Departement informatique - IUT2 Grenoble

Année universitaire 2022 - 2023

Mémoire de Stage

Implémentation d'un module OPC dans Alices

Corys - R&D Alices

Elian Loraux - LP MI ASSR- 27 août 2023

Jury :

IUT :
Responsable ASSR : Pierre-François Dutot
Tutrice IUT : Delphine Chollat-Namy
Corys :
Maitre d'apprentissage : Adrien Tenand

Déclaration de respect des droits d'auteur

Par la présente, je déclare être le seul auteur de ce rapport et assure qu'aucune autre ressource que celles indiquées n'ont été utilisées pour la réalisation de ce travail. Tout emprunt (citation ou référence) littéral ou non à des documents publiés ou inédits est référencé comme tel.

Je suis informé qu'en cas de flagrant délit de fraude, les sanctions prévues dans le règlement des études en cas de fraude aux examens par application du décret 92-657 du 13 juillet 1992 peuvent s'appliquer. Elles seront décidées par la commission disciplinaire de l'UGA.

À Grenoble, le 27 août 2023.

Signature

Remerciements

Je tiens à exprimer ma reconnaissance à mes tuteurs d'apprentissage, Adrien Tenand côté Corys et Delphine Chollat-Namy côté IUT.

Carole Pasini, Frédéric Gargaud et Hugo Wetter ont toute ma gratitude pour leurs conseils avisés et leurs recommandations.

Je souhaite remercier également Laurent Chauvineau pour son aide apportée dans la mise en place de l'outil LaTeX (outil de mise en forme PDF) dont je me suis servi pour rédiger ce rapport.

Pour finir, je remercie le département informatique de l'IUT2 ainsi que Corys pour leur formation et leurs enseignements.

Sommaire

Table des figures

1 | Introduction

1.1 Corys en bref

Corys est une entreprise grenobloise située sur la Presqu'île qui développe des simulateurs dans les domaines du transport, des industries de procédés et du nucléaire.

Corys répond à une problématique de formation, d'ingénierie et de sûreté. En effet, que ce soit pour conduire des trains, superviser des procédés industriels ou piloter des centrales nucléaires, les opérateurs ont besoin d'être formés. Pour l'ingénierie, la simulation permet d'effectuer des tests, des mesures et des relevés afin d'aider à la conception des systèmes.

Pour des besoins de sécurité, dans le domaine nucléaire, la majorité des autorités de sûreté demandent un simulateur à jour pour chaque site nucléaire. C'est le cas notamment en France.

1.2 Plan

Dans la prochaine partie nous allons détailler le contexte de ma mission en présentant le logiciel cible (Alices), le protocole à intégrer, l'objectif et les contraintes de la mission.

Puis, dans la deuxième partie, nous allons voir comment j'ai réalisé la mission. En premier lieu, nous allons présenter quels sont les choix techniques que nous avons dû faire et pourquoi. Ensuite, nous parlerons de l'implémentation au sein du logiciel puis des difficultés rencontrées.

2 | Contexte de la mission

2.1 Atelier Logiciel Intégré de Conception et d'Étude des Simulateurs

Dans le cadre du développement, de la maintenance et de l'exploitation de grands projets de simulation, Corys fournit à ses clients une suite de logiciels intégrés spécifiques à chaque activité. Ces projets concernent plusieurs domaines : le transport, mais aussi l'énergie nucléaire et les industries de procédés.

Je présenterai uniquement Alices, Atelier Logiciel Intégré de Conception d'Études des Simulateurs, outil de conception et d'exécution de simulateurs pour le nucléaire se décomposant en trois niveaux.

Pour illustrer le fonctionnement d'Alices, prenons l'exemple d'un simulateur de chasse d'eau.

Niveau 1 : C'est ici que l'on va créer des objets de base [1]. Cela prend la forme d'actionneurs, de capteurs, de réservoirs... Dans le cadre de notre chasse d'eau, nous avons besoin d'une arrivée d'eau et d'une évacuation, toutes deux contrôlées par des vannes. Nous avons aussi besoin d'un réservoir d'eau. Dans le niveau 1, nous allons donc définir ces objets, leurs comportements physiques, leurs apparences graphiques, etc

Niveau 2 : L'idée est de connecter ensemble les objets définis au niveau 1 [2] afin de créer un sous-système. Pour la chasse d'eau, on connecte l'arrivée d'eau à la vanne d'arrivée, puis le réservoir d'eau, puis une vanne de sortie et l'évacuation. On connecte tout cela à l'aide de canalisations.

Niveau 3 : C'est la configuration et l'exécution du simulateur. On va assembler des modules pour pouvoir construire un simulateur complet. Un module a plusieurs fonctions, il peut :

- Être le résultat du modèle de niveau 2 et représenter un système physique (thermo-hydraulique, électrique...) ;
- Servir de logique afin de réguler les composants. Par exemple, pour la chasse d'eau, on veut garder un niveau d'eau stable dans le réservoir. On fait alors varier l'ouverture des vannes d'entrée et de sortie pour obtenir le résultat souhaité ;
- Représenter une interface ;
- Communiquer avec l'extérieur.

Dans le cadre de la chasse d'eau (sous-système 1), on peut rajouter d'autres sous-systèmes comme une douche et un lavabo afin de simuler une salle de bain complète. On peut aussi rajouter un thermostat pour contrôler un chauffe-eau par l'intermédiaire d'une interface. On pourrait même connecter le tout à une application mobile.

On peut trouver un schéma du système de niveaux d'Alices à la figure 2.9 ainsi que des captures d'écran en annexe.

1. Un objet correspond à une classe C++
2. Instance des classes C++

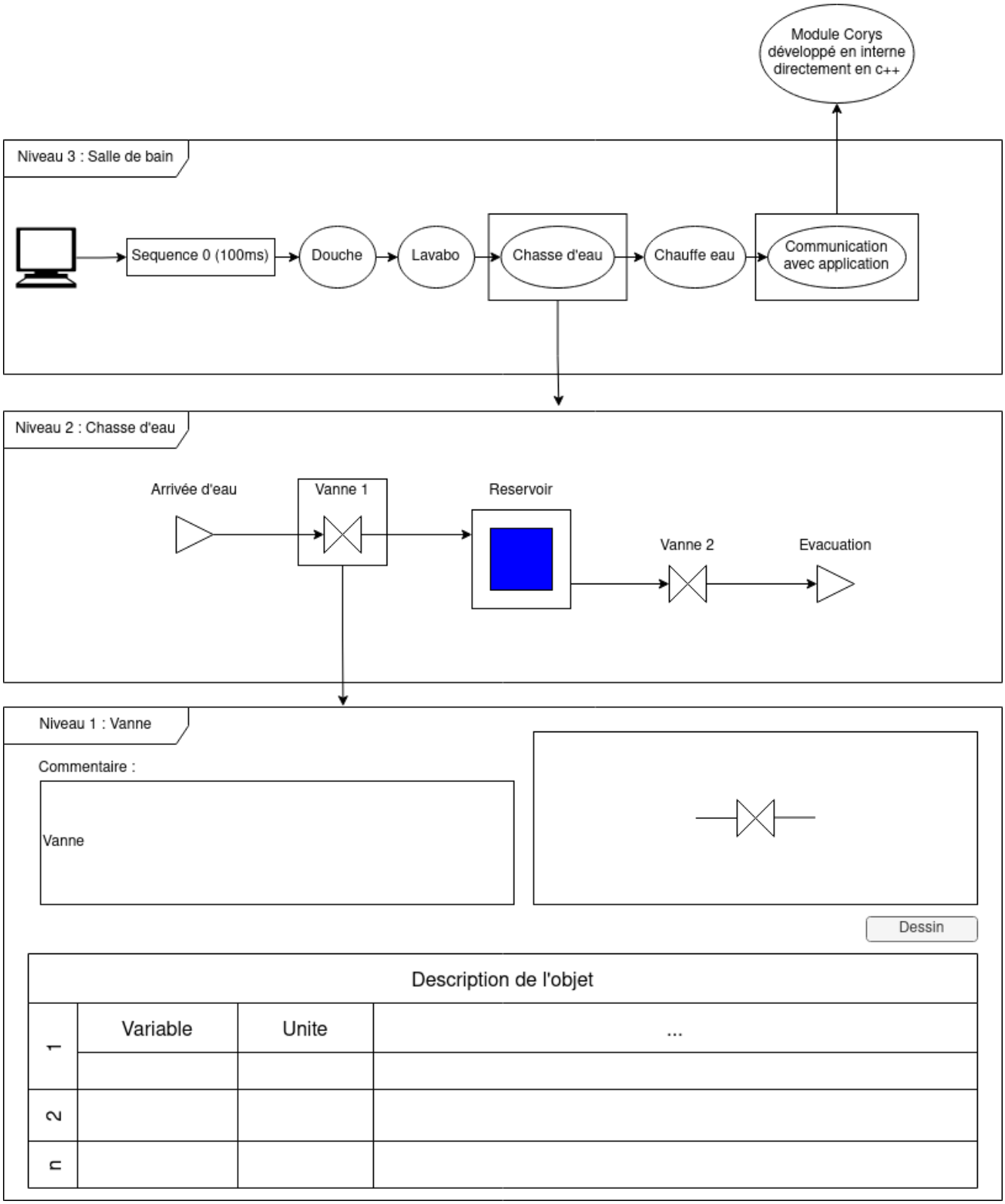

FIGURE 2.1 – Niveaux d'Alices

2.2 Open Platform Communications

OPC signifie Open Platform Communications. Il s'agit d'un protocole de communication Client-Serveur standard dans le domaine de l'industrie. Le protocole OPC a deux variantes : OPC Classic et OPC UA (architecture unifiée).

