Haute Ecole Arc - Ingénierie

Rapport
Connected Factory

Rapport Travail d'Automne 2016 – Projet 15DLM-TA655

Superviseur : Huber Droz

Nicolas Gonin
24.01.2017

Abstract

Ce rapport présente le déroulement de l'exploration du groupement de spécification OPC UA. La création d'un serveur OPC UA virtuel a été réalisée. La communication avec le serveur est faite par un client personnalisé et assure toutes les fonctionnalités essentielles OPC UA. Cependant, l'interfaçage avec une machine réelle n'a pas été possible car l'école ne possède pas de contrôleur compatible. Il a donc été décidé de faire un guide développeur pour faciliter un interfaçage futur.

This report presents the progress of the exploration of the UC OPC specifications. A virtual UC OPC server has been made. Communication with the server is done by a custom client and ensures all the essential functionalities of OPC UA. However, interfacing with a real machine was made impossible because the school does not have a compatible controller. It was therefore decided to make a developer guide to facilitate future interfacing.

1. Table des matières

2. Introduction

Opc Ua est l'acronyme de « Open Platform Communication Unified Architecture. ». C'est un protocole de communication de machine à machine développé par la fondation OPC. C'est protocole standard d'intégration de systèmes et de communications pour l'automatisation, l'industrie 4.0 et l'internet des objets « Internet of Things ».

L'objectif de ce projet est d'implémenter et tester un prototype de connexion de machines en utilisant la technologie OPC Unified Architecture. Le travail s'inscrit dans une vision stratégique de la Haute Ecole Arc du « Smart and Micro Manufacturing » visant à soutenir les industriels de l'arc jurassien à faire face à la concurrence mondiale dans ce domaine.

3. Rappel du cahier des charges du projet

3.1. Automatisation complète d'un prototype

On automatise la fabrication additive et réalise une ligne de fabrication interconnectée dans laquelle les informations circulent en continu pour la personnalisation de pièces de série avec des technologies Industrie 4.0. Un robot à sept axes établit l'interconnexion entre la presse à injecter et la machine de fabrication additive, un autre robot (ou le même pour les besoins du prototype) reprend la pièce à la sortie de la machine de fabrication additive pour l'amener à une machine d'usinage 5-axes en vue d'une opération de reprise. Les pièces sont ensuite contrôlées par vision dans un labo de métrologie (manutention par l'homme dans un premier temps), la machine de mesure fournit cependant les résultats sur une page internet spécifique.

Une fois que la presse a injecté une poignée en plastique sur des ciseaux de bureau et qu'un code Datamatrix a été inscrit, le robot à sept axes retire la pièce et son support de la bande transporteuse de la cellule d'injection. La paire de ciseaux est identifiée par scanner grâce à son code et l'étape de production suivante commence. Le robot charge/décharge la chambre de fabrication. La machine de fabrication additive ajoute une forme 3D en plastique personnalisée sur la poignée de ciseaux.

Le robot charge/décharge ensuite la pièce ainsi personnalisée pour l'amener sur une machine d'usinage 5-axes en vue d'une opération de reprise.

Une pièce unique de taille de lot 1 est ainsi produite.

Avant la remise des pièces par le robot, elles sont soumises à un contrôle qualité par vision.

3.2. Contraintes

Les machines concernées par le projet sont les suivantes :

1. Centre d'usinage 5axes Mikron XSM400U
    - Commande Heidenhain HR410M
2. Presse d'injection Arburg Allrounder 170S 180-70
    - Commande Selogica
3. Robot 6 axes Staübli (RX90 ou TX60L)

- Contrôleur Staübli CS8

3.3. Méthodologie

Dans un premier temps j'ai étudié le fonctionnement du protocole OPC UA.

Dans un deuxième temps, j'ai cherché une implémentation fonctionnelle et pratique d'utilisation.

Troisièmement, j'ai appris à m'en servir afin de faire fonctionner un système client et serveur capable de communiquer au travers du protocole OPC UA.

Finalement, j'ai étudié la spécificité des machines à interfacer.

4. Analyse

Dans cette partie, je vais détailler la technologie Opc-Ua et les diverses possibilités étudiées sur les implémentations existantes à utiliser pour le projet seront détaillées. Le choix définitif sera justifié.

4.1. Spécification OPC UA

La spécification OPC est définie le protocole de communication entre un client et un server. Le client envoie une requête au serveur et celui-ci lui répond puis se met en attente. Cependant, le client peut demander au server de l'informer lors d'un changement sur une donnée spécifique. Dans ce cas, le serveur enverra des données au client de manière autonome.

