Ada for Automation

SUIVI DU DOCUMENT			
INDICE	DATE	MODIFICATIONS	NOM
2015.02	2015-02-15		SLO

Table des matières

A ma famille et à mes amis.
Aux petits chevaux blancs de Monsieur Brassens.

Chapitre 1

Introduction

"Ada for Automation" (A4A en version courte) est un cadre applicatif, ou framework, pour la conception d'applications d'automatisme industriel dans le langage Ada. Il s'appuie sur la bibliothèque libmodbus pour permettre de réaliser un client ou un serveur Modbus TCP, ou encore un maitre Modbus RTU. Il s'appuie également sur les cartes de communication de la société Hilscher permettant de s'interfacer sur les principaux bus de terrain du marché comme AS-Interface, CANopen, CC-Link, DeviceNet, PROFIBUS, EtherCAT, Ethernet/IP, Modbus TCP, PROFINET, Sercos III, POWERLINK, ou VARAN.

Il est nécessaire de préciser un peu le contexte, ce qui va être fait succinctement ci-après.

1.1 Automatisme et automates

Historiquement, les systèmes automatisés sont gérés depuis les années 1980 et l'avènement des microprocesseurs par des ordinateurs spécialisés que l'on désigne sous le terme d'Automate Programmable Industriels -API- ou, en Anglais, Programmable Logic Controller -PLC.

Le système à contrôler, une machine, un réacteur chimique, biologique ou nucléaire, un engin quelconque, dispose de capteurs, qui permettent de connaître à chaque instant l'état du système, et d'actionneurs qui permettent d'agir sur celui-ci.

Les capteurs fournissent des informations Tout-ou-Rien -TOR- qui suivent une logique booléenne, vrai ou faux, chaud ou froid, ouvert ou fermé, etc. ou des informations sur une grandeur mesurée, température, pression, vitesse, que l'on dit analogique, et qui en fait est une grandeur physique convertie dans un signal électrique que l'on numérise pour pouvoir le traiter par programme.

A cette époque, les automates recevaient leurs informations ou pilotaient leurs actionneurs par des cartes électroniques, disposées dans un châssis qui assurait à la fois une protection mécanique et électrique, contre les perturbations électromagnétiques.

L'unité de traitement ou unité centrale ou encore CPU (Central Processing Unit) logeait également dans le même châssis ou encore rack. L'on pouvait étendre le châssis via des nappes d'extension et disposer ainsi de davantage d'entrées, pour acquérir les signaux, et sorties, pour piloter les actionneurs.

Chaque capteur remonte son information soit en commutant une tension 0 - 24 - 48 voire 110 Volts pour les entrées TOR, soit en générant un signal 0 - 10V pour une entrée en tension, 0 - 20mA ou 4 - 20mA pour une entrée courant.

Les actionneurs sont pilotés soit en TOR, via des cartes relais, transistors de puissance, triacs..., soit en analogique en envoyant un signal 0 - 10V, 0 - 20mA ou 4 - 20mA à un régulateur, un gradateur, un variateur...

Sur une installation conséquente, les châssis automate occupaient un place importante dans les armoires électriques et la quantité de câbles nécessaires était proportionnelle à la quantité de capteurs et actionneurs raccordés. Pour un capteur, un câble pour le signal et un câble d'alimentation, qui pouvait alimenter plusieurs capteurs éventuellement, étaient nécessaires. Pour les actionneurs, comme il est possible d'avoir un câble pour les signaux analogiques, disons la vitesse pour fixer les idées, un câble pour les signaux TOR, commande de marche / arrêt par exemple et bien sûr les câbles de puissance, les tireurs de câbles avaient du travail.

Tout ces câbles ont un coût, à l'achat, en études, en mise en œuvre, et en maintenance, sachant que les problèmes de connectique représentent environ 80% des pannes.

1.2 Les bus de terrain

Dans les années 1990, on a commencé à voir apparaître les bus de terrain comme Modbus ou PROFIBUS.

Un bus de terrain, c'est un câble, on dit un médium lorsque l'on est spécialiste, qui va d'équipement en équipements, et sur lequel s'opère une communication entre ces équipements selon un protocole de communication.

Comme il y a toujours de nombreuses solutions à un problème, il s'est établi un grand nombre de protocoles, une nouvelle Babel.

Sur ce bus de terrain, on peut coder les signaux selon de multiples caractéristiques pour arriver à transférer les signaux avec une bonne performance et à un coût toujours moindre, la performance s'entendant selon des critères variés comme le débit, la vitesse, la résistance aux perturbations, la distance, la sécurité… Chaque compromis trouvé a ses partisans et ses détracteurs.

Pour développer, standardiser et promouvoir chacune de ces technologies, des organisations sont mises en places.

Sur ce bus de terrain donc, de nouveaux équipements communicants font leur apparition, des modules disposant d'entrées et sorties locales et permettant de déporter l'acquisition et le contrôle des capteurs et actionneurs, mais également, des variateurs pour le contrôle de moteurs, des gradateurs, toutes sortes.

L'alimentation et le bus de terrain suffisent au pilotage des installations par des automates dont la capacité de traitement augmente en conséquence.

Toujours en quête de performance et de réduction des coûts, l'industrie a créé depuis de nouveaux protocoles basés sur une technologie abordable et performante, la technologie Ethernet. Bien sûr, il y a toujours un bémol et il a fallu beaucoup amender cette technologie pour qu'elle convienne aux applications de contrôle-commande.

Comme il y a toujours de nombreuses solutions à un problème, il s'est établi un grand nombre de nouveaux protocoles, une nouvelle Babel…

Ces capacités de communication profitent également aux échanges avec les outils de supervision et de diagnostic qui peuvent rafraîchir les données plus rapidement et de façon plus transparente. Ils permettent ainsi une meilleure disponibilité des installations et un contrôle plus poussé.

On peut également mentionner comme avantages les suivants :

— il est possible d'insérer sur un bus de terrain des équipements de différents constructeurs, ce qui est moins évident dans des racks automates. L'architecte de la solution est ainsi plus libre dans le choix des éléments.

— la modularité des stations d'E/S déportées est bien supérieure à celle des cartes pour racks. Il est possible de panacher des entrées / sorties de différents types dans une station alors qu'une carte aura une fonction (entrées ou sorties - TOR ou analogique) avec un nombre de voies fixes.

1.3 Les PC Industriels

On trouvait depuis longtemps de gros ordinateurs pour gérer des procédés complexes, hauts-fourneaux, raffineries… Avec des systèmes d'exploitation propriétaires, des ingénieurs pour leur administration et d'autres pour la maintenance évolutive, ces systèmes n'étaient pas pour tout le monde.

Les PC avaient envahi les bureaux mais étaient cantonnés à la supervision car jugés pas assez fiables pour le contrôle-commande. Il y avait bien des tentatives mais elles avaient mauvaise presse. On les réservait aux besoins spécifiques nécessitant de la puissance de traitement non disponible avec les processeurs équipant les automates ou des capacités de stockage ou d'accès aux données d'une base.

Depuis, les choses ont un peu changé, Linux est passé par là et la fiabilité des systèmes d'exploitation s'est améliorée. Les matériels sont également bien moins coûteux et leur fiabilité est bien plus importante en choisissant des configurations sans pièces tournantes.

Le MTBF (Mean Time Between Failure) des PC industriels est très proche de celui des automates, cela dit, c'est un peu normal puisqu'ils partagent aujourd'hui nombre de composants.

Et en offrant une capacité d'évolution et une ouverture sans commune mesure avec l'automate, le PC industriel dispose d'atouts indiscutables.

Dès lors que l'on avait besoin d'un PC pour la supervision ou des traitements évolués, certains se sont demandé pourquoi on ne ferait pas de contrôle-commande avec la puissance de calcul disponible en se passant de l'automate.

On y gagne en coût, en compacité, en simplicité d'architecture…

On a donc rajouté des extensions Temps Réel aux systèmes d'exploitation généraliste et développé des solutions d'automatisme sur PC.

Siemens avec WinAC® RTX® avec l'extension Temps Réel de IntervalZero®, ISaGRAF®, CoDeSys®, ProConOS®, Straton®, TwinCAT®… Une palanquée.

Il est possible d'installer des cartes d'E/S dans les PC et d'obtenir ainsi une architecture ressemblant fortement à un automate en rack. C'est même la solution la plus utilisée dans nombre d'applications exigeantes.

Il est également possible d'installer dans le PC une ou plusieurs cartes de communication industrielle, comme les cartes de la société Hilscher par exemple, afin de fournir une connexion au bus de terrain en mode maître pour piloter les équipements ou en mode esclave pour communiquer avec le niveau supérieur.

Les automaticiens sont encore un peu réticents à mélanger l'informatique et l'automatisme. Mais les temps changent, les seniors sont ringardisés par les juniors élevés au Web 2.0, les technologies poussent à la roue et l'on assiste à une convergence PC - Automate, avec une situation économique qui comprime les budgets…

1.4 Automaticien

Un automaticien, c'est un spécialiste de l'automatisation. Autant c'est facile à dire, autant c'est un métier difficile à saisir.

Selon le domaine, chimie, pétrochimie, pharmacie, industrie manufacturière, gestion technique de bâtiments, machines spéciales…, la formation initiale, instrumentiste, électricien, mécanicien, électronicien, technicien de maintenance, la structure de la société qui l'emploie, sa curiosité et ses penchants enfin, le profil de l'automaticien standard est plutôt vague et ses tâches extrêmement variées.

Il peut être amené à exercer tour à tour plusieurs professions sur un même projet. Il n'est absolument pas rare qu'un automaticien mène à bien les études concernant l'instrumentation, l'électricité, l'automatisme et la supervision, le tout agrémenté de réseaux et bases de données, etc…

Vous remarquerez que l'informatique n'est pas citée dans ses cordes. C'est en fait souvent sur le tas et un peu par obligation qu'il s'y met.

Un automaticien conçoit donc des programmes automates, entre autres, mais n'a souvent qu'une culture, et non pas une formation, en informatique.

1.5 Langages en automatisme

Aussi, on lui a concocté une panoplie de langages adaptés à ses tâches, traiter de la logique combinatoire, des machines d'état, asservir ou réguler des grandeurs, surveiller des procédés…

On trouve ainsi :

— le langage ladder ou à échelle ou à contacts, avec ses contacts en série ou parallèle, pour faire des ET et des OU, les bobines de relais pour mémoriser des états, le langage des électriciens, les automates ayant remplacé les dizaines de mètres d'armoires à relais d'antan.

— les logigrammes, une représentation conçue pour les électroniciens et leurs portes AND, NAND, NOR… peu utilisé aujourd'hui il nous semble.

— la liste d'instructions, un pseudo assembleur pour les électroniciens de l'ère numérique, avant l'avènement du C.

— le GRAFCET, une spécialité française pour la représentation des machines d'état, adapté aux mécaniciens, quand la mécanique va lentement…

— les schémas blocs, où le système est représenté par des blocs fonctions reliés par des conducteurs, adaptés pour la régulation, l'asservissement…

— le texte structuré, un ersatz de Pascal, Ada, C… sensé limiter la "créativité" et les bogues qui vont avec.

Tous ces langages ont des capacités de débogage intégrées ou visualisation dynamique, ce qui est très pratique, convenons-en.

Il est de plus en général possible de procéder à des modifications en ligne du programme, c'est à dire durant le fonctionnement de la machine sans devoir la réinitialiser, ce qui est fort utile lors des démarrages de l'installation. Bien évidemment, il y a quelques bémols et certaines opérations nécessitent une recompilation du programme complet, comme les modifications de la configuration, celles qui ont trait à l'interface des fonctions et des blocs, etc…

Cependant, ces langages sont adaptés à des tâches d'ampleur limitées et leur expressivité est assez mauvaise.

Dès qu'une équation booléenne est un peu complexe, les réseaux du ladder deviennent difficiles à comprendre et il est encore plus difficile de les faire évoluer sans introduire de régression. Les programmes font des kilomètres.

Le logigramme a disparu, nous n'avons pas entendu de plainte.

Le LIST, le pseudo assembleur, est aussi austère que le vrai assembleur. Difficile à écrire, encore plus difficile d'y relire. Le langage préféré de mon collègue indépendant qui disait : "il faut mystifier le client !". Il y parvenait le bougre…

Le GRAFCET… Surtout en France donc.

Avec tous ces langages, si tant est que l'on soit formé à leur pratique et à leurs spécificités, on arrive à faire des choses, surtout si l'on choisit le langage le plus approprié.

Faut-il encore qu'ils soient disponibles. Ce n'est pas le cas en général, les commerciaux nous gratifient de leurs offres à tiroir, et les gestionnaires ont souvent une vision court-termiste. Ainsi, tel atelier ne proposera de base que le contact, le logigramme -inutile, et le list.

En gestionnaire avisé, l'on ne fera pas par exemple l'investissement dans des langages plus évolués comme le langage structuré ou les schémas blocs car ils offrent une productivité supérieure dans biens des cas. Ainsi, les automaticiens, que l'on ne formera pas et qui ne pourront le faire eux-mêmes car ne disposant pas des outils, continuerons de développer en langage à contact ou assembleur.

Avec le temps, ces mêmes automaticiens finissent par œuvrer chez des clients -le rêve de tout développeur en SSII- et imposent dans leur cahier des charges le langage à contact ou l'assembleur. Sans fonctions ou blocs fonctions parce que c'est trop compliqué… C'est du vécu vous dit-on.

Dans tel autre atelier, on ne disposera pas de la possibilité d'écrire une simple fonction… Exaspérant. On nous parle de programmation orientée objet et l'on n'est pas capable de faire ne serait-ce que de la programmation structurée !

Il existe également des systèmes dans lesquels des langages plus ou moins évolués sont utilisés comme C/C++, Basic, Delphi ou autres. C'est assez confidentiel mais ça existe.

Le Visual Basic for Application que l'on peut trouver dans les applications Microsoft Office® ou de supervision comme PcVue® de Arc Informatique est également pratiqué.

Chapitre 2

Ada for Automation

Il y a donc déjà une multitude de solutions propriétaires pour faire de l'automatisme sur une base PC. Ce n'est pas pour autant que cela bouscule le marché de l'automate. Les habitudes ont la vie dure et les industriels sont des gens prudents, des fois conservateurs...

Le logiciel a fait sa révolution avec le libre. Au delà de toute autre considération, l'apport que nous trouvons essentiel est d'ordre quasi philosophique : remplacer l'idéologie concurrentielle par une idéologie contributive. En s'aidant mutuellement, cela va mieux qu'en s'entre-tuant. On pourra nous soutenir ce que l'on veut, il n'y a pas d'argument qui tienne face à ce constat.

2.1 Qu'est-ce donc ?

"Ada for Automation", c'est donc un framework, c'est à dire du code en source ouverte pour vous permettre d'écrire plus facilement et rapidement vos propres applications. Avec un peu de réussite, ce framework devrait s'étoffer avec le temps et les contributions.

Ce framework ou cadre applicatif fournit un environnement où l'on trouve un certain nombre d'éléments comme des boucles d'exécution, des moyens de communication vers les entrées et sorties mais également vers les outils de supervision, des bibliothèques de fonctions et de composants...

Il est nécessaire d'y ajouter le code de votre application pour qu'il en sorte quelque chose d'utile.

Il vous faudra apprendre Ada, doucement mais sûrement, pour l'utiliser. Cependant la disponibilité du code source vous y aidera et si vous connaissez le langage ST pour "Structured Text" vous serez surpris par une certaine familiarité.

Ada est un langage compilé et il faudra apprendre à utiliser les outils et les techniques de debug. Ada est un langage particulièrement puissant et élégant, tout en restant relativement facile à apprendre pour un technicien ou un ingénieur en automatisme.

En fait, celui-ci retrouvera dans ce langage les concepts qu'il manipule au quotidien en utilisant son atelier de programmation d'automate favori -ou imposé-, avec une syntaxe claire rappelant le langage "Structured Text" de la norme IEC 61131-3, ce qui est somme toute normal puisque ce langage s'inspire notamment de Ada.

2.2 Objectifs

Quelques objectifs ont été identifiés pour ce projet.