OPC Classic regroupe les protocoles OPC Data Access (DA), Alarm & Event (A&E) et Historical Data Access (HDA). Sa première version date de 1996.

OPC Unified Architecture est destiné à être une version plus moderne et multi-plateformes d'OPC Classic. Sa première version date de 2008.

À la différence d'OPC Classic, OPC UA intègre les trois fonctionnalités en un seul protocole.

Voici un petit récapitulatif des différences entre OPC Classic et OPC UA :

OPC Classic	OPC UA
Première sortie en 1996	Première sortie en 2008
Windows uniquement	Multi-plateformes
Pas de sécurité intégrée	Sécurité intégrée
Com/Dcom	TCP/IP
Configuration distance lourde	Configuration distance facile
Technologie Microsoft	Technologie non propriétaire

Si nous devons utiliser OPC Classic, cela sera uniquement OPC DA.

2.3 Besoin & Contraintes

2.3.1 Besoin

Afin d'élargir son catalogue et de pouvoir répondre à un plus grand éventail d'appels d'offres, Alices à besoin de supporter le protocole de communication OPC. Ce protocole servira principalement à accueillir des émulateurs de contrôle-commande fournis par le client.

Le contrôle-commande est l'ensemble des techniques et des technologies utilisées pour surveiller et commander des systèmes physiques ou des processus. Il prend en entrée les informations des capteurs ainsi que les commandes des opérateurs et donne en sortie les commandes des différents actionneurs. Dans le domaine nucléaire, il permet de surveiller et de contrôler les réacteurs.

Un émulateur de contrôle-commande va venir reproduire à l'identique un système ou une partie du système de contrôle-commande réel. Le but est de pouvoir se passer des systèmes réels qui sont très encombrants et coûteux.

2.3.2 Contraintes

L'implémentation d'un module OPC dans Alices est soumise à quelques contraintes :

C++ : Alices et Alices+ sont des outils entièrement développés en C++. Les outils et bibliothèques utilisés doivent alors être compatibles avec ce langage.

Mutli-plateformes : Alices est disponible sous Linux ainsi que sous Windows. Dans la mesure du possible, le module OPC doit pouvoir être exécuté sur les deux systèmes d'exploitation. Or, le protocole OPC DA est fondé sur des technologies Microsoft et n'est, par conséquent, pas supporté par d'autres systèmes d'exploitation que Windows. En revanche, OPC UA est disponible sur toutes les plateformes y compris Mac OS et Android.

Fourniture du client : Le but est de pouvoir supporter les émulateurs de contrôle-commande fournis par nos clients. Pour cela, deux options sont possibles : utiliser le même protocole que celui fourni, ou alors faire en sorte que les deux protocoles soient compatibles.

Contrainte de performance : Les simulateurs doivent fonctionner en temps réel afin de se conformer aux systèmes réels. Pour cela, le pas de temps entre chaque mise à jour des données se doit d'être faible. Il doit etre inférieur à cinq-cents millisecondes malgré la grande quantité de données manipulées.

Environnement de simulation : Bien que les simulateurs doivent fonctionner en temps réel pour reproduire de vrais systèmes, ils doivent aussi pouvoir accélérer le temps, le mettre en pause, avancer pas à pas, etc.

Contrainte temporelle : Ma mission s'étale sur toute la durée de l'alternance ce qui veut dire que, en comptant les semaines d'école et de congés, mon travail doit être terminé, fonctionnel et documenté pour le 8 septembre 2023.

2.4 Existant

En tout premier lieu et afin de réaliser ma mission, j'ai fait le constat de l'existant. OPC n'a jamais été utilisé dans Alices donc aucun élément n'existe à ce niveau-là. Je me suis alors tourné vers mes collègues de la partie hydrocarbure qui ont déjà utilisé OPC avec leur logiciel Indiss. Deux problèmes se sont alors posés : Indiss est entièrement en C# et utilise uniquement OPC DA sous Windows.

Hormis cela, tout élément existant au niveau d'OPC est externe. Ce sont principalement des kits de développement propriétaire et quelques kits open source. De plus, certains kits de développement sont certifiés par la fondation OPC (fondation en charge du protocole).

3 | Implémentation d'un module OPC dans Alices

3.1 Choix technique

Pour implémenter le protocole OPC dans Alices, nous avons plusieurs choix techniques à effectuer. Tout d'abord, nous devons choisir entre OPC DA et OPC UA. Puis choisir quelle base prendre primis : un kit de développement *open source*, propriétaire, l'existant d'Indiss+ ou partir de rien.

3.1.1 Travail préliminaire

J'ai effectué un ensemble de tâches pré-implémentation afin de chiffrer l'intégration du protocole, mais également d'acquérir une expertise en vue des premiers choix techniques. L'essence de ces tâches était de comparer OPC DA et UA, autant du point de vue des spécifications que de leur implémentation. Concernant les spécifications, on peut retrouver cette comparaison dans la partie 2.2 "Open Platform Communications". Concernant la partie implémentation, j'ai réalisé quelques prototypes grâce à deux solutions, une pour OPC DA et une pour OPC UA. Par souci de rapidité, je me suis concentré sur des kits de développement *open source*. Tester des kits de développement propriétaire nécessite une discussion avec l'éditeur de kit afin d'obtenir une licence de test.

Pour OPC UA, j'ai choisi le projet de open62541. Il est écrit en C et intègre le client et le serveur OPC. Je l'ai privilégié car c'est le kit de développement *open source* le plus visible pour OPC UA, mais surtout car le projet présente une meilleure fiabilité. Il a des sponsors importants tels que des universités et des entreprises privées ainsi que de nombreux contributeurs. Le code du projet a aussi un taux de couverture de 75% et une documentation bien construite. J'ai utilisé le serveur d'exemple fourni par le projet.

Pour OPC DA, j'ai choisi le projet de edimetia3d. C'est le seul kit de développement accessible pour OPC DA en C++. Il présente néanmoins un problème de pérennité : le créateur et mainteneur principal du projet a expliqué que les problèmes actuellement rencontrés sur le projet dépassent ses capacités et qu'il traiterait uniquement les *pull request*[1] *open source*. Cependant, pour un usage assez simple du protocole (c'est-à-dire un simple échange de variable pour mon prototype), cela fonctionne parfaitement et est simple d'utilisation. Ici, j'ai utilisé un serveur de test existant chez Indiss+.

1. Demande d'ajout du travail d'un contributeur pour un projet

3.1.2 OPC DA ou OPC UA

Toute la question est d'essayer de savoir lequel des deux sous-protocoles sera demandé en priorité. La question subsidiaire est de savoir lequel est le plus simple à implémenter.

Grâce à l'expérience partagée par les équipes du pôle "industries de procédés", nous savons que ce sont les clients qui fournissent les serveurs OPC dans tous les cas. La majorité du temps, ceux-ci sont fournis dans les deux formats, OPC DA et UA. Par conséquent, le problème de compatibilité avec les fournitures clients est moindre, indépendamment de notre choix, du moins en théorie. En conclusion, le choix s'effectue en fonction de la pérennité, la simplicité d'implémentation et la maintenabilité du module.

La solution OPC DA tend à être dépréciée, et n'offre donc aucune pérennité. OPC UA est donc à privilégier. En outre, OPC UA intègre nativement une surcouche de sécurité et de chiffrage. Cet aspect est capital en sachant que la sécurité informatique est une problématique majeure.

Grâce au kit de développement *open source* ainsi que mon prototypage, OPC UA semble plus simple à implémenter. Ici, la surcouche de sécurité nuit légèrement à la simplicité d'implémentation, mais cela est négligable à côté de l'avantage qu'elle présente.

Concernant la maintenabilité, le kit de développement compte de nombreux sponsors et presque 250 contributeurs. Cela augmente son indice de confiance vis-à-vis de la pérennité du projet. Plusieurs employés ont utilisé OPC DA, ce qui leur offre un certain savoir-faire sur ce protocole. Par contraste, ils n'ont jamais utilisé OPC UA (même si une partie des connaissance de OPC DA est conservée).

En somme, le seul gros défaut de OPC UA est la création d'une possible dette technique. Hormis cela, il ne présente que des avantages par rapport à OPC DA. Nous avons donc choisi d'employer OPC UA principalement du fait de l'aspect vieillissant de OPC DA. Il est cependant crucial de bien documenter le travail effectué pour éviter une dette technique autant que faire se peut.