La fonctionnalité principale est la possibilité d'accéder et de modifier une donnée. Pour ce faire, chaque donnée possède un nombre d'attribut minium définit. Elle doit posséder un nom et une valeur. En plus de ça certaines informations supplémentaires viennent s'ajouter. L'instant à laquelle la donnée a été lue fait aussi partie des attributs minimum. Cette information peut correspondre au moment où l'information a été envoyée par le nœud où la machine est connectée ou le moment où le serveur l'a reçue. Finalement, le dernier attribut est la qualité de la donnée.

En résumé: Data = Name, Value, Timestamp and Quality.

Le deuxième protocole permet de monitorer une ressource sur le serveur. Le client peut s'inscrire pour recevoir la nouvelle valeur d'une donnée spécifique si celle-ci venait à être modifiée. Le serveur ne conserve aucune information de l'échange et envoie juste la nouvelle valeur ainsi que le moment où elle a été modifiée mais il n'envoie pas les attributs du nom ou de la qualité.

Le troisième protocole ajoute la fonctionnalité d'historique d'un élément. Le client peut faire une requête sur plusieurs valeurs antérieures d'une donnée.

Opc Ua rend ces protocoles agnostiques aux systèmes d'exploitation et les encapsules dans une nouvelle structure de donnée. Il est possible de faire une requête concernant plusieurs données et toutes vont être lu simultanément et envoyé au client sous la forme d'un seul paquet.

La communication entre le client et le serveur est indépendante des protocoles de communication par réseau bas niveau. Le transport d'information se fait au travers du protocole TCP/IP puis une surcouche en SSL, http ou https est ajoutée. La couche de communication ne sécurise pas seulement

les données, il sécurise aussi l'authentification ce qui protège le serveur de l'infiltration ou de la modification par un tiers non autorisé. Cette partie est basée sur la norme de certificat X.509[i] qui comprend trois parties dont la première doit être réalisée manuellement de pair-à-pair puis le reste est automatique.

4.2. Fonctionnement de OPC UA

4.2.1. NodeId

Chaque entité dans l'espace d'adressage est un nœud. Pour identifier un nœud de manière unique, chaque nœud possède un NodeId, qui est toujours composé de trois éléments :

4.2.1.1. NamespaceIndex

Représente l'index utilisé par un serveur UA OPC pour référencer l'URI (Uniform Ressource Identifier) de l'espace de noms donné. L'URI de l'espace de noms identifie la convention de nommage définissant les identifiants de NodeIds. Ils sont stockés dans le tableau nommé namespace ou namespace table). Les index d'espace de noms sont des valeurs numériques utilisées pour identifier des espaces de noms afin d'optimiser le transfert et le traitement.

4.2.1.2. IdentifierType

Représente le format et le type de données de l'identifiant. Il peut s'agir d'une valeur numérique, d'une chaîne, d'un l'identifiant unique (GUID) ou d'une valeur opaque (un format d'espace de nommage spécifique dans une chaîne de Byte). Le format le plus utilisé est le format numérique car c'est le moins gourmand en ressource et le plus rapides à résoudre par le serveur.

4.2.1.3. Identificateur

Représente l'identificateur d'un nœud dans l'espace d'adressage d'un serveur OPC UA. Exemple : NumericalNode (ns=2, i=3) référence le nœud dont l'identifiant numérique est « 3 » au sein de l'espace de nom dont l'index est « 2 ».

4.2.2. Node

Il existe plusieurs types de nœuds. Nous allons nous intéresser au nœud capable de stocker des informations et ferons ici abstraction des nœuds représentant des types de donnée.

4.2.2.1. Object

L'objectif principal de cette classe est de fournir au serveur un moyen standard de représenter des objets à aux clients.

4.2.2.2. Variable

Une variable est un nœud qui possède les mêmes possibilités qu'un objet mais ajoute la possibilité de stocker des données.

4.2.2.3. Property

Une propriété possède les mêmes attributs que le nœud variable. Cependant ses types des données ne sont pas obligatoirement des types issus de la spécification OPC UA. Le serveur peut définir ses propres types de données.

4.2.2.4. Methode

Une méthode permet d'appeler une fonction du serveur. Il est possible de configurer une méthode avec des arguments de n'importe quels types.

4.2.3. OPC UA Concept d'abonnement

Un client peut s'abonner à différents types d'informations fournies par un serveur OPC UA. Le but d'une souscription est de regrouper ces sources d'information, appelées « Monitored Items », en un seul message appelé « Notification ».