— Disposer d'un langage adéquat pour des applications d'automatisme exigeantes, mais qui convienne aussi pour des applications basiques, multiplateforme.

— Permettre au programmeur en automatisme de se former sur les paradigmes (sous-)utilisés dans les ateliers actuels, ce qui le rendra plus productif sur ces mêmes ateliers.

— Permettre à Ada de sortir du ghetto élitiste où ses origines et son histoire l'ont enfermé, les applications sensibles dans les domaines militaire, nucléaire, aérospatial, de la finance, médical, ferroviaire...

2.2.1 Un langage puissant

Alors qu'à l'origine les automates remplaçaient des armoires électriques dans lesquelles la logique était câblée, les automates d'aujourd'hui communiquent via des réseaux avec leurs congénères, les outils de supervision, le web, les bases de données, le MES (Manufacturing Execution System)… Le nouveau concept "à la mode", l'Industrie 4.0, devrait accélérer cette tendance générale : l'innovation par l'intégration de fonctionnalité.

La notion de composant réutilisable est devenue centrale car permettant justement de capitaliser les développements et facilitant ainsi cette réutilisation. Ada, avec les paquetages permettant et organisant la modularité et la programmation objet, fournit depuis son origine en 1983 les moyens de créer ces composants.

Les applications sont désormais distribuées, l'intelligence est répartie, et la communication fédère toutes ces entités autonomes.

Ce concept trouve sa réalisation dans des standards comme IEC 61499 implémenté dans ISaGRAF® ou chez Siemens avec la "Component Based Automation". Ada avec l'annexe "Distributed Systems Annex" n'est pas en reste.

Ada est un langage à vocation généraliste. Il est possible d'écrire dans ce langage toutes sortes d'application. "Ada for Automation" ne limite en rien cette vocation tout en facilitant son adoption par une communauté plus proche du terrain.

2.2.2 Un investissement rentable

L'étude de ce langage est fascinante. Ada est un monument érigé à la gloire du Génie Logiciel. Tous les concepts élaborés au cours d'années de développement de cette science trouvent leur expression en Ada. Cela fait de ce langage le témoin de l'évolution de cette science et la cristallisation de celle-ci.

Le langage est conçu pour créer aussi bien des applications critiques dans l'embarqué et le temps réel que pour des applications tout aussi critiques pour de gros systèmes. Modularité, encapsulation, programmation structurée, programmation orientée objet, programmation concurrente, multitâche, temps réel, traitements distribués… Et tout ça libre !

Avec cette disponibilité des outils de développement, de la documentation et du code source, l'apprenant a toutes les cartes en main pour évoluer selon ses besoins et au gré de ses envies.

En général, on devient automaticien un peu par hasard et souvent dans la douleur. On apprend sur le tas, rarement dans de bonnes conditions, avec la pression du résultat, du client et de la hiérarchie.

En fait, hormis la matière traitée, l'information, les fondamentaux sont les mêmes. On conçoit un programme d'automatisme comme un programme informatique. Et les mêmes erreurs conduisent aux mêmes résultats catastrophiques.

Ada promeut les bonnes pratiques de conception : typage fort, modularité, encapsulation, objets et composants réutilisables… En apprenant Ada et utilisant "Ada for Automation" sur certains projets, un automaticien peut sans doute en tirer quelque enseignement.

2.2.3 Des qualité méconnues mais reconnues

On trouve sur le web des tentatives pour expliquer pourquoi un langage aussi vertueux que Ada soit resté si confidentiel. Peut-être est-ce dû aux domaines dans lesquels il est traditionnellement utilisé, peu enclins à communiquer. Bien d'autres raisons sont invoquées. Peut-être que tout simplement il n'y a pas assez d'exemples d'utilisation, ce que "Ada for Automation" souhaite contribuer à faire évoluer.

Peut-être aussi que Ada impressionne. La quantité de concepts exprimés dans ce langage est tout bonnement exceptionnelle et les livres sont épais et en Anglais. D'autres langages paraissent plus abordables et plus rapidement maîtrisables. Ce peut être le cas, mais chaque médaille a son revers : qu'en est-il lorsque l'application devient trapue, qu'elle a été écrite il y a longtemps par des personnes aujourd'hui absentes et avec des outils indisponibles ?

2.3 Pour quels types de projets ?

Ada est un langage compilé, d'une performance comparable à celle du C/C++. Sur les plateformes actuelles, la performance obtenue permet d'envisager la migration d'une large majorité des applications aujourd'hui sur automate.

Est-ce à dire que "Ada for Automation" s'oppose aux fabricants d'automates ? Bien sûr que non !

D'une part, la plupart des fabricants d'automates disposent également d'une offre en "PC Based Automation". C'est notamment le cas de Siemens avec WinAC® sur les MicroBox et NanoBox par exemple, mais aussi de Beckhoff, B&R…

D'autre part, s'il advenait que "Ada for Automation" soit adopté par une frange significative des automaticiens, rien n'interdit aux fabricants d'automates d'intégrer cette possibilité dans leur panoplie. C'est ça aussi le libre !

Selon la plateforme matérielle et le système d'exploitation utilisé, ainsi que l'architecture du système, la distribution des traitements, toutes les applications sont possibles.

— Sur des processeurs d'entrée de gamme et des OS standards comme Microsoft Windows® ou Linux / Linux RT on peut envisager de la Gestion Technique Centralisée ou Gestion Technique de Bâtiment (GTC / GTB), de la Gestion d'infrastructure, le traitement de l'eau et des effluents, le convoyage et la manutention…

— Sur des plateformes Windows RTX®, VxWorks® ou Linux RT, Xenomai, RTEMS… et un matériel adapté on peut accéder aux applications haut de gamme comme des machines spéciales avec gestion d'axes.

Cela peut paraître ambitieux. Le projet "Ada for Automation" n'a pour objectif que de fournir une plateforme ouverte pour bâtir des applications d'automatisme. Ce sont les utilisateurs et contributeurs du projet, et bien sûr les clients finaux, qui fixeront eux-mêmes les limites.

Cependant c'est atteignable. Les consomm-acteurs sont des entités puissantes qui ont tout intérêt à voir s'établir des standards, des bibliothèques de composants réutilisables, de bonnes pratiques de conception.

On peut imaginer également une sorte d'intégration verticale avec des applications "métier" créées par leurs utilisateurs.

D'autres communautés ont pu se constituer dans des domaines pas si éloignés et ont obtenu de bousculer les monopoles établis.

La rénovation ou la transformation d'installations existantes sont sans doute des phases propices à l'évaluation / adoption de "Ada for Automation".

Il est possible de conserver et réutiliser une partie, sinon la totalité, des équipements et du câblage en équipant les racks automates avec des coupleurs bus de terrain en lieu et place de l'unité centrale (CPU), les cartes d'entrées / sorties étant conservées.

Cf. par exemple les solutions développées par la société EFSYS : http://www.efsys.fr/

Ainsi, seule la partie commande est à remanier, ce qui doit être également fait de toute façon quelle que soit la solution choisie. La compatibilité de l'ancien logiciel avec le nouveau matériel est plus que rare et même si c'était le cas, ou si des moulinettes permettent de s'en approcher, il est sans doute souhaitable de démêler les modifications effectuées au cours de la vie de l'installation dans des conditions suboptimales.

Tant qu'à rénover, autant le faire correctement et ne pas bâcler ce qui constitue l'intelligence du système.

2.4 En conclusion

"Ada for Automation" constitue une solution économique, puissante et pérenne pour la réalisation d'applications de contrôle-commande évoluées.

Bien sûr, nous ne prétendons pas révolutionner le domaine. L'auteur est loin d'être un expert en Ada. C'est même en parfait débutant qu'il a commencé le développement de ce cadre applicatif.

Et nous connaissons les résistances au changement pour les avoir expérimentées.

"Ada for Automation" ne s'adresse pas aux inconditionnels de tel ou tel constructeur / atelier / langage. Il est nécessaire d'avoir un peu de curiosité intellectuelle pour s'affranchir des réflexes acquis et s'aventurer hors des sentiers battus. Il faudra un peu de temps aux personnes intéressées pour maîtriser les concepts et outils utilisés. Cependant les concepts sont universels et les outils libres. Nous sommes certains que les bénéfices en termes de formation et d'ouverture peuvent être considérables.

Nous n'opposons pas les langages d'automatisme et Ada. Nous pensons qu'il y a de la place pour tous. Mais nous serions très heureux si ce langage pouvait trouver sa place dans la panoplie de l'automaticien.

Ada permettrait de résoudre la problématique de la portabilité qui n'est pas encore résolue avec PLCopen, plus sans doute par manque de volonté des fabricants que par une impossibilité technique.

La maintenabilité a toujours été l'un des critères majeurs dans l'élaboration de Ada. L'accent est mis sur la lisibilité du code car s'il est écrit au moins une fois il sera l'objet de maintes relectures.

Il est évident que le milieu de l'automatisme et de l'informatique industrielle et ceux dans lesquels Ada est traditionnellement utilisé partagent les mêmes besoins en termes de durée de vie, de pérennité et de fiabilité des solutions.

Avec une norme écrite en 1983, et mise à jour en 1995, 2005 et 2012 en intégrant les nouveautés du Génie Logiciel apparues dans l'intervalle et jugées intéressantes par les experts des domaines d'utilisation et du langage, Ada fait montre d'une belle santé. Et il est encore aujourd'hui possible avec les outils dernière génération de compiler des applications écrites à l'origine du langage, même si cela dit elles mériteraient quand même un petit coup de dépoussiérage…

Nous croyons aux vertus de l'exemplarité. Si les fruits sont bons, c'est que l'idée est bonne. C'est le sens, sinon la formule…

Nous pensons que l'on peut se limiter pour "Ada for Automation" à utiliser les concepts de base du langage, de façon à être le plus intelligible aux non-Adaïstes. Cela ne limite en rien l'utilisation de Ada pour l'application utilisateur.

Nous sommes depuis longtemps acquis aux idées du logiciel libre et espérons contribuer à cette révolution avec "Ada for Automation".

La liberté est un droit en principe… mais elle se mérite.

2.5 Licence

2.5.1 Code source

Le code source de "Ada for Automation" est couvert par la GNAT Modified General Public License (GMGPL) : http://en.wikipedia.org/wiki/GNAT_Modified_General_Public_License

C'est ainsi que l'on trouve en en-tête de chaque fichier le texte suivant :

```
--                      Ada for Automation                          --
--                                                                  --
--              Copyright (C) 2012-2014, Stephane LOS               --
--                                                                  --
-- This library is free software; you can redistribute it and/or    --
-- modify it under the terms of the GNU General Public              --
-- License as published by the Free Software Foundation; either     --
-- version 2 of the License, or (at your option) any later version. --
--                                                                  --
-- This library is distributed in the hope that it will be useful,  --
-- but WITHOUT ANY WARRANTY; without even the implied warranty of   --
-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU --
-- General Public License for more details.                         --
--                                                                  --
-- You should have received a copy of the GNU General Public        --
-- License along with this library; if not, write to the            --
-- Free Software Foundation, Inc., 59 Temple Place - Suite 330,     --
-- Boston, MA 02111-1307, USA.                                      --
--                                                                  --
-- As a special exception, if other files instantiate generics from --
-- this unit, or you link this unit with other files to produce an  --
-- executable, this  unit  does not  by itself cause  the resulting --
-- executable to be covered by the GNU General Public License. This --
-- exception does not however invalidate any other reasons why the  --
-- executable file  might be covered by the  GNU Public License.    --
```

2.5.2 Ce document

2.6 Se procurer Ada for Automation

"Ada for Automation" est hébergé chez Gitorious :
https://gitorious.org/ada-for-automation

Vous pouvez donc désormais vous procurer "Ada for Automation" avec les outils git usuels :

```
git clone git://gitorious.org/ada-for-automation/ada-for-automation.git A4A
```

ou télécharger une archive :
https://gitorious.org/ada-for-automation/ada-for-automation/archive-tarball/master

Un tutoriel Git est disponible ici par exemple :
http://www.kernel.org/pub/software/scm/git/docs/gittutorial.html

2.7 Contribuer

Il est bien sûr possible de contribuer à ce projet :

— vos remarques, critiques et suggestions sont bienvenues et seront étudiées et débattues.

— vos éléments de bibliothèque, composants, fonctions, etc. l'enrichiront.

— traduire la documentation, ce livre en particulier, accroîtra sa diffusion.

— les utilisateurs bénéficieront de votre aide sur le forum.

Un grand merci à tous nos contributeurs et notamment à :

Aurélien MARTEL pour le logo de "Ada for Automation"

François FABIEN pour ses conseils avisés (francois_fabien@hotmail.com)

2.8 Sur la toile

La page d'accueil du projet :
http://slo-ist.fr/ada4autom

Le forum dédié :
http://slo-ist.fr/forum/index.php

Hilscher France, qui emploie l'auteur :
http://www.hilscher.fr/

Chapitre 3

A propos de Ada

"Ada for Automation" c'est du 100% Ada.

Toutes les ressources Ada sont donc des ressources "Ada for Automation".

3.1 Tutoriel en Français

Vous pouvez apprendre beaucoup sur Ada en Français avec le cours de Monsieur Daniel Feneuille :
http://libre.adacore.com/tools/more-resources/ada-course-in-french

Pour ceux qui penseraient que Ada n'est accessible qu'à la fine fleur de l'Université ou des Grandes Écoles, ce cours était destiné aux étudiants de l'IUT d'Aix.

3.2 Atelier de développement

Sans outil l'homme n'est pas tout à fait l'homme.

Si vous êtes sous Linux, il y a de fortes chances pour que les outils soient disponibles dans votre distribution.

Cherchez GNAT dans votre Synaptic ou équivalent.

Monsieur Ludovic Brenta fait un travail remarquable en tant que mainteneur Ada dans Debian.

Ainsi Synaptic par exemple vous permet d'installer facilement les paquets gnat-gps, gprbuild et leurs dépendances.

Que vous soyez sous Linux ou Windows ou autre, AdaCore fournit tous les outils nécessaires et plus :
http://libre.adacore.com/

3.3 Support et formation

L'association **Ada France** avec plein de liens :
http://www.ada-france.org/

Forums Ada en Français :
https://groups.google.com/forum/?fromgroups#!forum/fr.comp.lang.ada/
http://www.developpez.net/forums/f227/autres-langages/autres-langages/ada/

Forums Ada en Anglais :
https://groups.google.com/forum/?fromgroups#!forum/comp.lang.ada/

AdaCore
La société AdaCore propose le support de Ada sur de nombreuses plateformes matérielles et logicielles.

Si vous envisagez d'utiliser "Ada for Automation" avec les cartes Hilscher cifX pour créer des solutions temps-réel, sachez que les pilotes pour les cartes cifX sont disponibles pour les plateformes Windows® avec RTX® ou InTime®, ainsi que pour VxWorks®, et bien sûr Linux.

Votre application pourra donc s'exécuter dans l'environnement de votre choix.

AdaCore University
AdaCore University est un site de e-learning consacré à Ada et aux techniques associées :
http://university.adacore.com/

Adalog
La société Adalog offre également ses services et propose des formations Ada.

3.4 Quelques références

Le Wikibook "Méthodes de génie logiciel avec Ada" de Monsieur Jean-Pierre Rosen (Adalog) :
http://fr.wikibooks.org/wiki/M%C3%A9thodes_de_g%C3%A9nie_logiciel_avec_Ada

Le Wikibook "Ada Programming" :
http://en.wikibooks.org/wiki/Ada_Programming

Le Wikibook "Ada Style Guide", une mine d'informations sur ce qu'il faut ou ne faut pas faire avec moult explications édifiantes :
http://en.wikibooks.org/wiki/Ada_Style_Guide

Le manuel de référence pour Ada 2005 :
http://www.adaic.org/resources/add_content/standards/05rm/html/RM-TTL.html

Le "Rationale for Ada 2005" est aussi une bonne source d'information :
http://www.adaic.org/resources/add_content/standards/05rat/html/Rat-TTL.html

Chapitre 4

Concepts

Dans ce chapitre efforçons nous d'établir un parallèle entre les concepts trouvés en automatisme et leur possible implémentation avec "Ada for Automation".