3.1.3 OPC UA avec open62541

Finalement, pour les différentes raisons évoquées dans les deux parties précédentes, notre choix s'est porté sur OPC UA avec le projet d'implémentation pour C et C++ de open62541. L'avantage de l'*open source* est l'accès au code source pour comprendre son fonctionnement, le modifier au besoin et en avoir un usage gratuit. Grâce à la licence MPL, on peut utiliser et modifier ce que fournit le projet sans contaminer le code propriétaire Corys.

3.2 Implémentation

Afin de réaliser une implémentation du protocole OPC dans Alices, même à des fins de prototypage, il nous faut un certain nombre d'éléments :

— Configuration du serveur (adresse, utilisateur et mot de passe) ;
— Connexion au serveur ;
— Déclaration des variables à échanger ;
— Échanges de variables entre Alices et OPC et inversement ;
— Système de remontée des erreurs.

L'implémentation du protocole OPC se fait donc dans un Module niveau 3 d'Alices (cf. 2.1 Atelier Logiciel Intégré de Conception et d'Étude des Simulateurs). Un module niveau 3 est défini par une classe C++ contenant des méthodes standards. Celles-ci définissent son comportement : sa création, son initialisation, son comportement à chaque pas de temps, etc.

En plus de cela, j'ai défini une classe *OPCnode* facilitant la gestion d'un nœud OPC. Elle permet simplement de stocker un nœud OPC, acquérir sa valeur, lui définir une nouvelle valeur, etc.

Les classes, attributs et méthodes sont régis par des conventions de nommage spécifiques à Alices. Par exemple, le préfixe "m" correspond à "membre", autrement dit une variable dans une classe. Le "p" signifie "pointeur". Dans la pratique, on utilise le *camelcase* en C++ plutôt que le *snakecase*.

Afin de faciliter l'implémentation et la maintenance du module OPC et de la classe *OPCnode*, j'ai réalisé un diagramme de classe décrivant les deux classes et leurs liens d'héritage et de cardinalité.

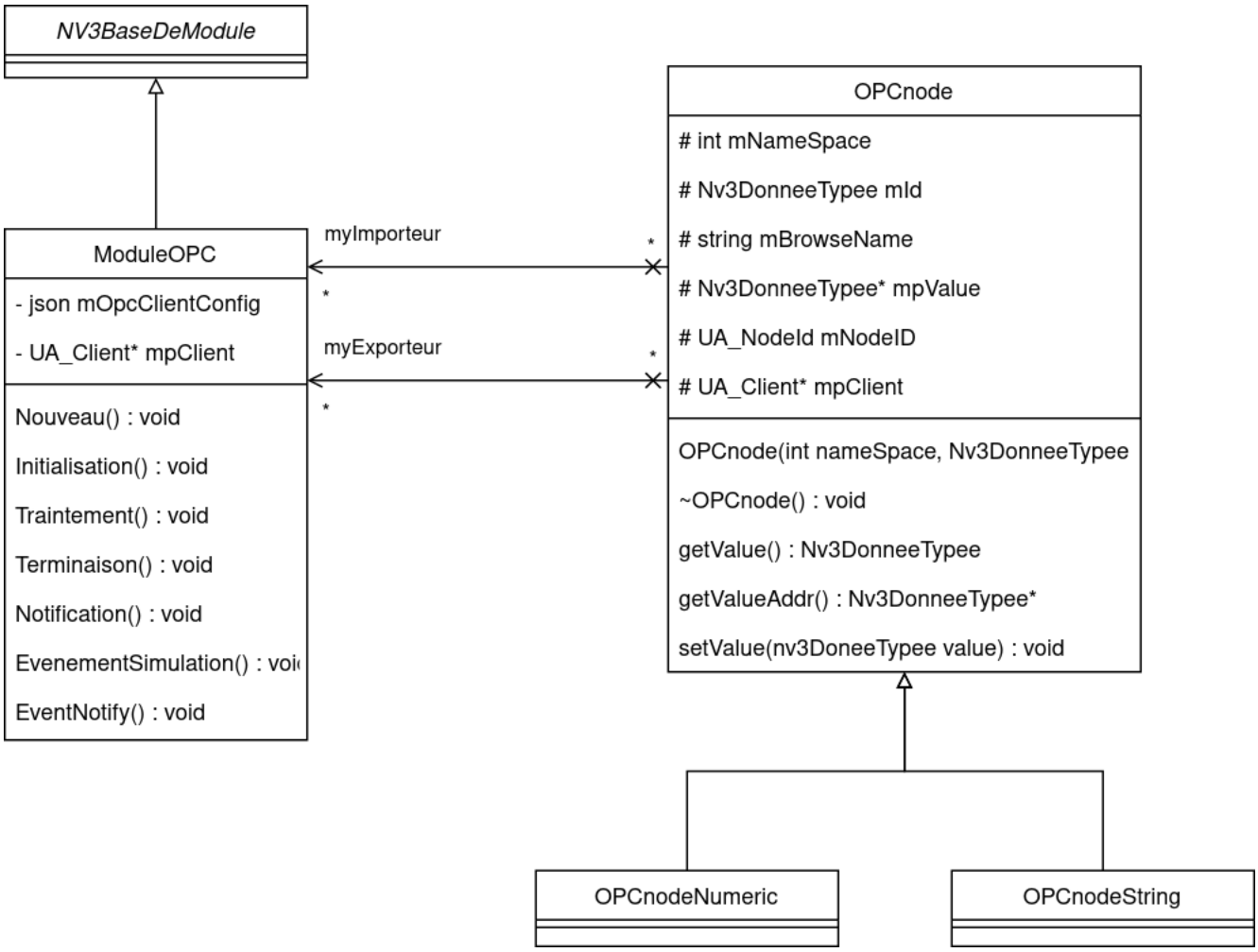

FIGURE 3.1 – Diagramme de classe pour l'implémentation du module OPC

OPCnode est une classe abstraite ; c'est *OPCnodeNumeric* et *OPCnodeString* qui sont instanciées, car l'identification d'un nœud OPC s'effectue à l'aide d'un identifiant numérique ou d'une chaîne de caractères.

3.2.1 Visualiseur du serveur OPC

Afin de faciliter les tests et le débogage, il est important de voir l'état des variables sur le serveur OPC. Dans ce but, j'ai développé un script Python me permettant de visualiser le serveur OPC. Je me suis servi du projet de FreeOpcUa, projet recommandé par open62541. Quand j'en ai eu la nécessité, leur GUI [2] n'était pas au point et il m'était impossible de me connecter au serveur. Je me suis donc servi de leur implémentation en Python pour afficher les données du serveur sous la forme d'un arbre.

J'ai créé une fonction récursive pour afficher correctement cet arbre et ses indentations. Une trace de l'exécution est disponible en annexe.

2. Graphic User Interface, application avec une interface graphique

3.2.2 Configuration du serveur

L'un des rôles essentiels du module OPC est de pouvoir se connecter à un serveur OPC. Pour cela, nous avons besoin de certaines informations comme l'adresse du serveur, le nom d'utilisateur et mot de passe à utiliser. Nous avons décidé de stocker ces informations dans un fichier de configuration Json par conformité avec le reste d'Alices. En effet, il sera aisé de faire un utilitaire à terme pour remplir ce fichier de configuration.

```
{
  "configClient": {
      "url": "opc.tcp://localhost:4840",
      "user": "user1",
      "passwd": "password"
  }
}
```

Pour *parser* le fichier Json, nous utilisons le projet de "nlohmann" Json for Modern C++ comme il est déjà présent dans Alices et très simple d'utilisation. Le type *json* provient de la bibliothèque du projet.

3.2.3 Connexion au serveur

La connexion au serveur est implémentée par open62541. La fonction de connexion renvoie un code d'erreur indiquant si la connexion a été réussie. En fonction de son succès, le programme remonte une erreur via le système de gestion d'erreurs d'Alices.

3.2.4 Déclaration des variables à échanger

Afin de transmettre des informations, Alices dispose d'une zone d'échanges avec des importeurs et des exporteurs. Dans notre utilisation, les exporteurs sont des variables qu'Alices met à disposition du serveur via le module OPC. Les importeurs, quant à eux, sont des variables qui récupèrent leurs valeurs depuis le serveur. Alices est muni des droits d'écriture sur les exporteurs mais seulement de lecture sur les importeurs, et vice versa pour le serveur. Les variables que nous échangerons ici seront uniquement des *OPCnode*.

Pour définir quelles variables sont des importeurs et lesquelles sont des exporteurs, on utilise le fichier de configuration Json avec le nom de la variable dans Alices d'un côté et l'identifiant de la variable sur le serveur OPC de l'autre.

```
{
  "importeur" : {
    "pressure" : "pression",
    "temperature" : "1548"
  },
  "exporteur" : {
    "viscosite" : "viscosite",
    "temperature" : "1549"
  }
}
```

Avoir les identifiants des nœuds est une première étape nécessaire à l'instanciation des *OPCnode*. Ce sont ces *OPCnode* qui vont être définis en tant qu'importeurs ou exporteurs. Afin de décrire toute la phase de déclaration des importeurs et des exporteurs, on présente ci-dessous l'algorigramme de la déclaration des importeurs, identique à celui des exporteurs.