4.3. Comparaison des implémentations existantes du protocole OPC-UA

Il existe déjà un bon nombre d'implémentation du protocole Opc-Ua[ii] Dans un souci de cohérence et de simplicité, je me suis uniquement penché sur les projets qui implémentent le stack, le client et le serveur. Je les ai testés dans un ordre de préférence de langage de programmation. Tous les projets sont capables de faire de la lecture et de l'écriture de donnée au travers du protocole Opc-Ua.

4.3.1. digitalpetri/ua-client-sdk et digitalpetri/ua-server-sdk[iii]

Ce projet est écrit en Java. La création du client ou du server a échouée lors de l'utilisation de l'exemple fourni.

Aucune classe n'était exécutable et mes efforts pour créer une classe permettant l'utilisation de classe provenant de ce projet n'ont pas été fructueux.

4.3.2. eclipse/milo[iv]

Ce projet est intègre le projet précédant en utilisant Maven.

La classe d'exemple renvoie une erreur lors de son exécution. Celle-ci est lancée par le processus d'instanciation d'un canal sécurisé. La phase d'authentification n'accepte pas l'utilisation du certificat par défaut. Le fichier contenant le certificat de sécurité est placé dans le dossier des certificats non-reconnu par l'application. La résolution de ce problème proposé par les développeurs eux-mêmes consiste à déplacer ce fichier à la main dans le dossier des certificats reconnus. Cette solution n'a donné aucun résultat.

Il semblerait qu'il ne soit pas possible d'instancier un protocole sans certificat de sécurité avec ce projet.

4.3.3. ctron/de.dentrassi.camel.milo[v]

Ce projet intègre le projet précédant en utilisant Camel. Apache Camel est un framework Java libre dont le but est de rendre l'intégration d'applications plus facile et plus accessible pour les développeurs.

Ce projet requière l'installation de plusieurs dépendances et nécessite donc l'apprentissage de ces outils englobants. Le résultat est une interface haut niveau qui ne permet pas un contrôle fin sur l'utilisation de la technologie Opc-Ua de manière directe. Ce projet occulte trop d'informations en proposant des interfaces haut-niveau qui s'adapte difficilement à des cas spécifiques. L'utilisation de beaucoup de dépendances rend la détermination de la source d'une erreur plus difficile. Pour toutes ces raisons, je me suis tourné vers un projet plus bas niveau.

4.3.4. FreeOpcUa/python-opcua[vi]

FreeOpcUa est un projet open-source développé en python 3.

Les exemples de base sont fonctionnels tant au niveau du client qu'au niveau du serveur. L'importation de XML coté serveur et coté Client ne semble pas fonctionnelle.

Le client graphique est fonctionnel et permet le test de certaines fonctions de base de la spécification Opc Ua. Son utilisation est utile à des fins de débogage.

La documentation est lacunaire. Seules les fonctions haut-niveau principales sont bien documentées mais n'est jamais exhaustive comme l'est la spécification de base OPC UA. Le code source n'est d'aucune aide car est dépourvu de tout commentaire supplémentaire.

Ce projet n'est pas encore finalisé donc certaines fonctionnalités optionnelles OPC UA ne sont pas implémentées ou ne sont pas fonctionnelles. En plus, la nomenclature de certaines classes, méthodes ou attributs ne respecte pas la spécification OPC UA.

Une autre complication provient du langage. Python est un langage dynamique moins adapté à l'analyse de code statique que les langages statiques comme C ou Java. Bien que les outils d'analyse statique de Python puissent extraire certaines informations du code source Python sans les exécuter, ces informations sont souvent très superficielles et incomplètes.

L'auto-complétion ne fonctionne pas toujours dans l'éditeur. Certaines options ne seront pas accessibles. Par contre, l'interpréteur a accès à tout ce qui se passe en temps réel, donc il peut utiliser les outils de réflexion de type de python pour récupérer toute les informations sur un objets.

4.4. Interfaçage sur les machines

Le but est d'accéder aux variables et fonctions d'une des machines listées au travers du protocole Opc Ua. Ce protocole doit être implémenté par le fabriquant de la commande de la machine.

Après des recherches approfondies, j'ai conclu qu'aucune des commandes spécifiées dans ce projet n'implémentaient ce protocole. Pire, aucune commande actuellement en possession par la He-Arc l'implémente. Un employé de la marque Stäubli a même affirmé que ce protocole n'allait pas être intégré à sa prochaine génération de commande telle que le contrôleur robot CS9[vii].