4.1 Sécurité

Votre application n'est pas une application dite de sécurité, pas plus ni moins que si vous utilisiez un automate traditionnel.

Sauf à utiliser un automate de sécurité, qui est donc conçu pour assurer cette fonction, la sécurité est assurée par des éléments annexes comme des capteurs de sécurité et des mécanismes, butées mécaniques par exemple, ou des équipements comme un bac de rétention, une enceinte sécurisée, etc. . .

Votre application peut surveiller votre installation et signaler des conditions anormales par des alarmes, ou couper un contacteur général, ces actions doivent être doublées selon des règles normalisées.

4.2 Temps réel

La notion de temps réel est très importante en automatisme comme en bien d'autres domaines.

Il y a plusieurs définitions de ce qu'est le temps réel et nous ne nous risquerons pas à en donner une autre.

Grosso modo, si les valeurs de commande que vous avez élaborées dans votre programme arrivent en dehors d'un laps de temps imparti, elles sont moins utiles. . . votre chariot est dans le mur ou la soupe a débordé.

Est-ce que pour autant on a toujours besoin d'un système d'exploitation temps réel ? Cela dépend des applications. On distinguera les applications qui nécessitent un temps réel approximatif ou mou et celle qui requièrent un temps réel dur.

Nombre d'applications peuvent se contenter d'un temps réel mou.

Si l'on pilote des vannes, des pompes, des régulateurs ou des variateurs autonomes, des équipements ayant des constantes de temps élevées, le temps réel mou qu'assurent les systèmes d'exploitation généralistes d'aujourd'hui peut amplement suffire.

Bien sûr, pour du contrôle d'axe où temps de réaction rapide et synchronisme sont requis, un système d'exploitation temps réel dur est nécessaire.

Comme toujours, le travail de l'ingénieur consiste à analyser correctement la problématique et à concevoir une architecture de contrôle-commande adaptée à celle-ci.

Les contraintes techniques sont rarement prépondérantes devant celles stratégiques ou économiques.

4.3 Structuration

4.3.1 Les tâches

Le concept de tâche est commun en automatisme et on retrouve cette notion fondamentale dans le langage Ada.

Ainsi dans tel automate Schneider Electric on disposera d'une tâche principale, cyclique ou périodique, et d'une tâche rapide périodique.

Dans tel automate Siemens, on aura une tâche cyclique, des tâches périodiques, des tâches sur alarme horaire, etc…

"Ada for Automation" implémente le même type de tâches le plus simplement du monde puisque ce concept est intégré dans le langage Ada.

Aussi, on trouve définies une tâche principale cyclique ou périodique et une tâche périodique, ce qui suffit dans beaucoup de cas.

Il est évidemment possible d'ajouter des tâches supplémentaires si besoin puisque vous disposez du code source.

4.3.2 Sections et blocs d'organisation

Chez Schneider Electric, les tâches exécutent le code contenu dans des sections que l'on peut voir comme des fonctions sans paramètre ni valeur de retour, dont l'exécution peut être conditionnée.

Chez Siemens, les tâches exécutent le code contenu dans les blocs d'organisation, les OB, que l'on peut également considérer comme des fonctions sans paramètre ni valeur de retour.

En Ada, l'équivalent de ces sections ou blocs d'organisation est la procédure.

4.3.3 Sous-routines

On peut également considérer les sous-routines comme des fonctions sans paramètre ni valeur de retour, donc des procédures.

4.3.4 Fonctions

Ce concept est central en programmation dite structurée.

Avec ou sans valeur de retour, avec ou sans paramètres, on les retrouve dans tous les langages d'automatisme.

En Ada, les fonctions sans valeur de retour sont des procédures, avec valeur de retour ce sont des fonctions.

Les fonctions peuvent travailler uniquement sur leurs paramètres formels, dans ce cas, pour un même jeu de paramètres elles fournissent une même valeur de retour, ou travailler avec des données globales.

4.3.5 Blocs fonctions

Ce concept dérive de la notion de fonction en y adjoignant des données d'instance.

Ces données d'instance permettent de conserver un état d'un appel à l'autre du bloc fonction.

En Ada, on dispose de la notion plus générale d'objet.

Un bloc fonction n'est qu'un objet possédant une seule méthode.

4.3.6 Bibliothèques

En automatisme, une bibliothèque regroupe un certain nombre d'éléments, fonctions, blocs fonctions, types de données…

Ada propose le paquetage, avec en prime la notion de hiérarchie.

Chapitre 5

Pour démarrer

5.1 Inscrivez vous sur le forum !

C'est toujours sympathique de trouver des congénères avec qui échanger sur les sujets qui nous occupent et nous ne saurions trop vous conseiller de vous inscrire sur le forum dédié :
http://slo-ist.fr/forum/index.php

Ainsi les réponses aux questions posées bénéficient à tous.

5.2 Acclimatation

Avant de vous lancer dans cette prodigieuse aventure qu'est le développement de votre application en langage évolué, vous pourriez avec profit suivre les cours de la AdaCore University. Vous y apprendrez les bases du langage Ada, l'utilisation des outils et accessoirement travaillerez la langue de Shakespeare.

Le tutoriel de Monsieur Daniel Feneuille vous permettra également de vous familiariser avec Ada, cf. Tutoriel en Français.

Équipés et aguerris, il vous faudra ensuite vous procurer "Ada for Automation" et libmodbus.

Pour ce qui est de libmodbus, sous Linux c'est normalement déjà disponible dans vos paquets et, si ce n'est pas le cas, les outils le sont ainsi que le mode d'emploi.

L'article suivant devrait vous permettre de l'obtenir si vous êtes sous Microsoft Windows® :
A4A : compilation de libmodbus avec GNAT GPS sous Windows

5.3 Lancez vous !

Enfin, faites votre choix parmi les applications exemples fournies en identifiant celle qui se rapproche le plus de l'application que vous souhaitez développer.

"Ada for Automation" est un framework pour concevoir des applications d'automatisme en Ada. Ce framework a pour vocation de vous permettre de créer vos propres applications simplement.

Il fournit donc quelques applications exemples pour, d'une part servir éventuellement de point de départ à la votre et, d'autre part illustrer l'utilisation de ce cadre applicatif.

Il ne s'agit pas de couvrir l'ensemble de la fonctionnalité mais de fournir des applications représentatives de celle-ci tout en restant simple d'accès.

Ainsi, l'application App1 met en œuvre uniquement le protocole Modbus TCP pour la communication tant avec les entrées et sorties du système qu'avec l'outil de supervision utilisé pour la démonstration, le produit PcVue® de Arc Informatique.

Cet exemple d'application 1 est un bon choix pour débuter. Vous n'aurez besoin que de votre machine pour apprendre et tester.

Pourquoi PcVue® ? Parce que dans la vie il faut faire des choix… Libre à vous d'en faire un autre. C'est un outil qui nous permet de réaliser simplement et rapidement une interface graphique pour nos applications, qui est connu d'une bonne partie des automaticiens, et dont vous pouvez vous procurer une version de démonstration auprès de Arc Informatique.

L'auteur utilise d'ailleurs cette version de démonstration qui a un inconvénient principal, être limitée à 25 variables d'entrée / sorties, ce qui en pratique est atteint rapidement. Aussi, il est nécessaire de jongler avec cette limite en créant par exemple plusieurs applications graphiques…

Nous sommes certains que d'autres outils peuvent également convenir. Ils sont légion sous Microsoft Windows® et il en existe quelques uns sous Linux.

MBLogic est très intéressant techniquement, il fonctionne bien mais il semble au point mort :
http://mblogic.sourceforge.net/index.html

Il est écrit en Python / JavaScript. Il doit être faisable d'adapter le principe sur Ada Web Server.

La première version de votre application est à un clic de là.

Comme il est possible de créer votre application sous Linux comme sous Microsoft Windows®, commencez donc sur la plate-forme que vous possédez et maîtrisez. Ne cumulons pas les difficultés.

Cependant, nous préférons bien sûr les technologies libres.

Chapitre 6

Visite guidée

6.1 Ce qui est disponible à ce jour

"Ada for Automation" traite votre application dans une boucle principale qui gère les entrées et sorties et appelle les fonctions utilisateur que vous allez écrire.

Une bibliothèque de composants, un peu rachitique pour l'instant certes, ne demande qu'à être étoffée. Merci pour vos contributions.

"Ada for Automation" utilise d'autres composants pour offrir une fonctionnalité élargie. Réinventer la roue, non merci !

6.1.1 libmodbus

Cette bibliothèque implémente le protocole Modbus dans ses différentes incarnations :

— Modbus RTU en mode maitre et en mode esclave,

— et Modbus TCP en mode client et en mode serveur.

La bibliothèque est disponible sous forme de code source sous licence LGPL ici :
http://libmodbus.org/

Il est possible de l'utiliser sur les plateformes : Linux, Mac OS X, FreeBSD, QNX et Win32.

Un binding Ada est disponible dans "Ada for Automation" pour ce qui a trait à la version Modbus TCP, Client et Serveur, comme pour Modbus RT Maître.

Si le besoin d'étendre ce binding pour pouvoir utiliser Modbus RTU Esclave est avéré, autrement dit, si quelqu'un en fait la demande, cela peut être réalisé rapidement.

Ce binding est mis en œuvre et "Ada for Automation" fournit ainsi :

— un Client Modbus TCP se configurant en remplissant un simple tableau, ce Client permet de scruter cycliquement un réseau de serveurs Modbus TCP.

— un Serveur Modbus TCP fournissant une interface pour une supervision ou un automate de ligne.

— un Maître Modbus RTU se configurant, à l'instar du Client Modbus TCP, via un simple tableau, ce Maître permet de scruter cycliquement un réseau d'esclaves Modbus RTU.

Pour plus d'information sur le protocole Modbus, et notamment en obtenir les spécifications, la visite du site consacré s'impose :
http://www.modbus.org

6.1.2 Hilscher

Un binding Ada de l'API Hilscher cifX est également disponible pour les fonctions nécessaires à des échanges cycliques, gestion de la mémoire image process, mais également pour les échanges acycliques et le diagnostic.

Vous en aurez besoin dans le cas où vous souhaiteriez utiliser une carte Hilscher cifX qui peut vous fournir la connectivité dont vous auriez besoin pour gérer vos équipements sur tous les principaux bus de terrain Ethernet Temps Réel ou conventionnels.

Avec la gamme cifX, disponible dans la plupart des formats, votre application peut être :

— Maitre (ou Client) sur les réseaux AS-Interface, CANopen, DeviceNet, PROFIBUS, EtherCAT, Ethernet/IP, Modbus TCP, PROFINET, Sercos III, ou VARAN,

— ou Esclave (ou Serveur) sur les réseaux CANopen, CC-Link, CompoNet, DeviceNet, PROFIBUS, EtherCAT, Ethernet/IP, Modbus TCP, POWERLINK, PROFINET, ou Sercos III.

Pour ne rien vous cacher, l'auteur travaille chez Hilscher France. :-)

Cela dit, rien ne vous empêche de fournir un binding pour vos propres interfaces de communication !

6.2 Répertoires

Vous vous êtes donc procuré "Ada for Automation" et vous vous trouvez devant un certain nombre de répertoires.

app1
C'est ici que vous trouverez les fichiers de l'application exemple 1. Cette application met en œuvre un serveur Modbus TCP et quelques clients.

app1simu
Ici que vous trouverez les fichiers de l'application de simulation pour l'application exemple 1.

app2
Dans celui-ci, vous trouverez les fichiers de l'application exemple 2. Elle met en œuvre une carte Hilscher cifX PROFIBUS DP Maître et un serveur Modbus TCP.

app3
Ici vous trouverez les fichiers de l'application exemple 3. Cette application met en œuvre un serveur Modbus TCP et un Maître Modbus RTU.

book
Le livre "Ada for Automation" est élaboré ici. Ce livre est composé en texte brut dans le format Asciidoc et traité pour fournir des fichiers HTML ou PDF.

build
Le répertoire pour les artefacts de la construction de A4A. Ayant leur propre espace, ils ne contribuent pas à l'encombrement.

doc
L'environnement de développement GNAT Pro Studio peut générer la documentation du code source automatiquement et est paramétré pour placer sa production dans ce répertoire.

exe
Vos exécutables seront placés ici.

exp
Quelques unes de mes expériences… Cela ne devrait sans doute pas être là.

hilscherx
Le répertoire du binding de l'API Hilscher cifX. Vous en aurez besoin dans le cas où vous souhaiteriez utiliser une carte Hilscher cifX qui peut vous fournir la connectivité dont vous auriez besoin pour gérer vos équipements sur tous les principaux bus de terrain Ethernet Temps Réel ou conventionnels.

src
Là bat le cœur de "Ada for Automation". Ne vous gênez pas pour vous y plonger et expérimenter.

test
L'endroit pour y placer toutes sortes d'essais et d'expérimentations ou exemples.

6.3 Projets GNAT Pro Studio

GNAT Pro Studio permet d'organiser les projets en une arborescence de projets avec des projets héritant d'autres. Cela permet d'étendre les projets en ajoutant ou substituant des fichiers de code source. Vous voudrez bien vous référer à la documentation de GPRbuild pour ce qui est des projets.

A4A/COPYING
La licence GPL que vous devriez lire attentivement.

Les projets suivants sont communs à toutes les applications CLI ou GUI.

A4A/lib_cifX32x86.gpr
Le projet utilisateur pour la librairie Hilscher cifX Device Driver.
Il est possible qu'il vous faille l'adapter à votre installation.
Nécessaire uniquement si vous utilisez une carte Hilscher cifX.

A4A/libmodbus.gpr
Le projet utilisateur pour la librairie libmodbus.
Il est possible qu'il vous faille l'adapter à votre installation.

A4A/shared.gpr
Un projet abstrait, partagé par les autres, contenant des éléments communs.

Les projets suivants sont pour les applications CLI (ligne de commande).

A4A/a4a.gpr
Ceci est le fichier projet principal.

A4A/app1.gpr
Ceci est le fichier projet de l'application exemple 1, il étend le projet principal "a4a.gpr".
Seul le programme utilisateur diffère de celui inclus dans le projet principal.

A4A/app1simu.gpr
Ceci est le fichier projet de l'application de simulation pour l'application exemple 1, il étend le projet principal "a4a.gpr".
Cette application remplace la tâche "Main_Task" du projet principal par sa propre "Main_Task" à laquelle on a retiré la gestion des clients Modbus TCP.

A4A/app2.gpr
Ceci est le fichier projet de l'application exemple 2, il étend le projet principal "a4a_hilscherx.gpr".
Cette application remplace la tâche "Main_Task" du projet principal par sa propre "Main_Task".
On a retiré la gestion des clients Modbus TCP et inclus la gestion d'une carte Hilscher cifX.
Le programme utilisateur diffère également de celui disponible dans le projet principal.

A4A/a4a_hilscherx.gpr
Celui-là étend le projet principal avec le binding de l'API Hilscher cifX Device Driver et des exemples relatifs.

A4A/a4a_exp.gpr
Ce projet étend le projet principal avec quelques expérimentations qui trouveront peut-être un jour une place dans le projet principal. Il ne devrait sans doute pas être ici…

La figure suivante présente l'arborescence de ces projets.

Les flèches en pointillé représentent les relations "with".

Ces relations existent entre un projet qui utilise les services d'un autre, ceux d'une bibliothèque, comme libmodbus ou GtkAda par exemple.

Les flèches pleines indiquent une extension de projet. On l'a déjà dit plus en amont, cela signifie que l'on ajoute ou remplace des fichiers.

Le cas "a4a.gpr" / "a4a_gui.gpr" est un peu particulier. On aurait sans doute pu considérer le second comme une extension du premier. Cependant, cela aurait encore compliqué les schémas ci-après ainsi que les répertoires pour un bénéfice ténu.