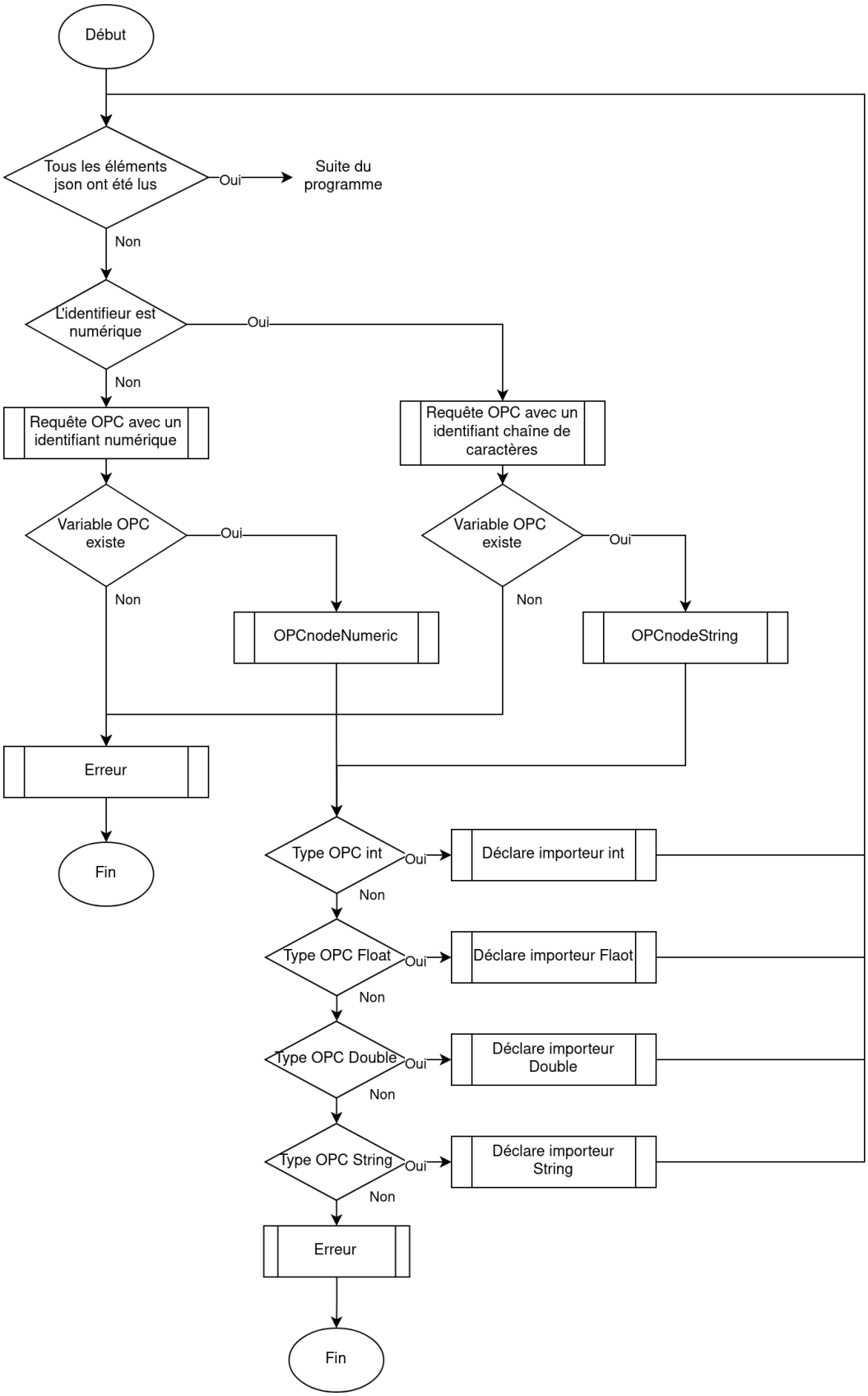

FIGURE 3.2 – Algorigramme de la déclaration d'importeur

3.2.5 OPCnode

La classe *OPCnode* permet de simplifier la communication avec le serveur OPC en stockant les nœuds OPC. Son rôle principal est de résoudre un important problème de typage. Les types des variables OPC ne sont pas connus lors de la compilation. On peut cependant obtenir le type d'une variable via une requête vers le serveur OPC pendant l'exécution du programme. La valeur est alors *cast* dans le bon type avant d'être stockée dans une instance de la classe *Nv3DonneeTypee*. C'est une classe native d'Alices permettant de définir des variables dont on ne connait pas le type avant l'exécution.

En outre, cette classe facilite la récupération et la modification des variables sur le serveur OPC. Elle maintient également les avantages de la gestion des types tout en prenant en charge l'intégralité du processus de requête auprès du serveur.

Le fonctionnement de cette classe est assez simple. Lors de son instanciation, elle prend l'identifiant du nœud comme paramètre. Lors de sa construction, elle envoie une requête OPC dans le but de récupérer les attributs du nœud : son type, son nom, sa valeur, etc. Grâce au type et à une instruction *switch/case* (disjonction de cas), on peut alors définir correctement les variables dans *OPCnode*.

3.2.6 Gestion d'erreurs

La gestion des erreurs dans Alices est commune à tous les modules. Une levée d'erreurs contient plusieurs attributs : sa gravité (avertissement, erreur, erreur critique, etc.), la fonction dont provient l'erreur et le contenu du message d'erreur. Le module OPC remonte des erreurs dans les cas suivants :

— le module n'a pas réussi à se connecter au serveur ;
— il est impossible de lire ou écrire une variable ;
— il y a un problème de type (type non supporté, notamment).

L'erreur est renvoyée à l'utilisateur sous la forme d'un pop-up comme sur la figure ci-dessous.

FIGURE 3.3 – Pop-up d'erreur dans Alices

3.2.7 Cycle de vie du module OPC

Pour visualiser le cycle de vie du module OPC, j'ai élaboré le diagramme de séquences ci-dessous. Il illustre le cycle de vie du module OPC, ses diverses étapes (initialisation, traitement, terminaison, etc.) et les intéractions entre les entités du simulateur (classes, simulateur, serveur, etc.). Il présente un intérêt considérable autant pour la phase de conception que pour celle de la maintenance.

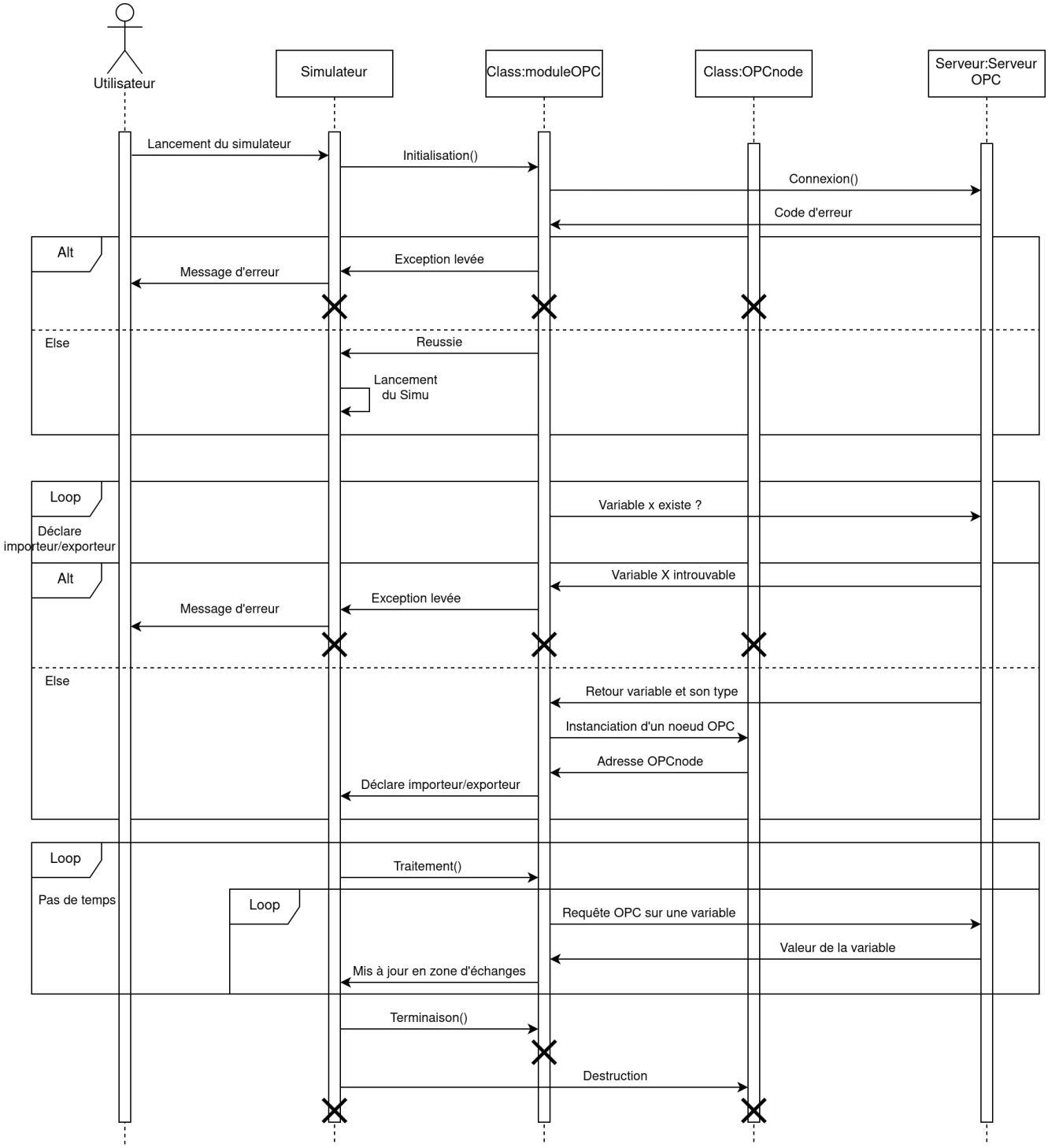

FIGURE 3.4 – Diagramme de séquences du module OPC

Au cours du cycle de vie du module, il faut bien penser à détruire [3] toutes les instances de *OPCnode* contenues dans le vecteur afin d'éviter des fuites de mémoire. Pour ce faire, on fait appel au destructeur de *OPCnode*. Cette opération a lieu dans la fonction *Terminaison()* du ModuleOPC. On profite de cette étape pour détruire le client OPC.