Nous avons donc rencontré un problème insoluble à ce niveau.

4.5. Implémentation d'une démonstration

Sans interfaçage physique possible, il n'est pas possible d'implémenter un client capable de réaliser une automatisation complète de la production d'un lot. L'objectif assigné par le cahier des charges est une impasse et a dû être contourné. La motivation première de ce projet est d'évaluer le potentiel et la faisabilité par rapport au prototype à implémenter du protocole OPC UA. A cette fin il a été convenu de développer une démonstration qui implémente les fonctionnalités clés qui auraient été nécessaire à l'implémentation du prototype.

4.5.1. Client

J'ai décidé de faire un client console qui teste automatiquement le fonctionnement des fonctionnalités.

Le client doit être capable de :

- Modifier et accéder aux variables du serveur ;
- Souscrire à la modification d'une variable ;
- Récupérer l'historique des modifications d'une variable ;
- Appeler une fonction avec des arguments personnalisés ;
- Souscrire aux événements du serveur.

Le fichier python client_demo.py implémente toutes ces fonctionnalités.

4.5.2. Serveur

Lors de l'acquisition d'un contrôleur supportant le protocole OPC UA, le serveur est fourni et configuré par le fabriquant. L'espace d'adressage entièrement peuplé et sécurisé afin d'empêcher l'envoie d'ordre invalide au robot. Il n'est donc pas nécessaire de manipuler le serveur pour utiliser le protocole. Il est néanmoins pratique de pouvoir ajouter des variables et des événements personnalisés aux sein du serveur. En l'absence d'un serveur propriétaire, j'ai créé mon propre serveur simulant la commande d'un robot munit d'un bras capable de se déplacer sur deux axes.

Le serveur doit être capable de :

- Instancier et modifier un objet ;
- Instancier et modifier une variable ;
- Empêcher la modification de certaine variable ;
- Conserver et l'historique des modifications d'une variable.

- Envoyer une notification lors de la modification d'une variable ;
- Envoyer une notification lorsqu'une condition est atteinte ;

Le fichier python server_demo.py implémente toutes ces fonctionnalités.

4.5.3. Démonstration de la communication client-serveur

Afin de réaliser cette démonstration il faut premièrement exécuter le fichier python server_demo.py. Le serveur a besoin de deux secondes pour se configurer et être prêt à écouter le port 4840 sur l'adresse local 127.0.0.1. Deuxièmement, il faut exécuter le fichier python client-demo.py. Le client va se connecter au serveur et effectuer automatiquement une série de tests sur les fonctionnalités mentionnées ci-dessus. Le client s'arrête automatiquement à la fin du test. Le serveur doit être arrêté manuellement.