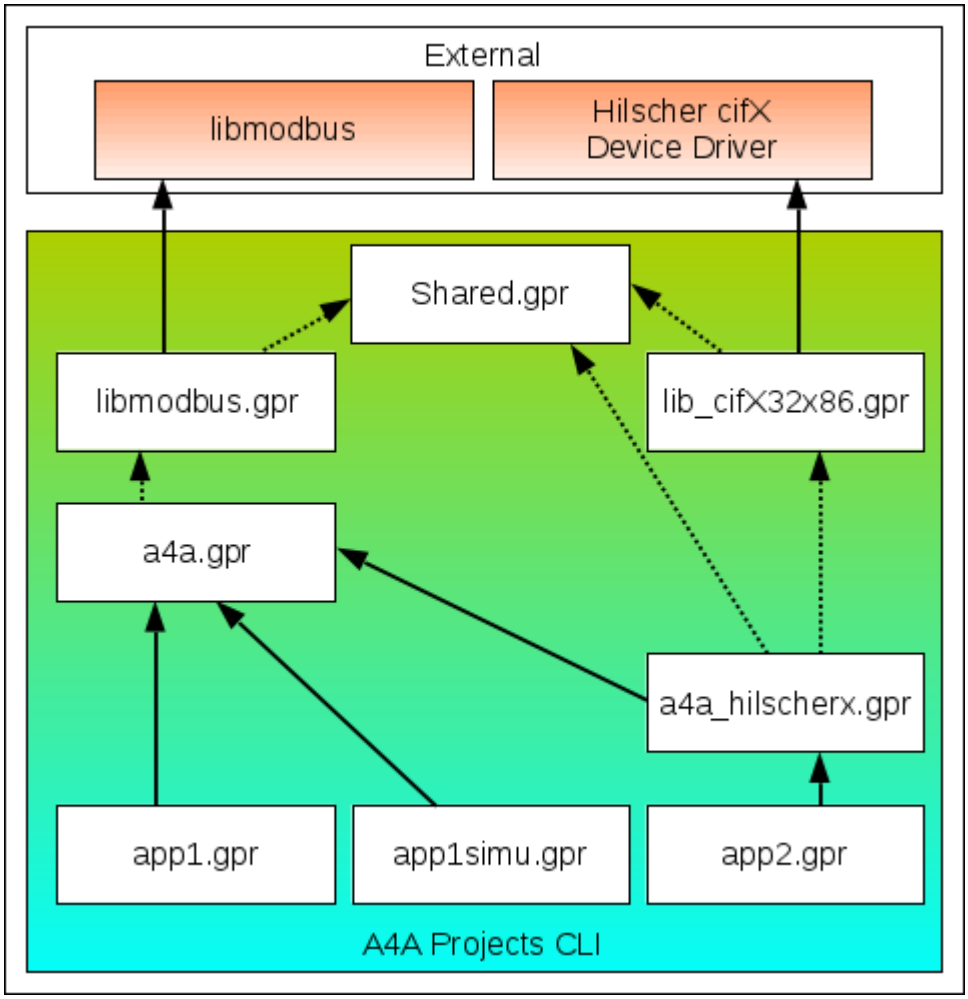

FIGURE 6.1 – A4A Arborescence des projets CLI

Les projets suivants sont pour les applications GUI (interface graphique).

A4A/a4a_gui.gpr
Ceci est le fichier projet principal avec une interface utilisateur graphique.

A4A/app1_gui.gpr
Ce fichier projet de l'application exemple 1 étend le projet principal "a4a_gui.gpr".
C'est la même application "app1" mais avec une interface graphique.

A4A/app1simu_gui.gpr
Ce fichier projet de l'application de simulation pour l'application exemple 1 étend le projet principal "a4a_gui.gpr".
C'est la même application "app1simu" mais avec une interface graphique.

A4A/app2_gui.gpr
C'est le fichier projet de l'application exemple 2 avec une GUI, il étend le projet principal "a4a_gui_hilscherx.gpr".

A4A/a4a_gui_hilscherx.gpr
Celui-là étend le projet principal muni de l'interface graphique avec le binding de l'API Hilscher cifX Device Driver et des exemples relatifs.

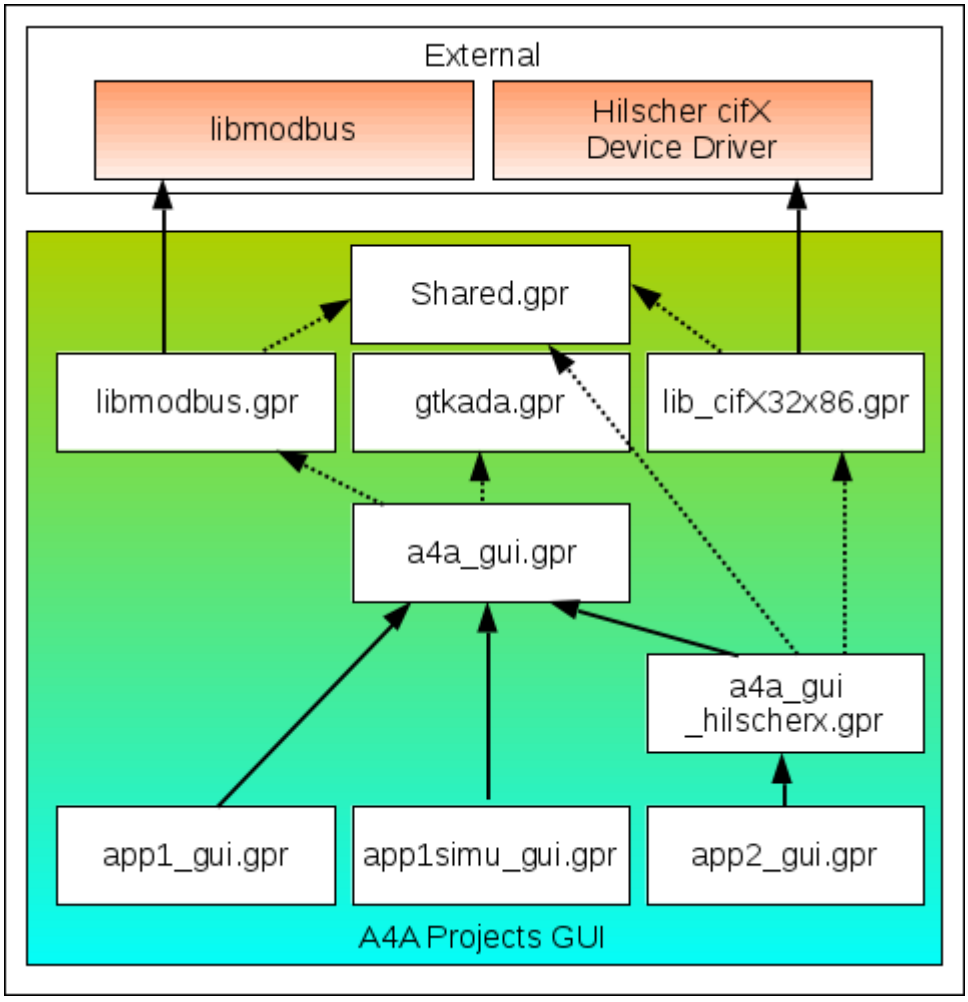

FIGURE 6.2 – A4A Arborescence des projets GUI

Chapitre 7

Conception

L'objet de ce chapitre est de documenter ce qui est implémenté, comment ça l'est et pourquoi.

7.1 Architecture générale

Le schéma ci-dessous présente l'architecture générale de "Ada for Automation".

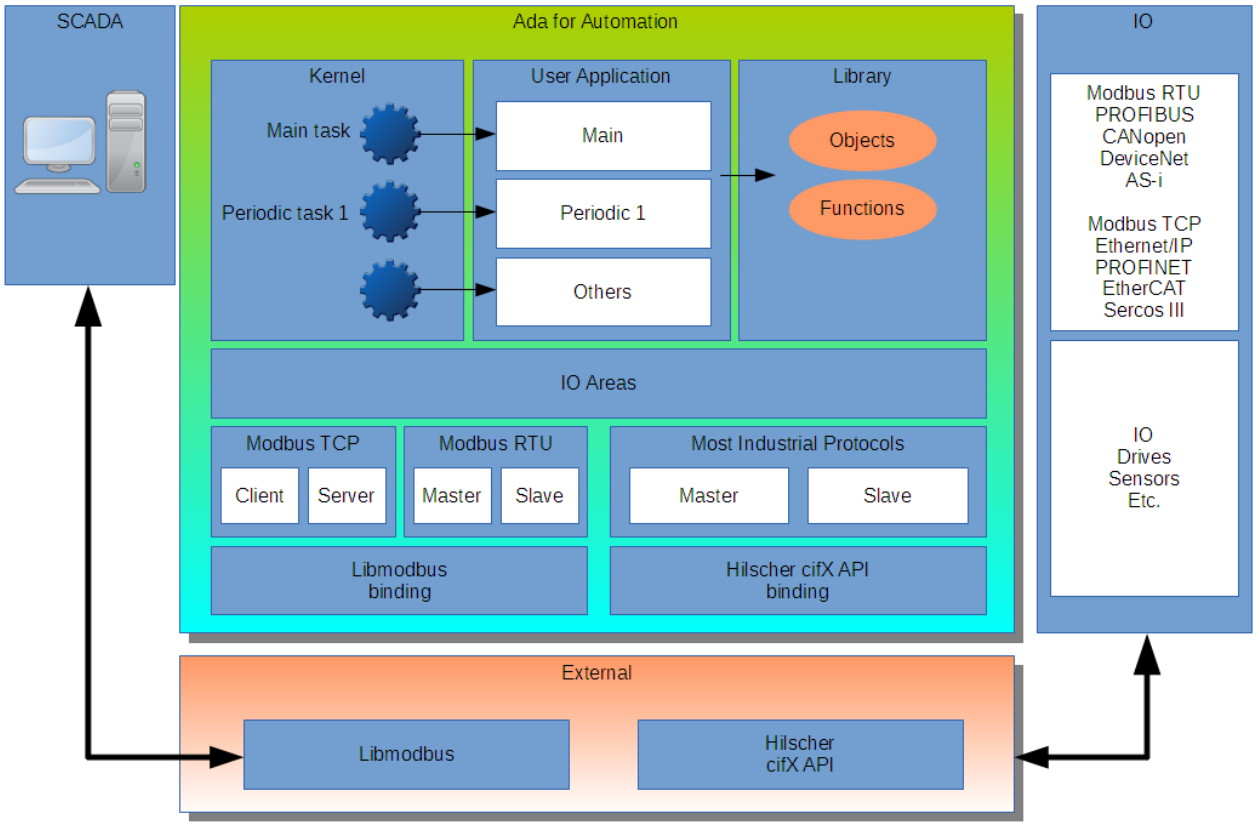

FIGURE 7.1 – A4A Architecture générale

Il se veut auto-explicatif. Le chapitre Conception est sensé en fournir une explication détaillée.

Une application console est définie en standard.

Bien sûr, il est tout à fait envisageable de créer une application avec une interface graphique par exemple avec GtkAda ou une application avec interface web avec Ada Web Server (AWS).

C'est tellement envisageable que c'est déjà disponible pour l'interface graphique et dans la "roadmap" pour AWS. ;-)

7.2 Les applications

7.2.1 Application console

Une application console est un exécutable qui dispose d'une interface ligne de commande.

L'interface en ligne de commande est certes un peu frustre mais a des avantages :

— votre application ne nécessite pas de matériel (écran, clavier, souris) ni de bibliothèque supplémentaire, c'est parfait si vous installez la CPU qui exécute celle-ci dans une armoire ou un coffret électrique,

— elle est moins lourde, sur de petites configuration ça compte,

— et elle est plus simple, ce qui est un avantage important quand on débute.

Cette application console trouve son point de départ dans la procédure "A4A_Console_Main". C'est la fonction principale qui va créer les différentes tâches qui concourent pour effectuer les actions nécessaires à votre contrôle - commande.

Le diagramme de séquence suivant montre la fonction principale "A4A_Console_Main" créer, démarrer et contrôler les tâches de l'application.

La tâche principale "Main_Task" en gèrera d'autres comme le serveur et les clients Modbus TCP.

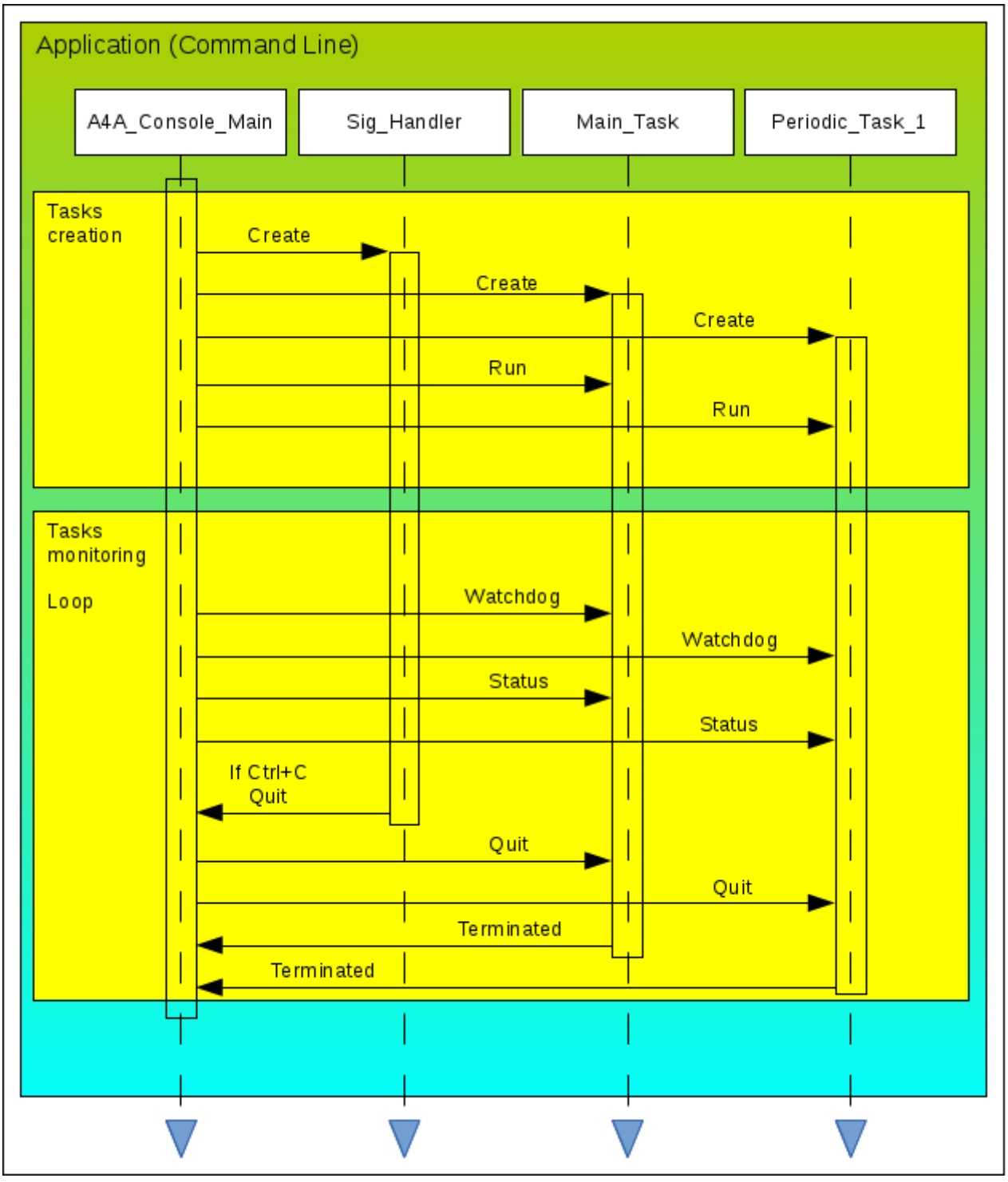

FIGURE 7.2 – A4A Application console

7.2.2 Application avec interface utilisateur graphique

Elle permet une interaction utilisateur - application bien plus riche et une prise en main plus rapide.

Elle est disponible tant sous Linux comme sous Windows® car l'on utilise le binding Ada de la bibliothèque GTK+, GtkAda, ceci avec le même source Ada.