3.2.8 Compilation

On exécute la compilation d'Alices et des modules associés grâce à un utilitaire développé par l'équipe R&D. Celui-ci se sert de CMake pour compiler Alices. Il est également possible de compiler séparément un module en particuler. Cette fonctionnalité s'avère très utile dans mon cas car elle m'évite de recompiler l'emsemble du logiciel à chaque modification.

Pour chaque module, il y a trois fichiers de configuration servant à la compilation.
- Un fichier général, regroupant le nom, le chemin vers les fichiers sources du code et les dépendances ;
- Un fichier spécifique à Windows ;
- Un fichier spécifque à Linux.

C'est dans les fichiers spécifiques aux systèmes d'exploitation que j'ai défini les chemins vers la bibliothèque open62541.

3.2.9 Débogage

Le débogage est une partie importante du travail de développement informatique. Pour réaliser cette tâche, je me suis servi du logiciel *VScode*. Il a l'avantage de remplir l'ensemble des besoins nécessaires au débogage de manière très intuitive. *VScode* comporte aussi des fonctionnalités avancées telles que le lancement du logiciel à déboguer ou un visualisateur pour les élements nouvellement conçus.

Le lancement depuis *VScode* est très utile, par exemple, lorsqu'il faut déboguer une phase d'initialisation. On peut aussi joindre le débogueur au logiciel Alices lorsque ce dernier est déjà lancé. On configure tout cela dans un fichier Json (cf. la documention *VScode*).

Le visualiseur permet de configurer la manière dont sont affichés les éléments à surveiller lors du debogage (objets, listes, variables, etc.). On peut les configurer à l'aide de fichiers XML, appelés "natvis". Ils permettent de changer l'affichage de certains éléments, comme les classes. Cela m'a rendu service pour régler certains bogues difficiles à cerner. Voici un exemple de configuration navis pour *OPCnode* :

3. Opération consistant à libérer la mémoire occupé par l'objets

```
<Type Name="OPCnode">
  <Expand>
          <Item Name="[mNameSpace]">mNameSpace</Item>
          <Item Name="[mId]">mId</Item>
          <Item Name="[mBrowseName]">mBrowseName</Item>
          <Item Name="[mpValue]">mpValue</Item>
          <Item Name="[mNodeID]">mNodeID</Item>
  </Expand>
</Type>
```

3.2.10 Souscription

Le système de souscription est un mécanisme qui permet au client de se faire notifier par le serveur en cas de changement de variable. Cette fonctionnalité est très intéressante pour des questions de performances. Cela évite au module de faire une requête par variable à chaque pas de temps (généralement 500ms) pour vérifier d'éventuels changements de valeurs. Cependant, bien que cette fonctionnalité présente un intérêt notable en termes de performances, elle n'est pas considérée comme une priorité car elle n'est pas indispensable au fonctionnement du module. J'ai réalisé un prototypage pour comprendre son mécanisme et évaluer sa complexité.

La première étape pour surveiller une variable OPC est de créer une fonction *callback*. Elle permet de définir quels changements seront effectués après le changement de variable. Ici, il suffira de mettre à jour la variable correspondante pour Alices. Ensuite, on fait appel à deux fonctions de la bibliothèque : l'une crée la requête qui permettra de surveiller la variable et l'autre active la souscription.

Son intégration risque d'être relativement fastidieuse car elle nécessite une fonction callback par variable souscrite.

3.2.11 Documentation

Afin de rendre mon travail maintenable, je suis actuellement en train d'écrire une documentation la plus complète possible à propos de l'utilisation du module, mais surtout sur son développement. L'idée est de synthétiser l'expérience acquise sur OPC, la bibliothèque open62541 et son intégration dans Alices. Ce rapport servira en partie pour la documentation sur le contenu et les choix de réalisation de l'implémentation.

Tout au long du développement, j'ai tenu à faire en sorte de rédiger mon code le plus lisiblement possible. Il est important de respecter les bonnes pratiques préconisées par Corys :

- utiliser la bonne taille d'indentation ;
- ne pas dépasser la taille maximum d'une ligne ;
- nommer correctement les éléments (cf. règles de nommage) ;
- écrire ses commentaires au format Doxygen.

Doxygen est un logiciel qui permet d'interpréter un format spécifique du commentaire et une partie du code pour créer une documentation automatiquement.

J'ai également écrit une spécification du module OPC en amont du développement décrivant ses fonctionnalités. Tous les travaux de documentation doivent être effectués en anglais.

3.2.12 Tâches et améliorations futures

Durant ma mission, je me suis concentré sur la partie fonctionnelle : j'ai fait en sorte que les éléments nécessaires au module fonctionnent. Il reste certaines tâches et améliorations que je n'ai pas encore pu faire à l'heure de l'écriture du présent rapport. Voici les améliorations qu'il reste à effectuer dans l'ordre de priorité :

— un système de log pour le client OPC ;
— l'intégration sous Linux ;
— une gestion des tests automatique ;
— la réalisation d'un test en situation réelle ou partielement réelle ;
— le système de souscription (cf. 3.2.8 Souscription).

La mise en place du système de log devrait être assez simple. Il suffit d'écrire sur la sortie standard qui est redirigée vers un fichier de log. Il faut prendre soin de verrouiller l'écriture avec un verrou numérique qui évite le chevauchement des fils d'exécution (mutex).

Le travail d'intégration sous Linux consiste principalement à compiler la bibliothèque open62541 sous Linux et à éditer correctement les liens au système de compilation d'Alices. La tâche de compilation de la bibliothèque s'est avérée davantage fastidieuse sous Windows que sous Linux. En outre, la documentation est plus exhaustive sous Linux.

La gestion des tests automatiques rentre dans le processus d'intégration et de déploiement continus (CI/CD) de Gitlab. Cela permet de prévenir un grand nombre d'erreurs nuisibles à l'exécution, la lisibilité ou la maintenabilité du programme. Les tests sont écrits en Python avec la bibliothèque de test nommée *pytest*.

Le test réel ou partiellement réel consisterait à faire fonctionner un simulateur en condition avec des modèles [4] concrets et un serveur OPC qui se comporte comme un émulateur. Pour l'instant, je me sers uniquement d'un serveur de test qui répond à mon usage, qui est de faire de simples échanges de variables. Le but du test est de mettre en lumière les problèmes et performances à pleine échelle. Cela permettra, de surcroît, de se doter d'une démonstration concrète.

4. Section de centrale

3.3 Difficultés et facilités

3.3.1 Le langage C++

Le C++ comporte des avantages évidents de performance, c'est pourquoi il est très prisé en simulation. Néanmoins, il est plus complexe que d'autres langages comme le Python. Il demande un temps d'apprentissage nettement plus élevé. De plus, le C++ est très exigeant et son débogage est plus épineux. Parfois, une petite erreur au début d'un programme est révélée bien plus loin au cours de l'exécution (par exemple lors d'une mauvaise instanciation). L'usage de ce langage et sa complexité accrue m'a posé des difficultés.

3.3.2 Acclimatation avec l'environnement d'Alices

Alices et son environnement sont aussi vastes que complexes. Cela est parfaitement logique pour un éditeur de simulateur nucléaire. Par conséquent, il faut un temps d'adaptation important pour se greffer à ce système. L'utilisation des outils d'Alices demande souvent de l'aide et beaucoup de pratique. De plus, comme évoqué au point précédent, l'usage du C++ rend la lecture de code complexe en comparaison avec des langages plus faciles d'accès.

3.3.3 Documentation de la bibliothèque open62541

La documentation de la bibliothèque open62541 n'est pas exhaustive. Elle est assez riche concernant l'utilisation des exemples fournis mais devient vite lacunaire à propos de fonctions appliquées à un autre cadre. Les informations les plus complètes sont disponibles à travers la lecture du code source. Cette tâche est parfois pénible et compliquée mais très fiable. Le code est toutefois bien ordonné et très lisible.