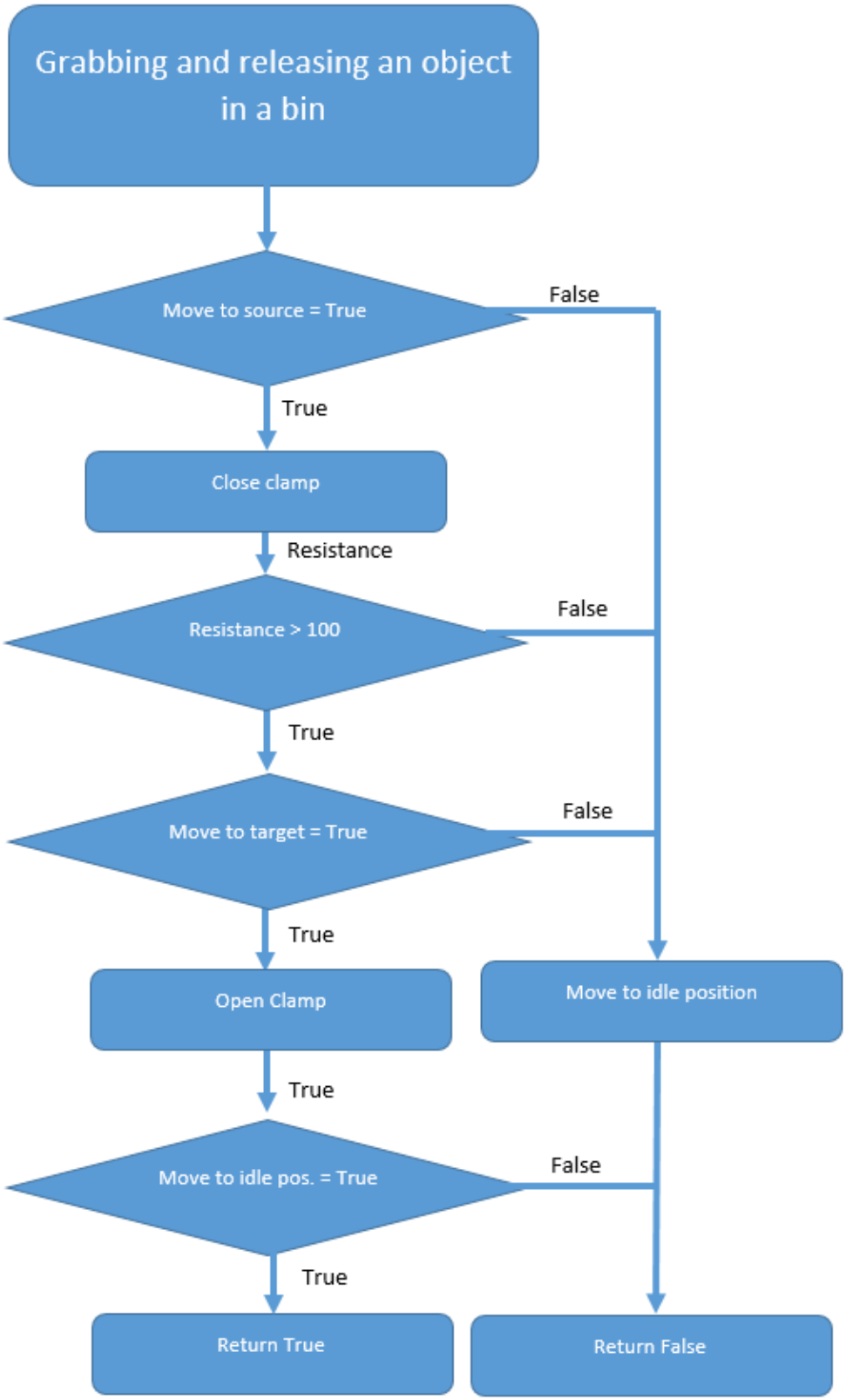
Afin de visualiser le transfert de données en local, j'ai utilisé le logiciel RawCap[viii] qui génère un fichier «. pcap » qui est lisible avec WireShark[ix].

4.5.4. Simulation d'un cas d'application réel

J'ai inventé un robot capable de déplacer des objets. Il est doté d'un bras articulé capable de se déplacer horizontalement et de saisir un objet à l'aide d'une pince deux doigts. La position du bras correspond à la position de sa pince. Ses déplacements sont confinés à l'intérieur d'une zone carrée de 20cm de côté. La pince est pourvue d'un capteur de pression. Le but de ce robot est de saisir un objet et de le déposer dans un conteneur.

La fonction bool grab_object(source_coord, target_coord) est implémentée du coté server et du coté client. Elle demande au robot de saisir un objet aux coordonnées sources et de le déposer aux coordonnées cibles. Elle retourne vrai si l'opération s'est bien effectuée et faux sinon. Des messages consoles permettent le suivit de chaque opération.

Le diagramme logique de cette fonction est illustrée ci-dessous.


Grabbing and releasing an object in a bin
Move to source = True
False
True
Close clamp
Resistance
Resistance > 100
False
True
Move to target = True
False
True
Open Clamp
Move to idle position
True
Move to idle pos. = True
False
True
Return True
Return False

5. Conclusion

Il n'a pas été possible d'implémenter le prototype définit dans le cahier des charges par manque de matériel adéquat. Cependant, les recherches effectuées dans le cadre de ce projet montrent que le protocole OPC UA est adapté à cette implémentation. OPC UA est très performant et agréable d'utilisation. Ce projet servira de base théorique à une future implémentation réelle.

6. Annexes

6.1. Dépôt du code

→ https://github.com/thegazou/connected-factory

6.2. Journal de travail

→ https://github.com/thegazou/connected-factory/wiki/Journal-de-travail

7. Références

[i] Page Wikipédia de la norme X.509
[ii] https://github.com/digitalpetri/opc-ua-stack
[iii] Liste d'implémentation OPC UA libres
[iv] https://github.com/digitalpetri/ua-client-sdk
[v] https://dentrassi.de/2016/07/18/bringing-opc-ua-to-apache-camel/
[vi] https://github.com/FreeOpcUa/python-opcua
[vii] Page de présentation du Contrôleur Stäubli CS9
[viii] Site de RawCap
[ix] Site de WireShark