Cette application avec IHM (ou GUI) trouve son point de départ dans la procédure "A4A_GUI_Main".

La fonction principale de l'application avec GUI diffère de l'application console :

— il n'y a pas de tâche gérant le Ctrl+C, la tâche "Sig_Handler", puisqu'il y a un bouton "Quitter",

— elle démarre l'interface graphique comportant la boucle gérant les événements souris et clavier,

— c'est l'UI qui contrôle les tâches, chien de garde, visualisation d'état, marche / arrêt… et ce périodiquement.

Ci-après, quelques copies d'écran sous Microsoft Windows 7® et sous Debian Wheezy avec l'environnement de bureau Gnome sont présentées afin d'illustrer le propos.

Pour le moment, l'application graphique persiste à écrire dans la fenêtre depuis laquelle elle est démarrée. Elle est relativement bavarde, il est possible bien sûr de la rendre moins loquace en changeant le niveau des messages de log, c'est pédagogique par défaut.

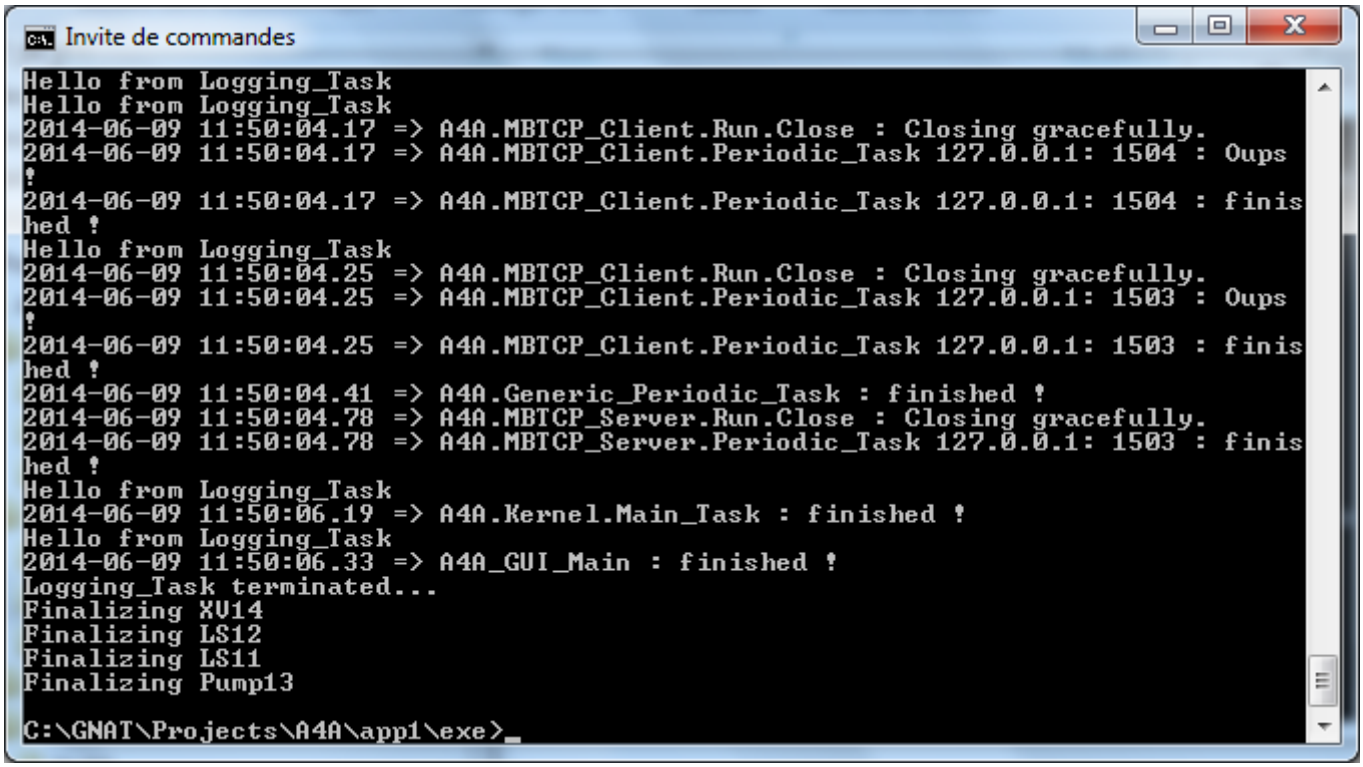

FIGURE 7.3 – A4A App1 Logs sous Microsoft Windows 7®

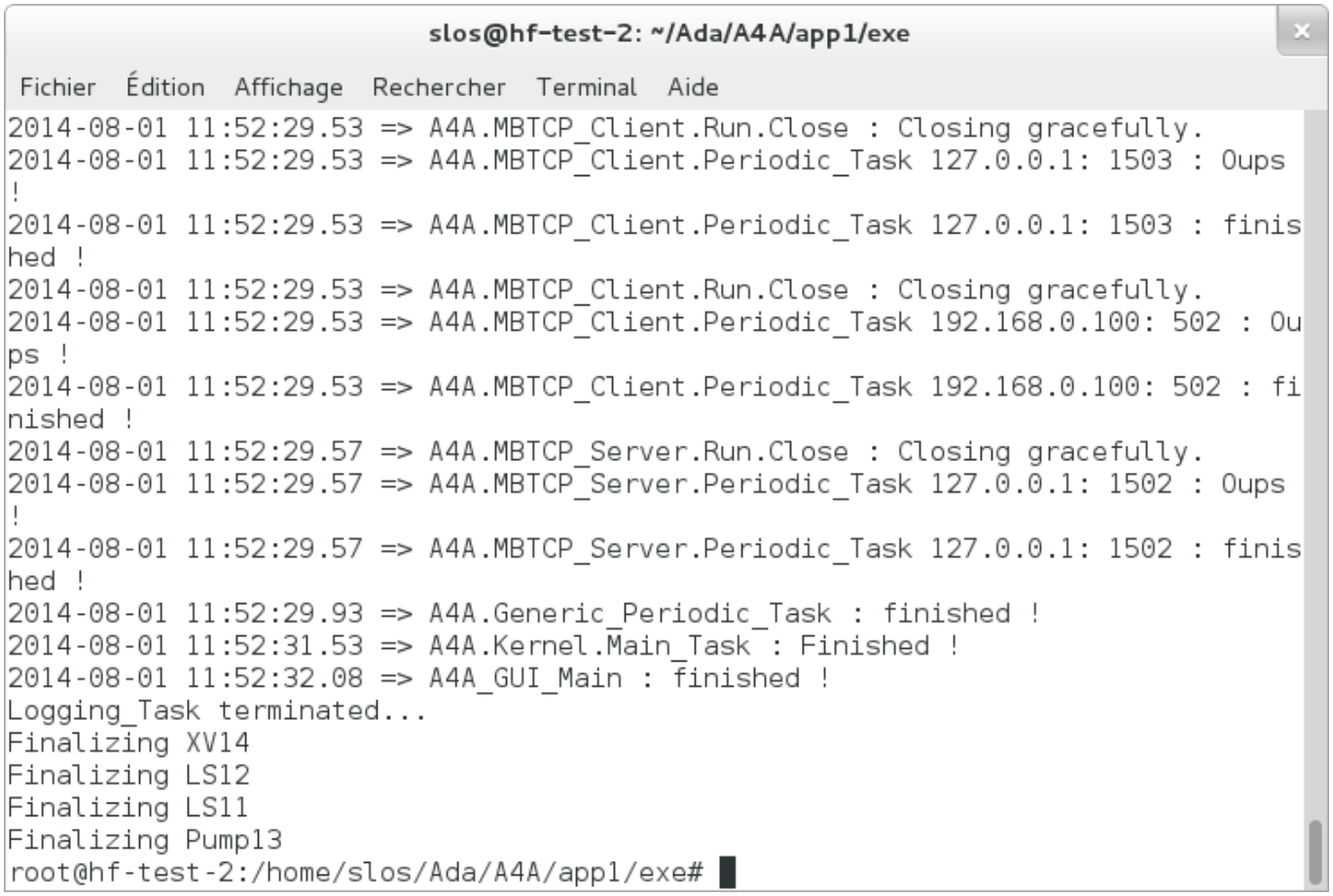

FIGURE 7.4 – A4A App1 Logs sous Debian Wheezy

Sous Linux, la commande "tee" permet de rediriger la sortie écran dans un fichier, la sortie écran est affichée en même temps qu'enregistrée.

L'interface graphique standard comporte un bandeau en partie haute et des onglets.

Dans le bandeau supérieur on trouve les boutons :

— Quit : l'équivalent du Ctrl+C disponible dans l'application en ligne de commande, qui permet de quitter l'application proprement.

— Stop : arrête le traitement du programme utilisateur. Cela n'interrompt pas les tâches de communication qui continuent donc de s'exécuter. Les sorties sont cependant mises à 0.

— Start : démarre le traitement du programme utilisateur.

L'activation de ces commandes affiche une boite de dialogue pour confirmation.

L'action "Fermer la fenêtre" est équivalente à celle du bouton "Quit" comme attendu.

Le voyant "Application status" reste pour le moment désespérément vert, il n'est pas encore animé. En effet, il n'y a pas encore consensus sur la manière d'afficher le status ni ce que cela recouvre comme signification.

Les onglets présentent :

— l'identité de l'application,

— l'état général de celle-ci,

— l'état du serveur Modbus TCP,

— l'état des clients Modbus TCP, pour les applications en disposant,

— l'état de la ou les cartes Hilscher cifX le cas échéant.

7.2.2.1 Onglet Identité

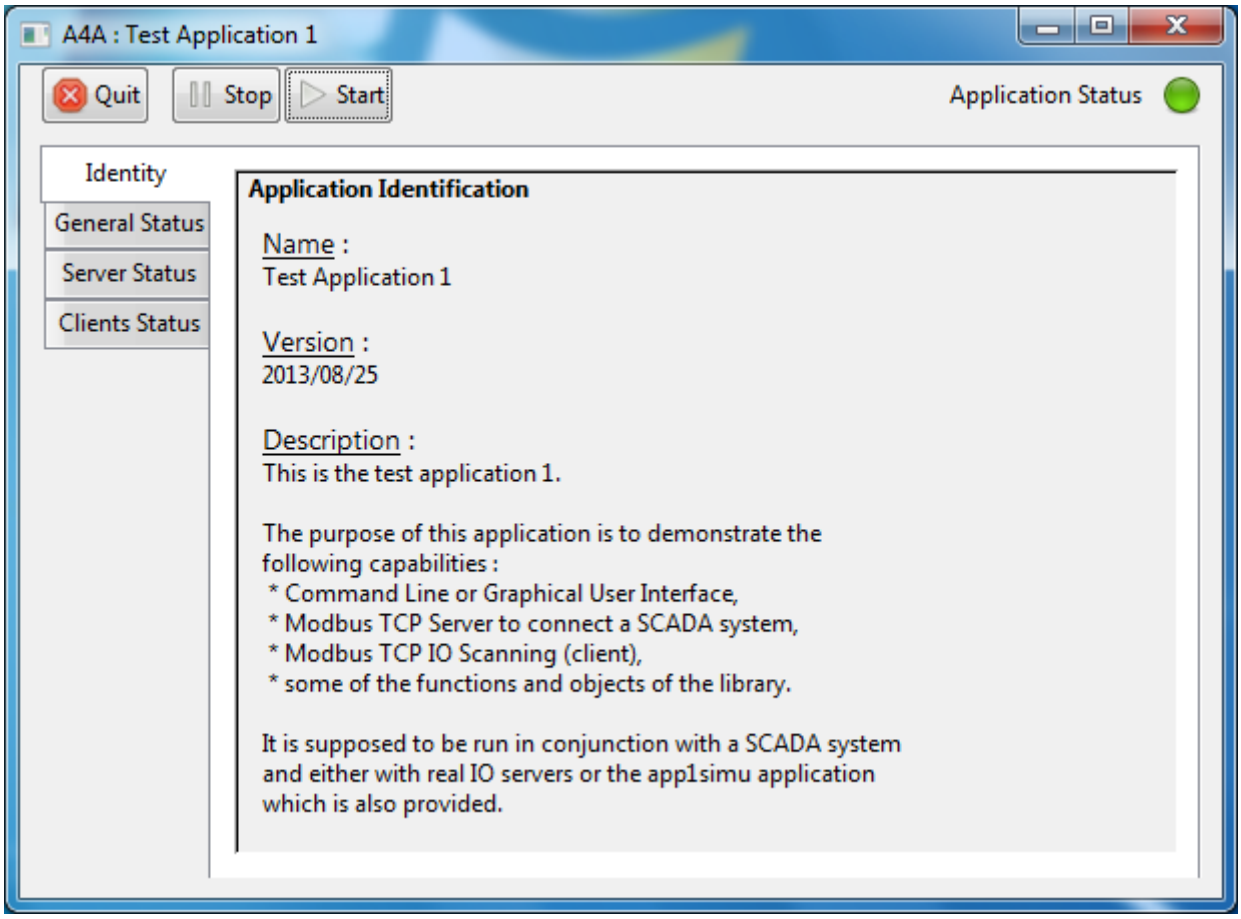

FIGURE 7.5 – A4A App1 Onglet Identity sous Microsoft Windows 7®

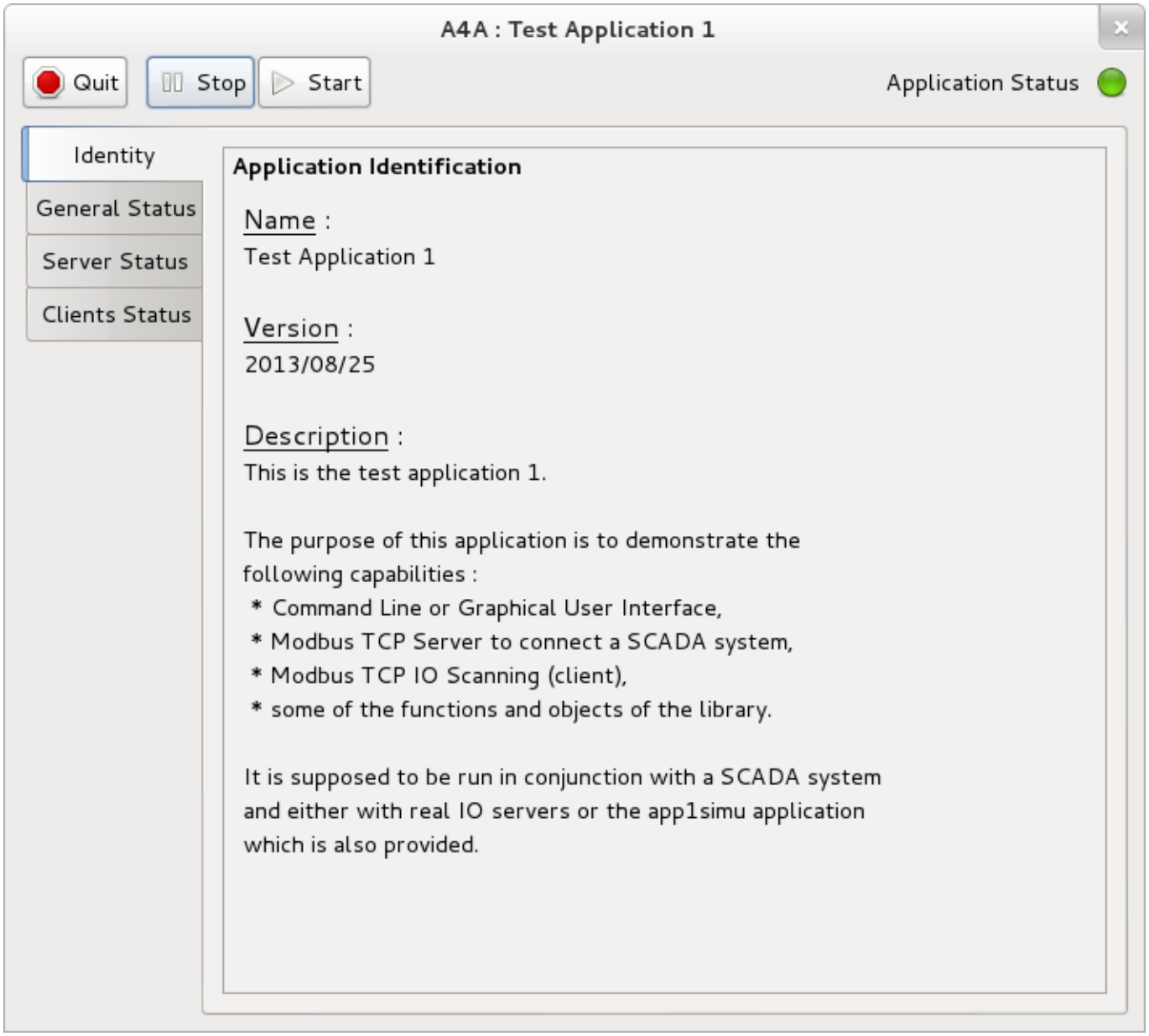

FIGURE 7.6 – A4A App1 Onglet Identity sous Debian Wheezy

Les données présentées dans cet onglet proviennent du paquetage "A4A.Application.Identification" que vous pouvez personnaliser à votre guise comme de bien entendu.