3.3.4 Docuemntation C++

Le C++ est un langage complexe. Cela est contrebalancé par une documentation complète et extrêmement claire. Celle-ci permet de vérifier aisément l'utilisation d'une fonction, la syntaxe d'un mot-clé, etc. La définition des concepts abstraits est également bien expliquée, même si ceux-ci sont plus subtils.

3.3.5 L'équipe R&D

Notre équipe R&D fonctionne de manière fluide et efficace grâce à un très fort esprit de collaboration. Chacun est à l'écoute, ouvert aux sollicitations et prêt à offrir une précieuse assistance. Cette atmosphère de soutien mutuel permet d'aborder les défis collectivement, en exploitant les compétences individuelles et les bienfaits de la coopération. Lorsque je me retrouve bloqué sur un élément précis, je peux demander facilement de l'aide pour me débloquer.

4 | Planification & Conclusion

4.1 Planification

Lors du point entreprise, j'ai présenté le planning prévisionnel ci-dessous :

Semestre 1				Semaines	38	40	41	43	44	46	48	50	51	52	2	4
Numéro	Nom Tâche	Durée en semaine	Date début	Date fin												
1	Renseignements sur OPC	2	19/09/22	07/10/22												
2	Hello World VBA	1	10/10/22	14/10/22												
3	Rédaction de la spec OPC	1	24/10/22	28/10/22												
4	Recherche d'un tool kit	2	31/10/22	18/11/22												
5	Hello World OPC DA	2	28/10/22	23/1222												
6	Congés	1	26/12/22	03/01/23												
7	Hello World OPC UA	2	09/01/23	27/01/23												
Semestre 2				**Semaines**	**6**	**7**	**9**	**11**	**13**	**15**	**16**	**18**	**19**	**20**	**22**	**24**
Numéro	Nom Tâche	Durée en semaine	Date début	Date fin												
8	Évaluation des côut	1/2	06/02/23	08/02/23												
9	Choix techniques	1/2	08/02/23	10/02/23												
10	Module niveau 3 minimal	4	13/02/23	17/03/23												
11	Module OPC	4	10/04/23	19/05/23												
12	Congés	1	01/05/23	05/05/23												
13	Tests approfondis	2	29/05/23	16/06/23												
Semestre 2 bis				**Semaines**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**			
Numéro	Nom Tâche	Durée en semaine	Date début	Date fin												
14	Documentation	2	26/06/23	07/07/23												
15	Révision / debogage	4	10/07/23	05/08/23												
16	Congés	3	07/08/23	27/08/23												

FIGURE 4.1 – Planning prévisionnel

Finalement, quelques écarts entre les prévisions et la réalité sont à noter. La création d'un module niveau 3 minimal n'a pris qu'une semaine. En revanche, le développement du module m'a pris beaucoup plus de temps que prévu en raison des difficultés mentionnées au point 3.3 "Difficultés et facilités". Le déroulement réel de mon travail est présenté ci-dessous :

Semestre 1				Semaines	38	40	41	43	44	46	48	50	51	2	4
Numéro	Nom Tâche	Durée en semaine	Date début	Date fin											
1	Renseignements sur OPC	1	19/09/22	23/09/22											
2	Hello World VBA	1	3/10/22	7/10/22											
3	Rédaction de la spec OPC	1	10/10/22	14/10/22											
4	Recherche d'un tool kit	1	24/10/22	28/10/22											
5	Hello World OPC DA	3	31/10/22	2/12/22											
7	Hello World OPC UA	3	12/12/22	13/01/23											
8	Évaluation des côut	1/2	23/01/23	25/01/23											
9	Spec	1/2	25/01/23	27/01/23											
Semestre 2				**Semaines**	**6**	**7**	**9**	**11**	**13**	**15**	**16**	**18**	**19**	**20**	**22**
Numéro	Nom Tâche	Durée en semaine	Date début	Date fin											
10	Choix technique	1	06/02/23	10/02/23											
11	Module niveau 3 minimal	1	13/02/23	17/02/23											
12	Module OPC	15	27/02/23	18/08/23											
Semestre 2 bis				**Semaines**	**24**	**26/27/28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**	
Numéro	Nom Tâche	Durée en semaine	Date début	Date fin											
12	Module OPC	15	27/02/23	18/08/23											
13	Congés	3	26/06/23	14/07/23											
13	Documentation	3	21/08/23	08/09/23											

FIGURE 4.2 – déroulement réel

4.2 Conclusion

Cette année au sein de Corys m'a permis d'avoir une première expérience professionnelle longue. Mener un projet de sa conception à sa réalisation avec de réelles contraintes. J'ai pu aussi découvrir le travail dans une équipe de recherche et développement. J'ai également beaucoup appris sur le domaine du nucléaire et ai abordé le fonctionnement d'une centrale.

Au cours de ma mission, j'ai beaucoup gagné en autonomie, perfectionné ma connaissance du C++ et découvert le protocole de communication OPC. Grâce à mon rapport, je suis devenu très à l'aise avec l'outil LaTeX et créé mon premier script d'intégration continue afin de compiler mon rapport.

Les enseignements de la Licence Professionnelle ont également été très bénéfiques. J'ai beaucoup appris sur l'administration et la sécurité en cours et en appliquant ces principes à titre personnel via le serveur que j'administre. J'ai grandement sécurisé celui-ci, dockeriser Nextcloud, crée un serveur de jeu (Minecraft) et mis en place un système de RAID. Il me reste à mettre en place un VPN [1] afin de me connecter à mon serveur depuis l'extérieur en toute sécurité et de mettre un système d'alerte en cas de panne.

En résumé, Corys ainsi que de l'équipe éducative de l'IUT m'ont beaucoup appris.

1. Virtual Private Network, réseau virtuel privé

Glossaire

Bibliothèque En informatique, une bibliothèque logicielle est une collection de routines et fonctions qui peuvent être déjà compilées et prêtes à être utilisées par des programmes. 5

C++ Langage de programmation compilé orienté objet datant de 1985 pour sa première version. Ses bonnes performances, et sa compatibilité avec le C en font un des langages de programmation les plus utilisés dans les applications où la performance est critique. 5

Callback une fonction dite "callback" est une fonction passée en argument d'une autre fonction afin d'être appelée par celle-ci. 16

Camelcase Convention de nomage avec des majuste : commeParExemple. Cette convention de nomage est stanrard en C et C++. 8

Cast Conversion explicite dans un type de données souhaité. 13

CMake CMake est un outil permettant de compiler un programme. 15

Fuite de mémoire Une fuite de mémoire se produit lorsque le développeur oublie de libérer une zone mémoire non utilisée. Cela peut avoir pour conséquence le plantage du programme. 15

Gitlab Gitlab est un logiciel libre de forge logiciel basé sur git. Il sert de gestionaire de version, de plateforme de test, de wiki et d'outils à l'intégration continue. C'est un concurent direct à GitHub. 17

Json Format de structure de donnée. Une sorte de version morderne de l'XML. 10, 15

MPL Mozilla Public License. MPL est une licence libre non contaminante créée par Netscape. Firefox ou LibreOffice notamment utilise la licence MPL. 7

Mutex Un mutex est un verrou numérique protégeant une partie d'un programme de la concurrence des *thread*. Un thread permet de paralléliser une partie d'un programme. 17

Nextcloud Nextcloud est une alternative libre et open source à Google drive. 20

Opérateur Il est en charge du matériel, de sa réception, de sa maintenance. Il garantit, à ce titre, la sécurité du site. C'est un poste à grande responsabilité, notamment en matière de sécurité. 1, 4

Parser Récupérer les informations depuis une structure de données. 10

Pointeur En informatique, un pointeur est l'adresse mémoire d'un élément. 8

RAID Redundant Arrays of Inexpensive Disks : ensemble de techniques de virtualisation du stockage permettant de répartir des données sur plusieurs disques durs afin d'améliorer soit les performances, soit la sécurité ou la tolérance aux pannes de l'ensemble du ou des systèmes. 20

Snakecase Convention de nomage avec des "_" : comme_par_exemple. Cette convention de nomage est stanrard en Python. 8

Sortie Standard La sortie standard est le flux de sortie dans lequel les données sont écrites par le programme. Les données sont habituellement écrites à l'écran, à moins d'une redirection. 17

Bibliographie/Webographie

[1] Page wikipedia de corys, 2017. Dernière modification 21 mai 2022.

[2] Page wikipedia du controle-commande, 2010. Dernière modification 4 juin 2020.

[3] Page wikipedia du y greneblois, 2007. Dernière modification 3 janvier 2023.

[4] Site c plus plus reference, 1999. Dernière modification 30 janvier 2023.

[5] Fondation OPC. Site et documentation de la fondation opc, 1997. Dernière modification en 2023.

[6] Corys. Document interne corys.

[7] Page wikipedia de la sortie standard, 2008. Dernière modification le 28 janvier 2021.

[8] Page wikipedia du raid (informatique), 2002. Dernière modification le 1 juillet 2023.

