qu'est-ce qu'OPC ?* Retrieved from <https://opcfoundation.org/about/what-is-opc/>
5. gatkinso. (2017, Janvier 4). *Unresolved symbols Visual Studio 14 2015 ARM*. Retrieved from Github Forum: <https://github.com/Mbed-TLS/mbedtls/issues/735>
6. Proditec. (n.d.). *Document Interne*.
7. user775171. (2012, juin 6). *How do I check out a specific version of a submodule using 'git submodule'?* Retrieved from stack overflow: <https://stackoverflow.com/questions/10914022/how-do-i-check-out-a-specific-version-of-a-submodule-using-git-submodule>
8. vrfEducationEnglish. (2023, juin). *Threading in Qt: How to use QThread*. Retrieved from Youtube: <https://www.youtube.com/watch?v=OxRJtBXFE48&t=812s>
9. xhimanshuz. (2022, Février 9). *library linking in Windows with QMake*. Retrieved from Github Forum: <https://github.com/firebase/firebase-cpp-sdk/issues/844>

Diagramme de Gantt

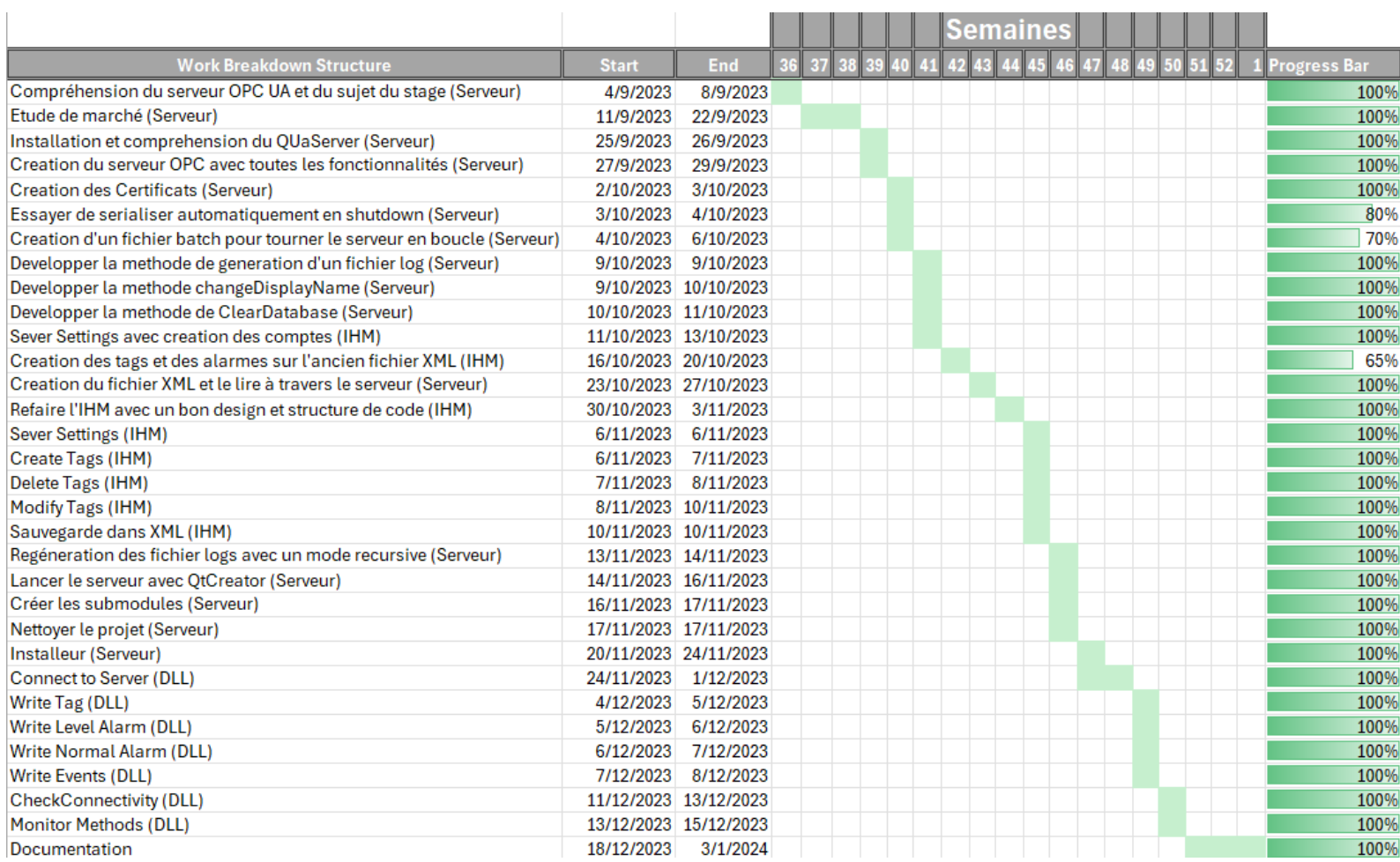


Figure 47 Diagramme de Gantt