ASTUCE
Il y a lieu de modifier les données privées. L'interface du paquetage est utilisée pour obtenir celles-ci.

7.2.2.2 Onglet Etat général

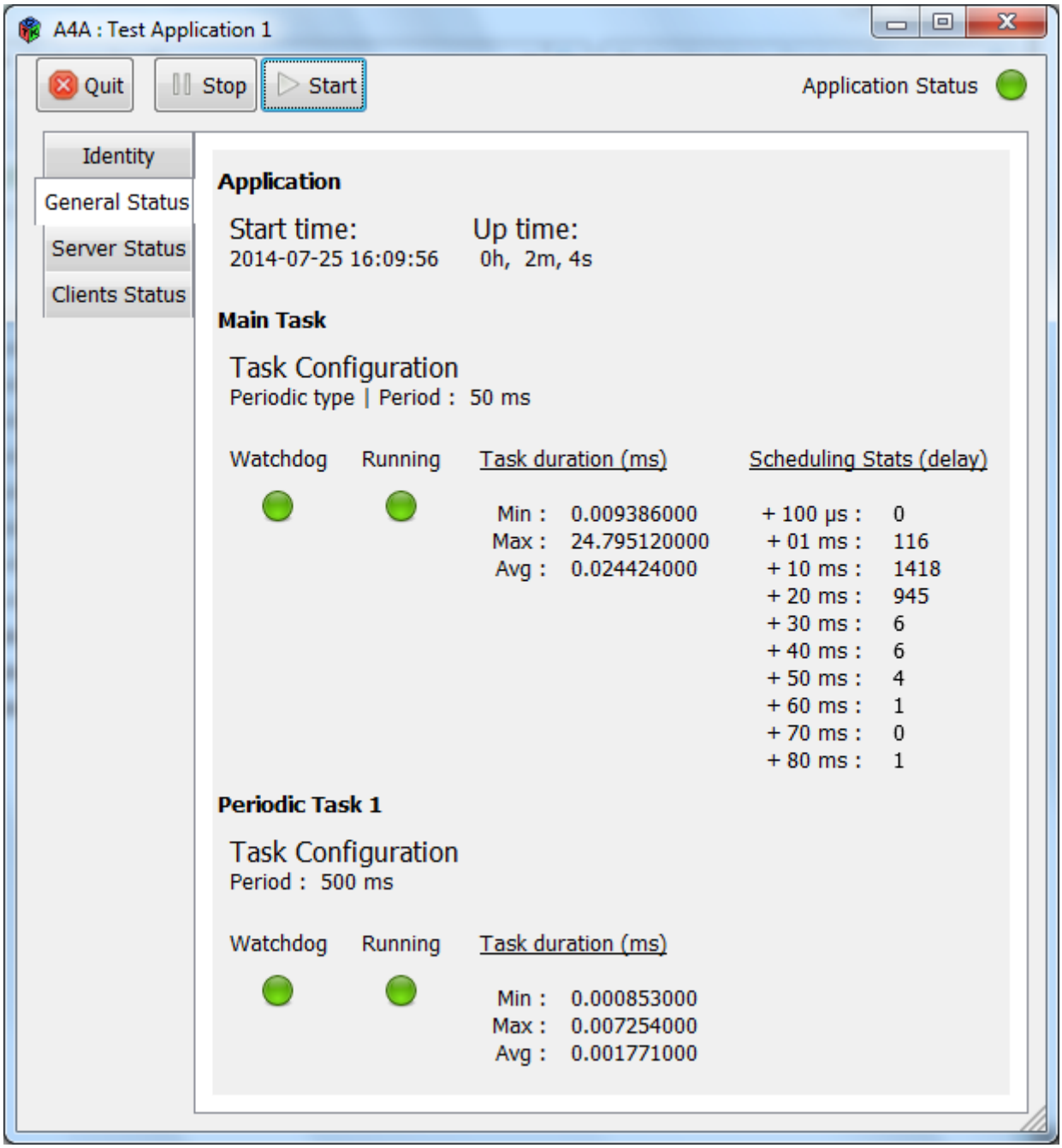

FIGURE 7.7 – A4A App1 Onglet General_Status sous Microsoft Windows 7®

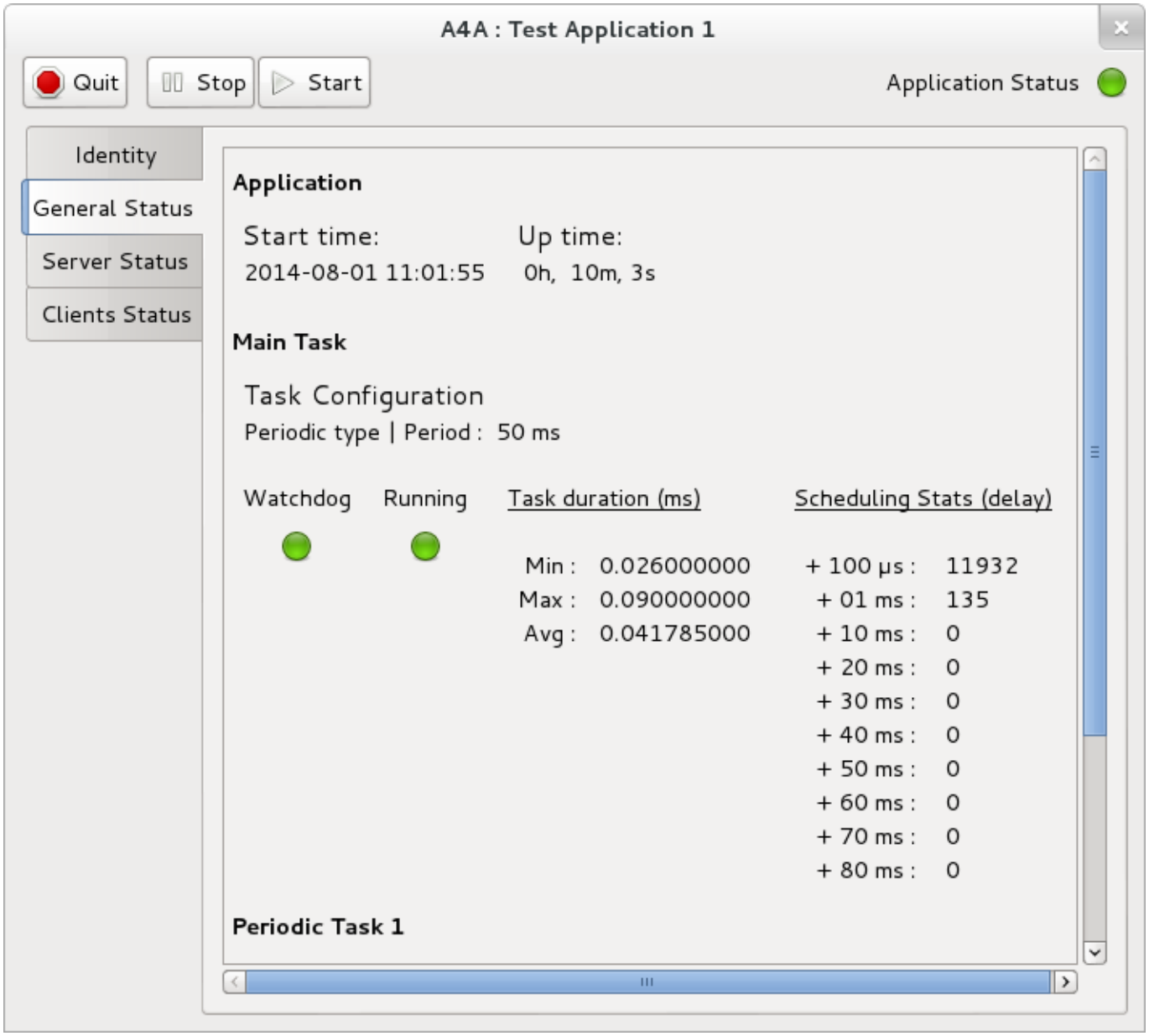

FIGURE 7.8 – A4A App1 Onglet General_Status sous Debian Wheezy

Cet onglet indique :

— la date et l'heure de démarrage de l'application ainsi que la durée de fonctionnement en heures, minutes et secondes,

— pour chaque tâche, le type, cyclique ou périodique, le délai ou la période, l'état du chien de garde, l'état de marche, quelques statistiques sur la durée d'exécution, min, max, moyenne,

— pour la tâche principale, des statistiques sur le retard par rapport à l'ordonnancement prévu, ce qui permet d'apprécier à la louche le comportement plus ou moins temps réel.

Note
Il faut se garder de tout jugement hâtif sur ces statistiques.
Par exemple, le maximum de la durée d'exécution est assez trompeur. Ce n'est pas tant que la tâche ait duré plus longtemps, c'est plutôt qu'elle a été préemptée par une ou plusieurs autres tâches.
Il faut préciser ici que l'on utilise la version avec le patch PREEMPT_RT du noyau Linux disponible dans Debian Wheezy.
Ce qui explique les différences constatées au niveau des statistiques d'ordonnancement bien meilleures.
Cependant on peut améliorer sensiblement le comportement de Windows® comme montré dans cet article :
A4A : Améliorer le temps réel mou sous Windows

7.2.2.3 Onglet Etat Serveur Modbus TCP

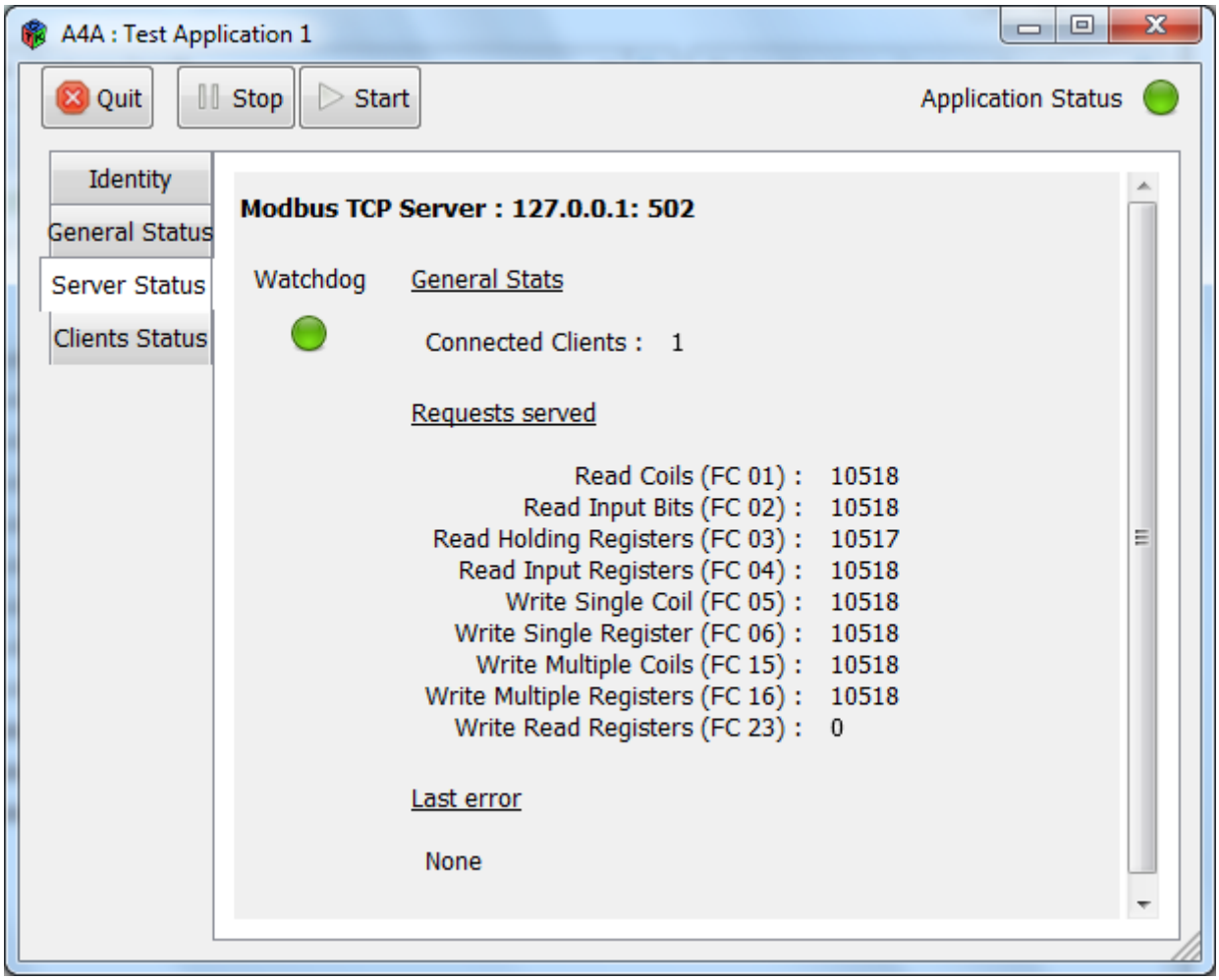

FIGURE 7.9 – A4A App1 Onglet MBTCPServer sous Microsoft Windows 7®

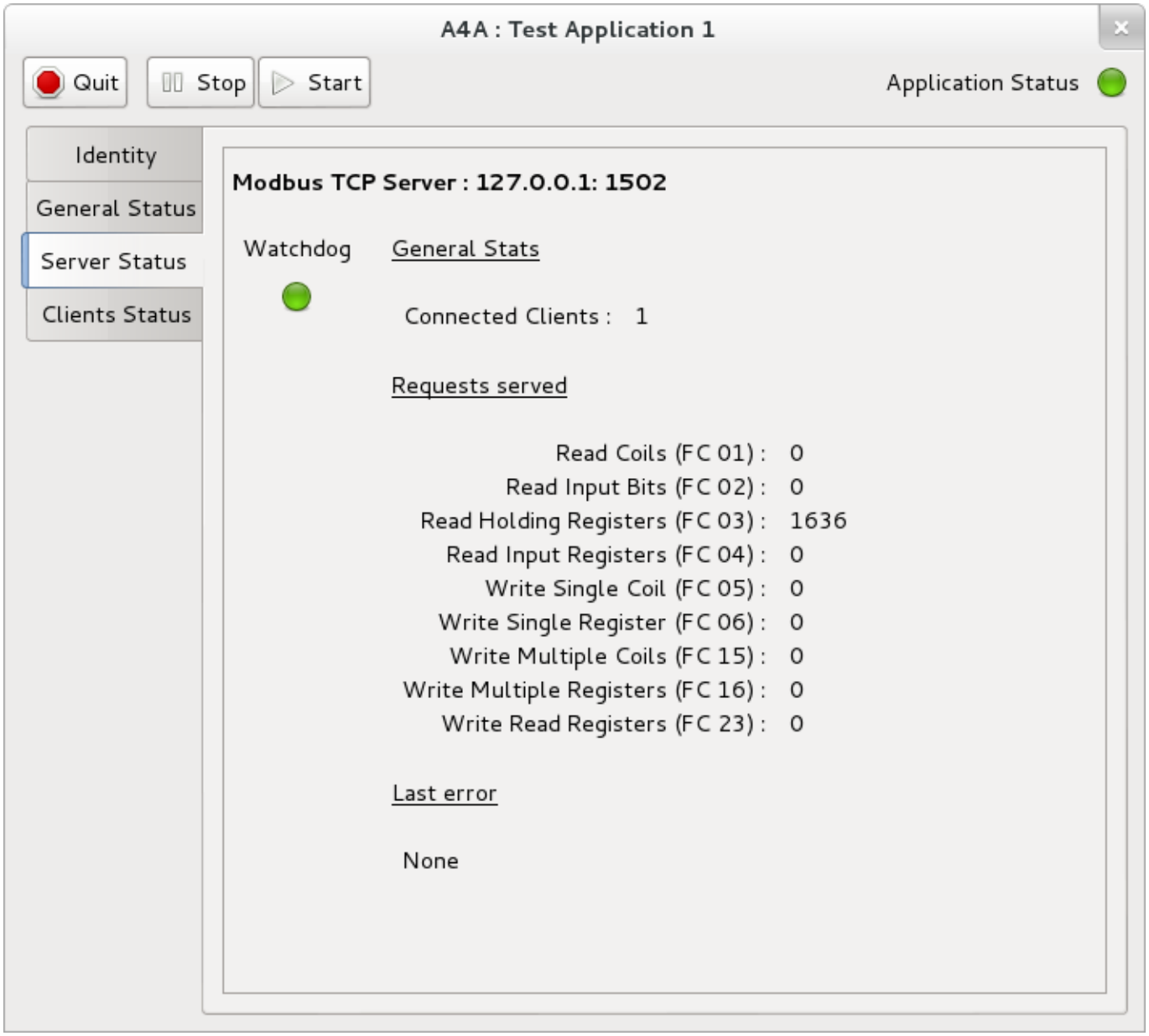

FIGURE 7.10 – A4A App1 Onglet MBTCPServer sous Debian Wheezy

Cet onglet affiche les statistiques du serveur Modbus TCP intégré.