Annexes

A | Corys

A.1 Organigramme

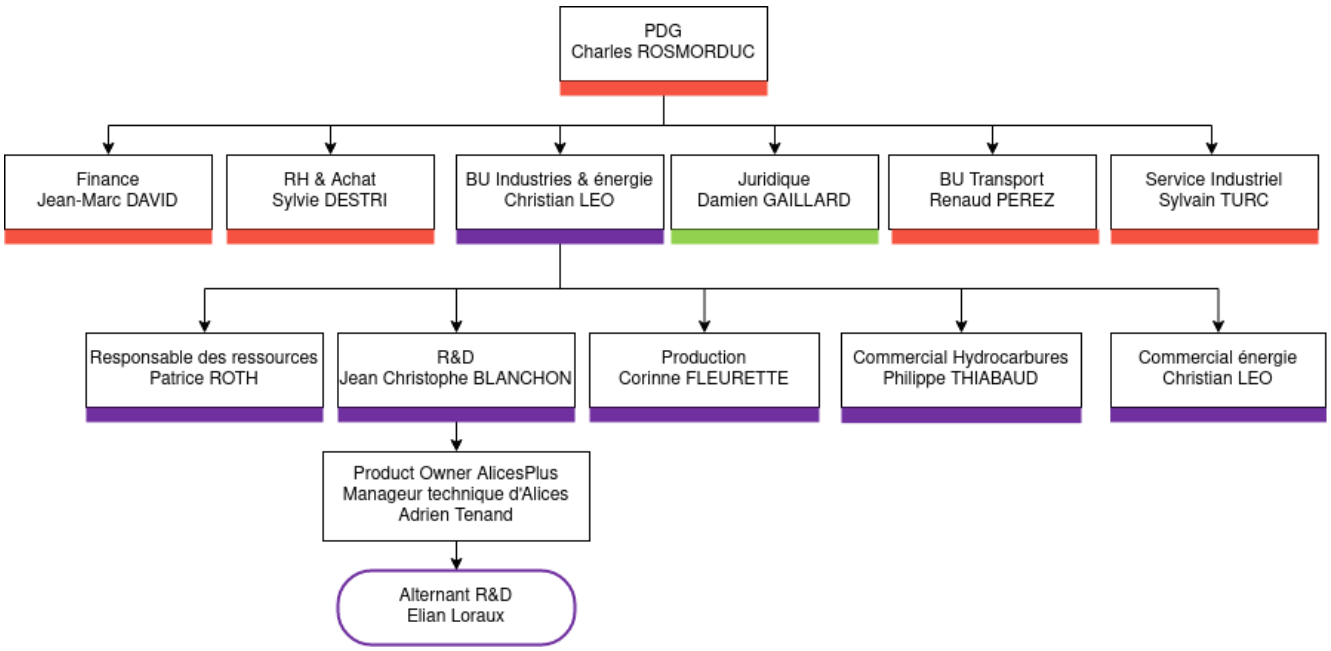

Figure A.1 – Organigramme

B | Alices

B.1 Captures d'écran d'Alices

Voici quelques captures d'écran d'Alices :