Son adresse IP locale, pas très utile mais c'est pour rester cohérent avec ce qu'affiche le client, et le port utilisé sont indiqués.

Note
Le port Modbus TCP standard est le port 502. Cependant, sous Linux il est nécessaire d'exécuter l'application avec des droits "root" si celle-ci utilise un port inférieur à 1000. Donc on choisit un port supérieur à 1000, par exemple 1502, si l'on n'est pas obligé d'utiliser le port 502.

On y trouve le nombre de clients connectés et des compteurs indiquent pour chacune des fonctions supportées le nombre de requêtes servies.

Cette vue montre une exécution sous Microsoft Windows 7®, on peut très bien utiliser le port 502 comme on le voit. Le client Modbus TCP connecté est une carte Hilscher cifX exécutant un firmware Modbus TCP configuré en client avec tous les types

de requêtes supportés. Le client Hilscher ne supporte pas la fonction 23, lecture / écriture de registres, le serveur "Ada for Automation" oui, ce que l'on peut tester avec un autre client ou avec le client "Ada for Automation" comme ci-dessous démontré.

Le client Hilscher se configure avec une table de commandes dans laquelle on a spécifié toutes les requêtes. Elles sont ici exécutées sans délai et le serveur "Ada for Automation" en sert environ 500 par seconde.

7.2.2.4 Onglet Etat Clients Modbus TCP

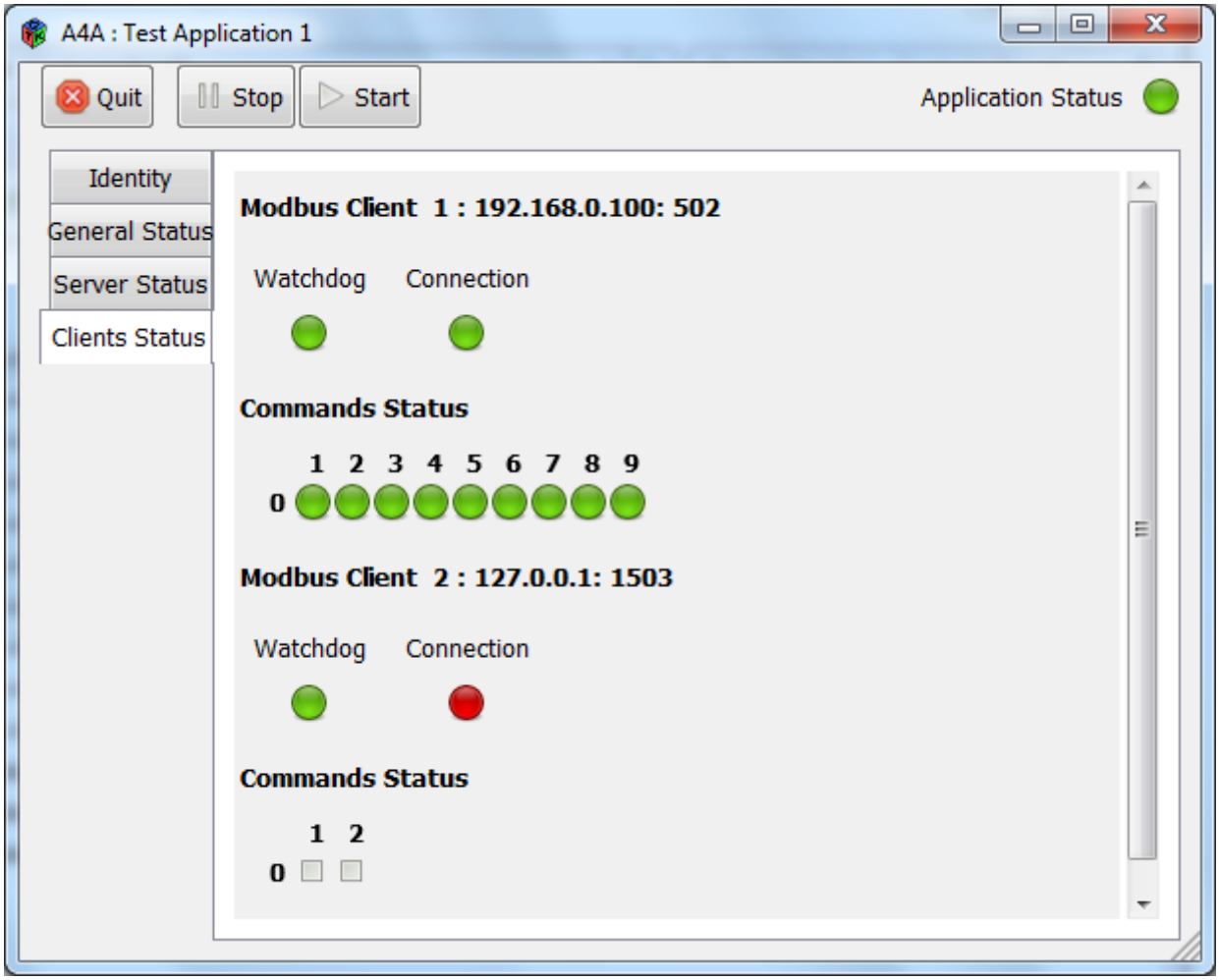

FIGURE 7.11 – A4A App1 Onglet MBTCPClients sous Microsoft Windows 7®

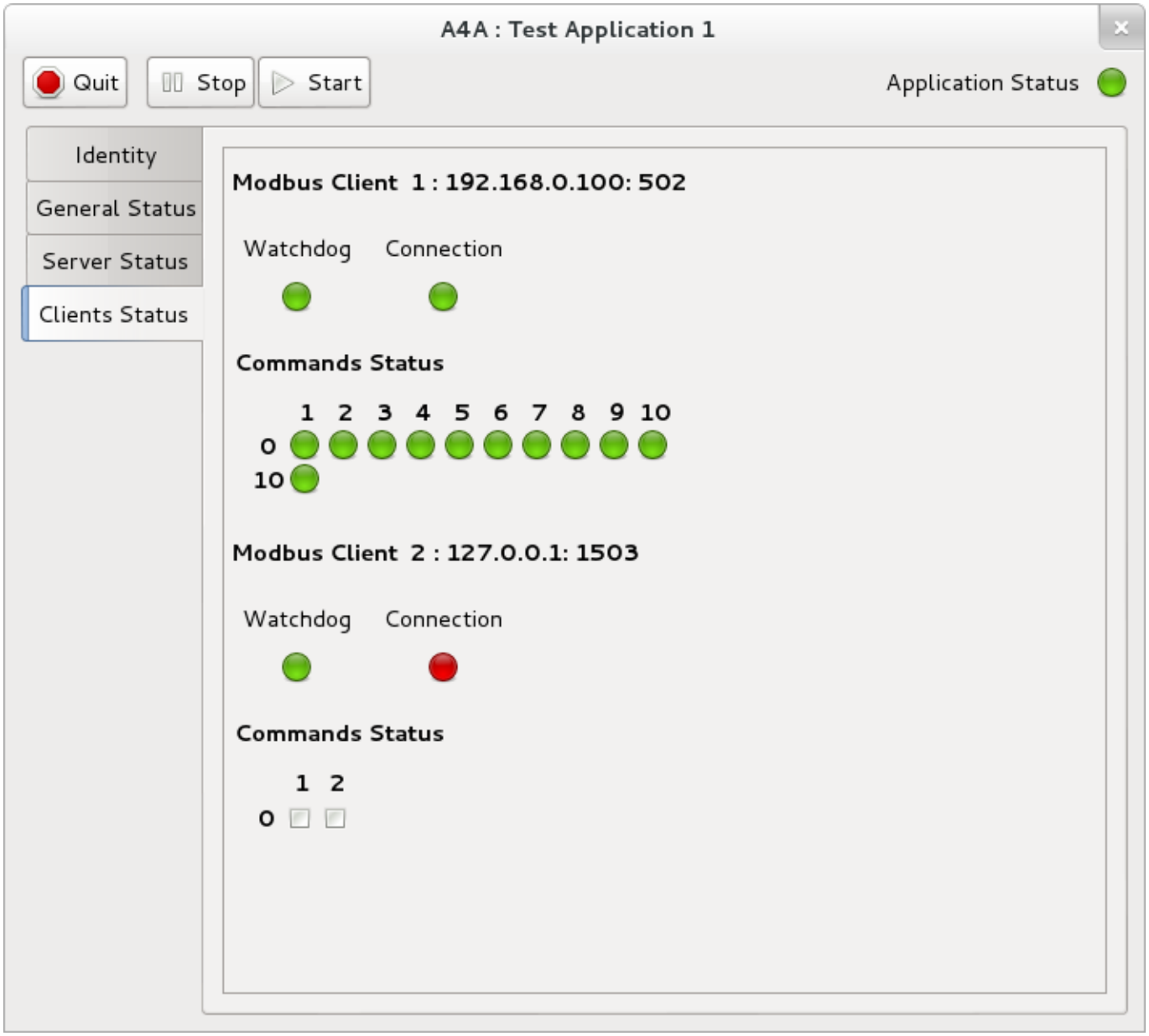

FIGURE 7.12 – A4A App1 Onglet MBTCPClients sous Debian Wheezy

Cette vue montre les clients Modbus TCP au travail configurés ainsi.

Note
L'interface graphique s'adapte automatiquement en fonction de la configuration.

7.2.2.5 Onglet Etat Maitre Modbus RTU

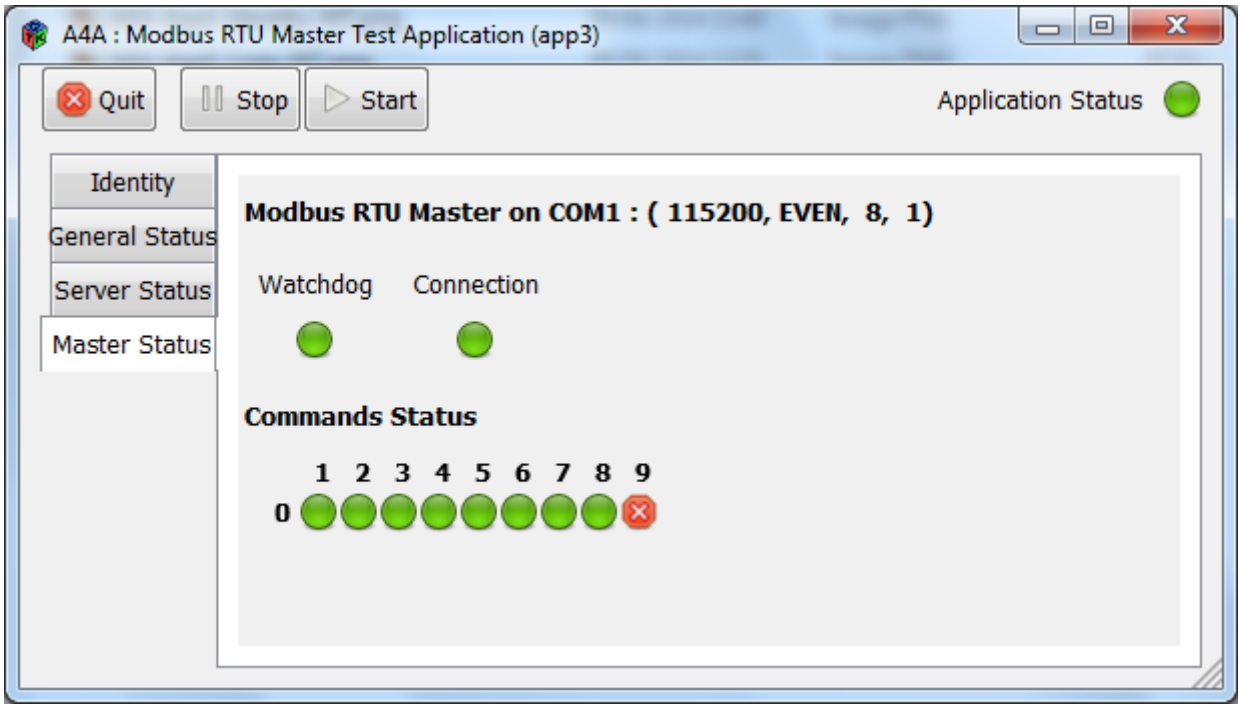

FIGURE 7.13 – A4A App3 Onglet MBRTUMaster sous Microsoft Windows 7®

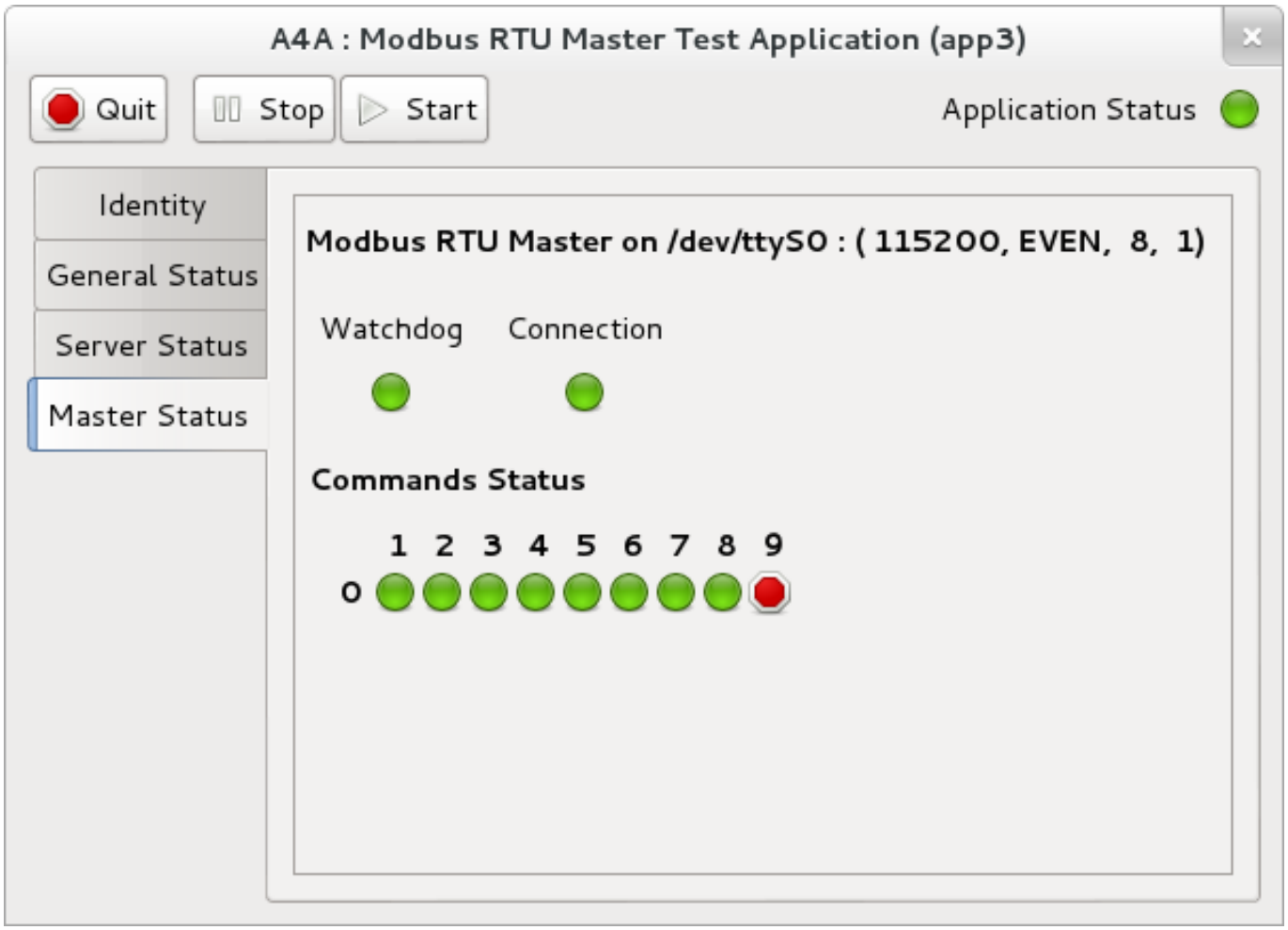

FIGURE 7.14 – A4A App3 Onglet MBRTUMaster sous Debian Wheezy

Ici c'est le Maitre Modbus RTU configuré de la sorte, l'unique différence entre la version pour Microsoft Windows 7® et la version Linux étant le "Device", "COM1" pour l'un et "/dev/ttyS0" pour l'autre.