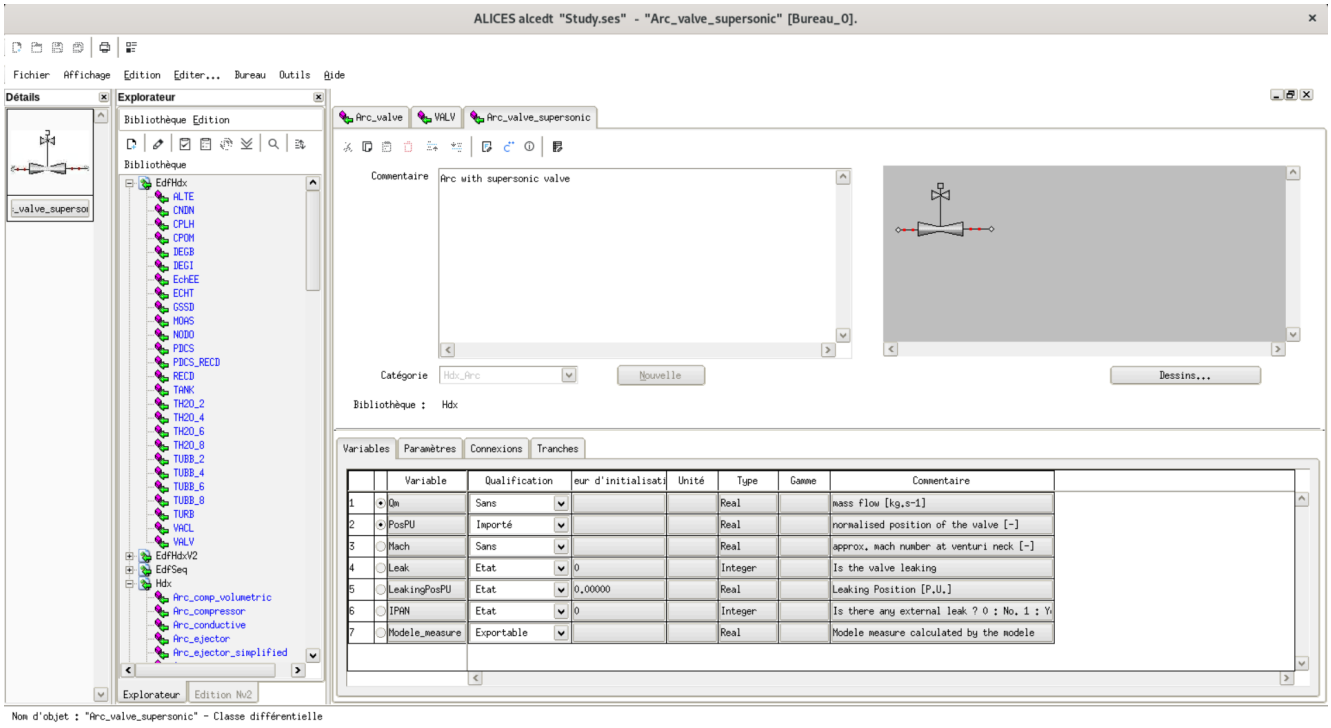

FIGURE B.1 – Niveau 1

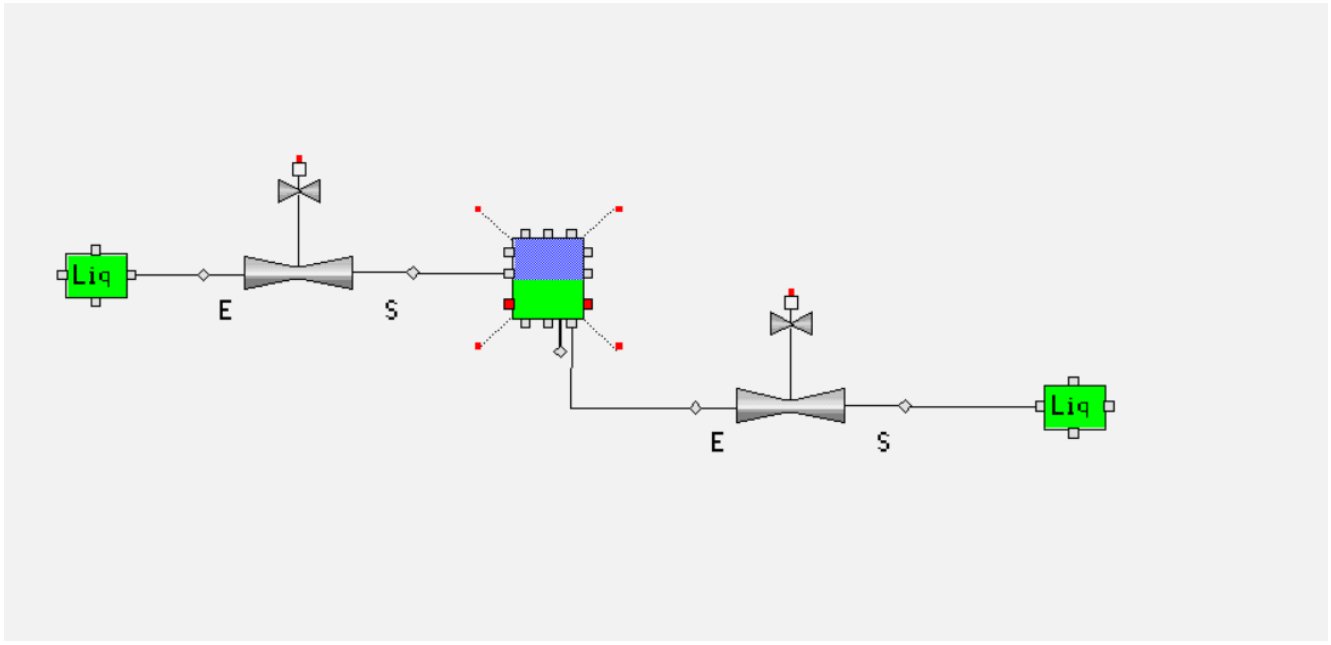

FIGURE B.2 – Niveau 2

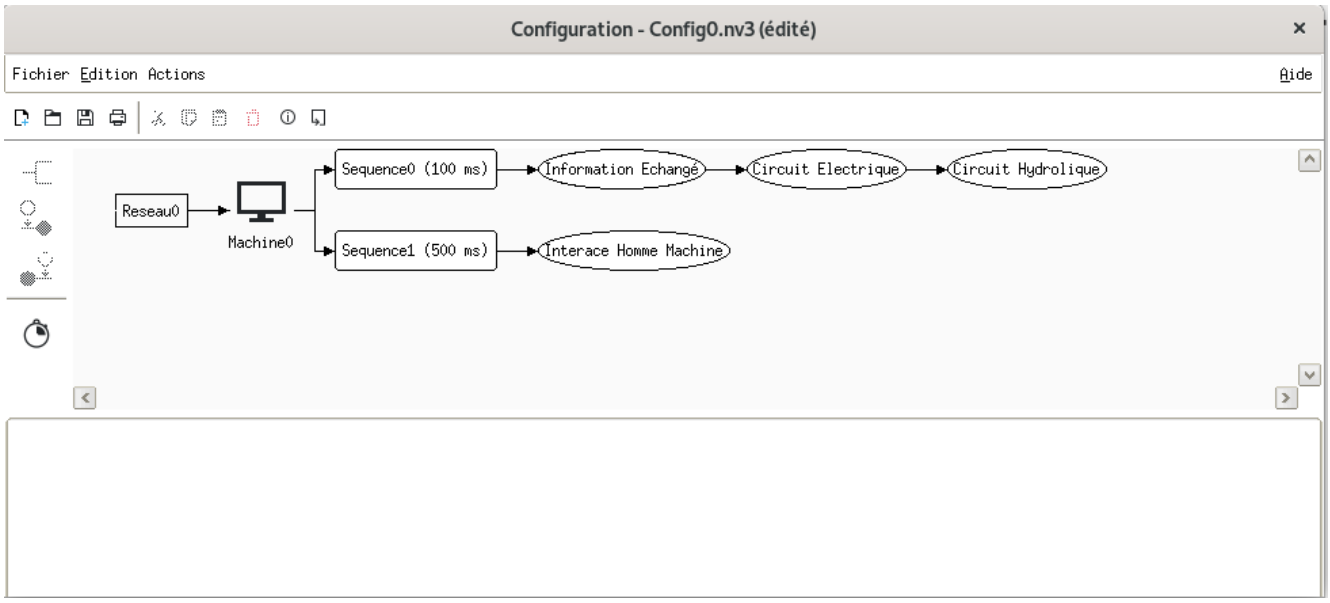

FIGURE B.3 – Niveau 3

C | Trace d'exécution

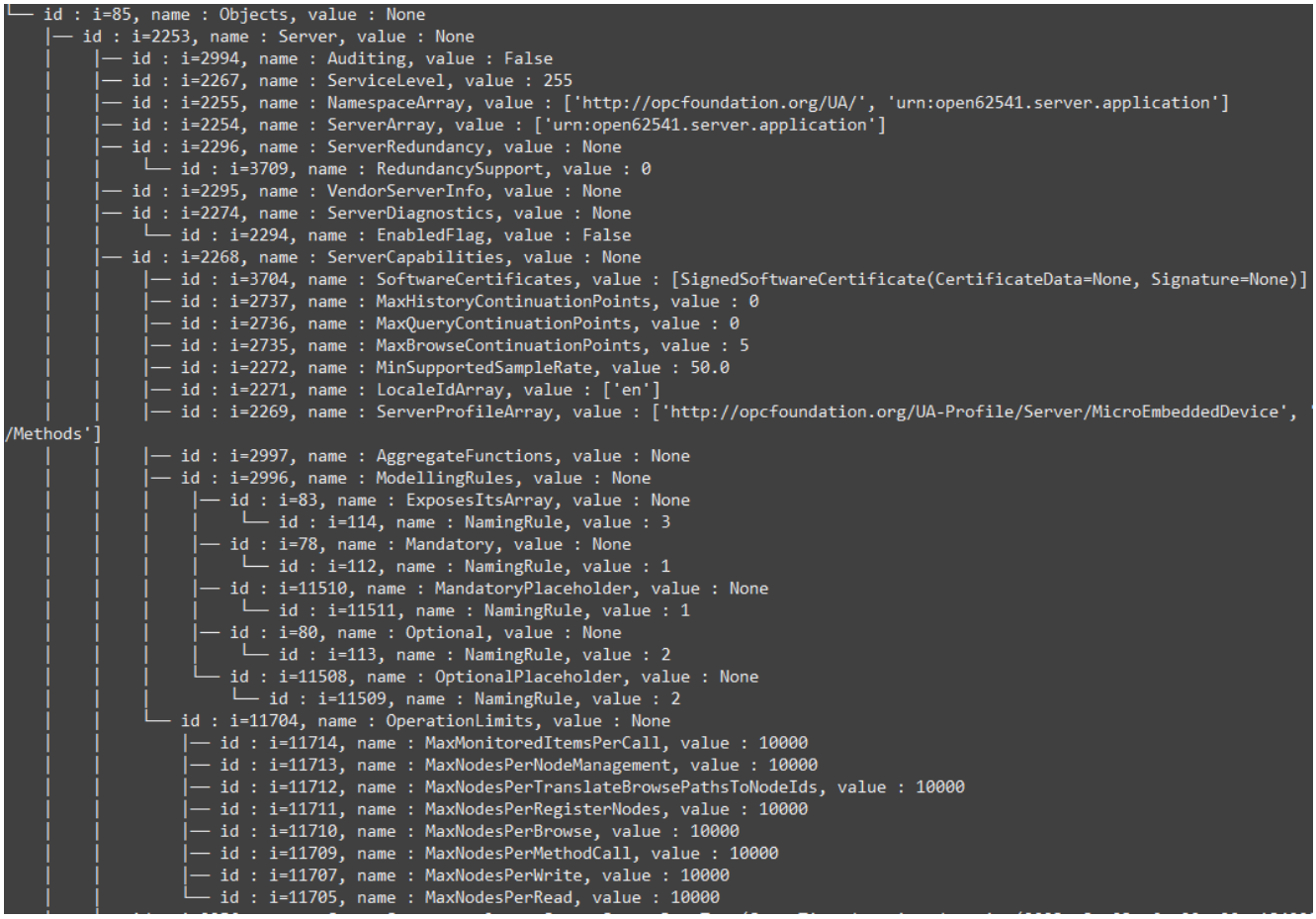

FIGURE C.1 – Trace d'execution du visualiseur du serveur OPC

5 | Résumé et Abstract

5.1 Résumé du rapport

Mon alternance s'effectue au sein de Corys, dans l'équipe recherche et développement d'Alices (Atelier Logiciel Intégré de Conception d'Études des Simulateurs). Corys est une entreprise née à Grenoble spécialisée dans la réalisation et le développement de simulateurs pour le nucléaire, le transport ferroviaire et les procédés industriels. Alices est la suite logicielle spécialisée pour le nucléaire. Ma mission est de développer un module OPC (Open Platform Communications) afin de pouvoir supporter ce protocole de communications dans Alices. L'idée est de disposer d'un meilleur catalogue pour répondre à un plus large éventail d'appels d'offres. L'implémentation du protocole OPC est soumise à quelques contraintes comme son langage de programmation, ici C++, le souci de performance dû à la simulation en temps réel, la portabilité d'Alices sous Windows et Linux, etc. Pour réaliser l'implémentation du protocole OPC, nous avons du choisir entre deux version d'OPC : OPC DA et OPC UA. Pour ce faire, j'ai prototypé un échange de variable avec OPC DA et OPC UA. Notre choix, c'est porté sur OPC UA. Ensuite, j'ai commencé l'implémentation de OPC dans Alices en réalisant, dans l'ordre : un Module Alices basique, un visualisateur OPC, la connexion au serveur OPC, la déclaration des variables à échanger, la création d'une classe *OPCnode*, la gestion d'erreurs puis la documentation. J'ai aussi décrit les tâches restant à effectuer ainsi que les améliorations futures et expliqué mes difficultés.

5.2 Abstract

Development of an Open Platform Communications (OPC) Module for the Alices Software Suite

Elian Loraux

Abstract : In order to develop nuclear control simulators, Corys provides a simulator design and execution tool called Alices. Alices (Integrated Software Workshop for the Design of Simulator Studies) sometimes includes control command emulators provided by our customers that communicate through the OPC (Open Platform Communications) protocol. To meet this requirement we need to integrate an OPC module in Alices. The implementation of the OPC protocol was subject to some constraints, such as its programming language, C++, and the need for real-time performance and portability under Windows and Linux. To implement the OPC protocol, I began by creating some prototypes in order to choose between OPC DA and OPC UA. Our choice fell on OPC UA. Then, I started the implementation process : creating a basic Alices module, establishing a connection to the OPC server, declaring the variables for exchange, crafting an *OPCnode* class, handling errors, and documenting the process. I also outlined the remaining tasks, as well as future enhancements, and explained the challenges I encountered.
Keywords : OPC, Alices, Simulator, Nuclear, Communication protocol, C++.