La fonction "Write_Read_Registers" n'est pas supportée par l'esclave utilisé, Modbus PLC Simulator.
Aussi elle est désactivée, ce qui est signalé par la croix rouge.

Note
L'interface graphique s'adapte automatiquement en fonction de la configuration.

7.2.2.6 Onglet Etat Hilscher cifX

Cette vue montre l'état général d'une cifX 50-RE exécutant un firmware Ethernet/IP Scanner.

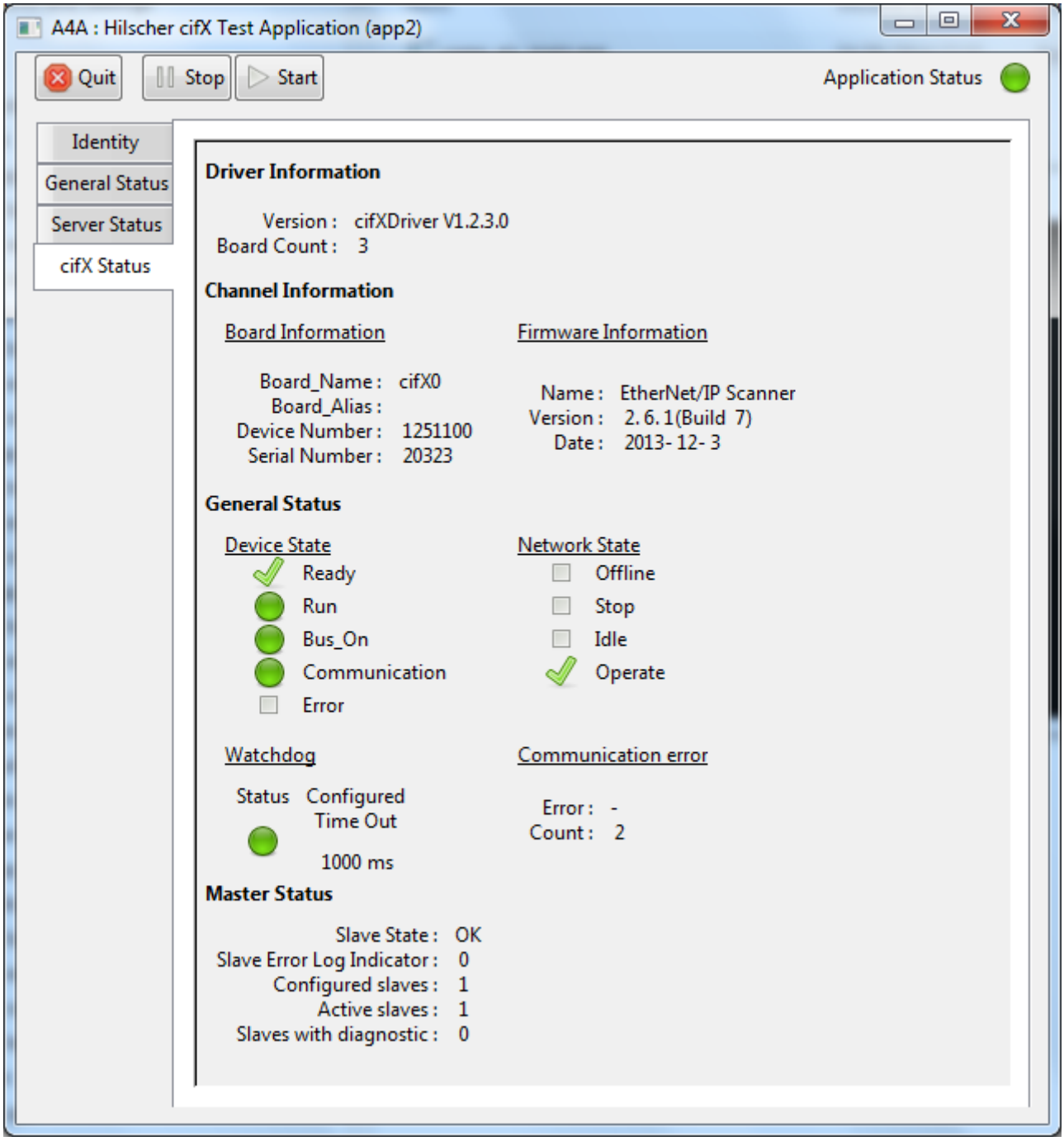

FIGURE 7.15 – A4A App2 Onglet cifXStatus (Ethernet/IP Scanner) sous Microsoft Windows 7®

Il est aussi possible d'utiliser d'autres protocoles. L'application exemple "app2" utilise une carte Hilscher cifX PROFIBUS DP Maitre.

La vue suivante montre l'état général d'une cifX 50-DP exécutant un firmware PROFIBUS DP Maitre.

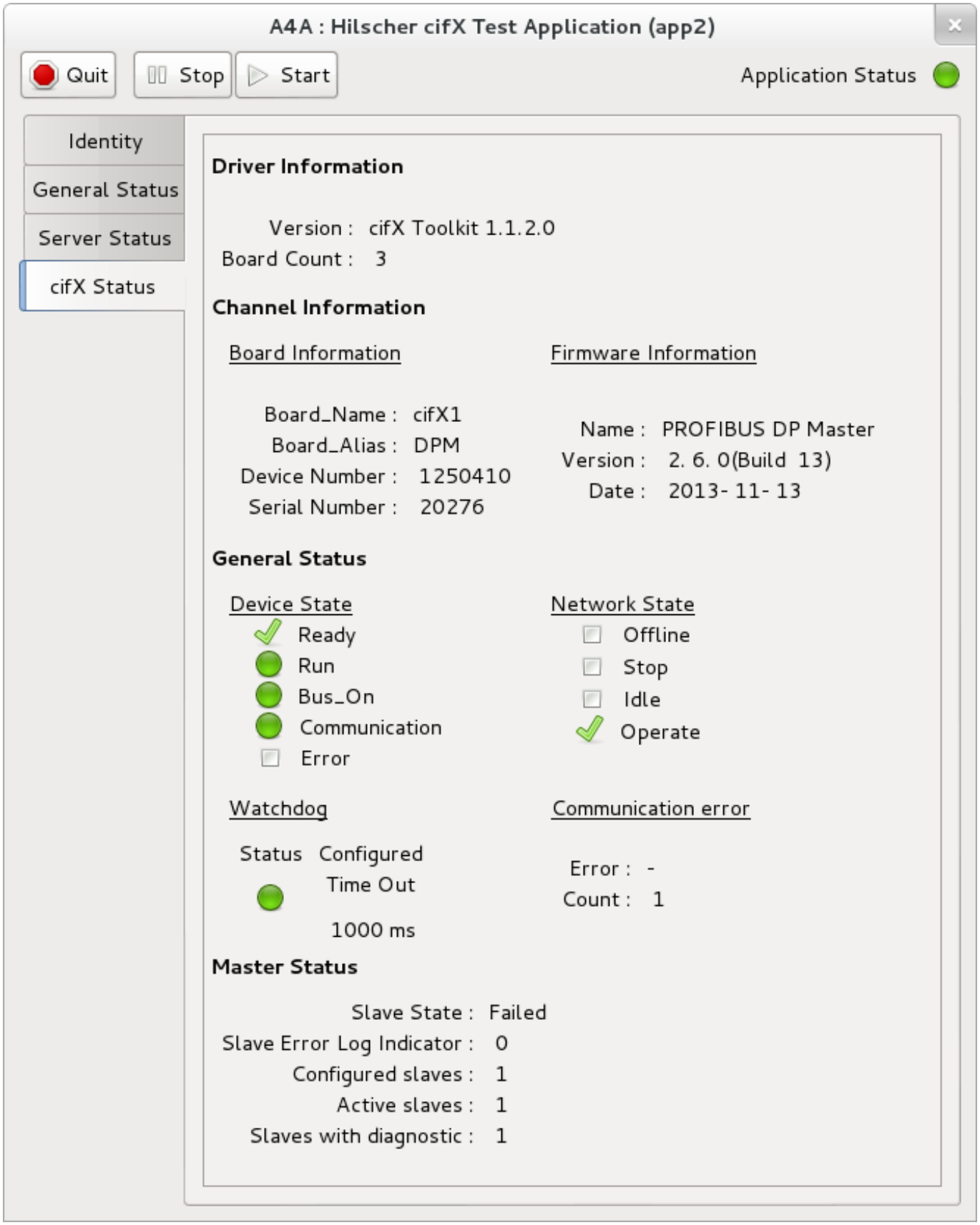

FIGURE 7.16 – A4A App2 Onglet cifXStatus (PROFIBUS DP Maitre) sous Debian Wheezy

7.3 Les paquetages

Les paquetages, packages en Anglais, permettent d'organiser le code Ada en regroupant les éléments de l'application, les types de données, les données, les fonctions et procédures, les objets, les tâches, d'autres paquetages, etc... selon une hiérarchie logique.

Les paquetages fils héritent du paquetage père, les choses définies dans celui-ci leurs sont visibles sans qu'il y ait besoin de les mentionner dans une clause "with". Ainsi, tous les paquetages fils de "A4A" voient ce qui y est défini et peuvent y utiliser sans restriction.

Une clause "with" définit une dépendance du paquetage où elle apparaît vers celui objet de la clause.

On peut représenter ces dépendances entre paquetages dans un diagramme, ce qui risque d'être fait ultérieurement. :-)

7.4 Les tâches

Nous avons abordé au chapitre Concepts le concept de tâche, une tâche représentant un fil d'exécution d'instructions. Plusieurs tâches peuvent s'exécuter de façon concurrente, le ou les processeurs travaillant sur les instructions et les données de chaque tâche selon un ordonnancement piloté par le système d'exploitation et, dans le cas de Ada, par l'environnement d'exécution.

En Ada, il est très simple de créer une tâche, le concept est disponible dans le langage.

Lorsque l'on conçoit une application multi-tâches, un problème récurrent est celui du partage des données et du corollaire, l'intégrité de ces données partagées ou leur cohérence.

Ce problème trouve en Ada une solution également simple puisque l'on peut par exemple définir un type protégé (protected type) qui encapsule les données critiques et permet l'accès concurrent via des fonctions et procédures qui se verrouillent mutuellement.

Un exemple d'utilisation de type protégé est donné par la scrutation sur Modbus TCP avec la "Dual Port Memory" ou mémoire double accès, un composant qui permet la lecture par plusieurs entités simultanément tandis que l'écriture n'est possible que par une seule entité et verrouille la lecture comme l'écriture.

Par principe, les tâches dans "Ada for Automation" disposent d'une interface d'un type protégé permettant le contrôle de la tâche (Quit, Run, Watchdog) et la récupération de son état.

Les tâches en relation surveillent un signe de vie mutuel, c'est le chien de garde.

Votre application est donc constituée de tâches, deux prédéfinies, la tâche principale et la tâche périodique, des tâches annexes, et celles que vous pourriez rajouter en fonction de vos besoins.

7.4.1 Main_Task

La tâche "Main_Task" définie dans le paquetage "A4A.Kernel.Main" peut être configurée pour s'exécuter cycliquement ou périodiquement.

Cette configuration a lieu dans le paquetage "A4A.Application.Configuration".

Configurée pour s'exécuter cycliquement, elle sera à la fin de l'itération programmée pour une exécution après un laps de temps déterminé. Par exemple, la tâche dure 200 millisecondes. A la fin de son exécution, elle est programmée pour s'exécuter à nouveau 100 millisecondes plus tard. Sa période varie en fonction de la durée de la tâche.

Si elle est configurée pour une exécution périodique, elle sera exécutée avec une période déterminée, plus ou moins précise en fonction du système d'exploitation utilisé.

Périodique ou cyclique ? Cela dépend de la nature de votre application et des composants nécessaires. Si vous ne pilotez que du tout ou rien, des pompes, des vannes, des voyants, etc. une exécution cyclique convient très bien. Si vous utilisez des objets dont le comportement dépend de la notion de temps, comme un régulateur PID ou un générateur de signaux, il est nécessaire que le traitement de ces objets se fasse à une période la plus précise possible.

Cette tâche appelle la procédure "Main_Cyclic" définie par l'utilisateur dans le paquetage "A4A.Application". Cette procédure est donc le point d'entrée pour l'application utilisateur comme peut l'être le bloc d'organisation OB1 chez Siemens. Si l'état de l'application le permet, c'est à dire que l'utilisateur a activé le traitement et qu'aucune condition (un défaut interne ou applicatif)

ne s'y oppose, elle est donc en état "Marche", la procédure principale utilisateur est appelée et les sorties sont pilotées. Sinon les sorties sont pilotées à 0.

La tâche "Main_Task" telle qu'elle est définie dans le projet de base gère un serveur Modbus TCP et des clients Modbus TCP en fonction de la configuration à piloter.

Comme l'on peut substituer un fichier du projet de base dans un projet d'extension, il est tout à fait possible de redéfinir cette tâche principale en fonction du besoin.

Ainsi, dans le projet "app1simu", comme il n'est pas nécessaire de mettre en œuvre des clients Modbus TCP, la tâche est redéfinie sans.

De même, dans le projet "app2" qui utilise une carte Hilscher cifX pour communiquer avec des équipements PROFIBUS DP, la tâche principale est également différente de celle de base.

Nous utilisons le terme de "IO Manager" pour signifier le composant en charge de la scrutation des E/S sur le bus de terrain. Ce peut donc être le client Modbus TCP, le maitre Modbus RTU, la carte Hilscher ou autres.

La tâche "Main_Task" met à jour la mémoire image des entrées du process, traite le programme applicatif et affecte la mémoire image des sorties. Cela permet durant toute la durée du traitement du programme d'avoir une cohérence des informations.

La figure suivante montre une structure somme toute très commune en automatisme.

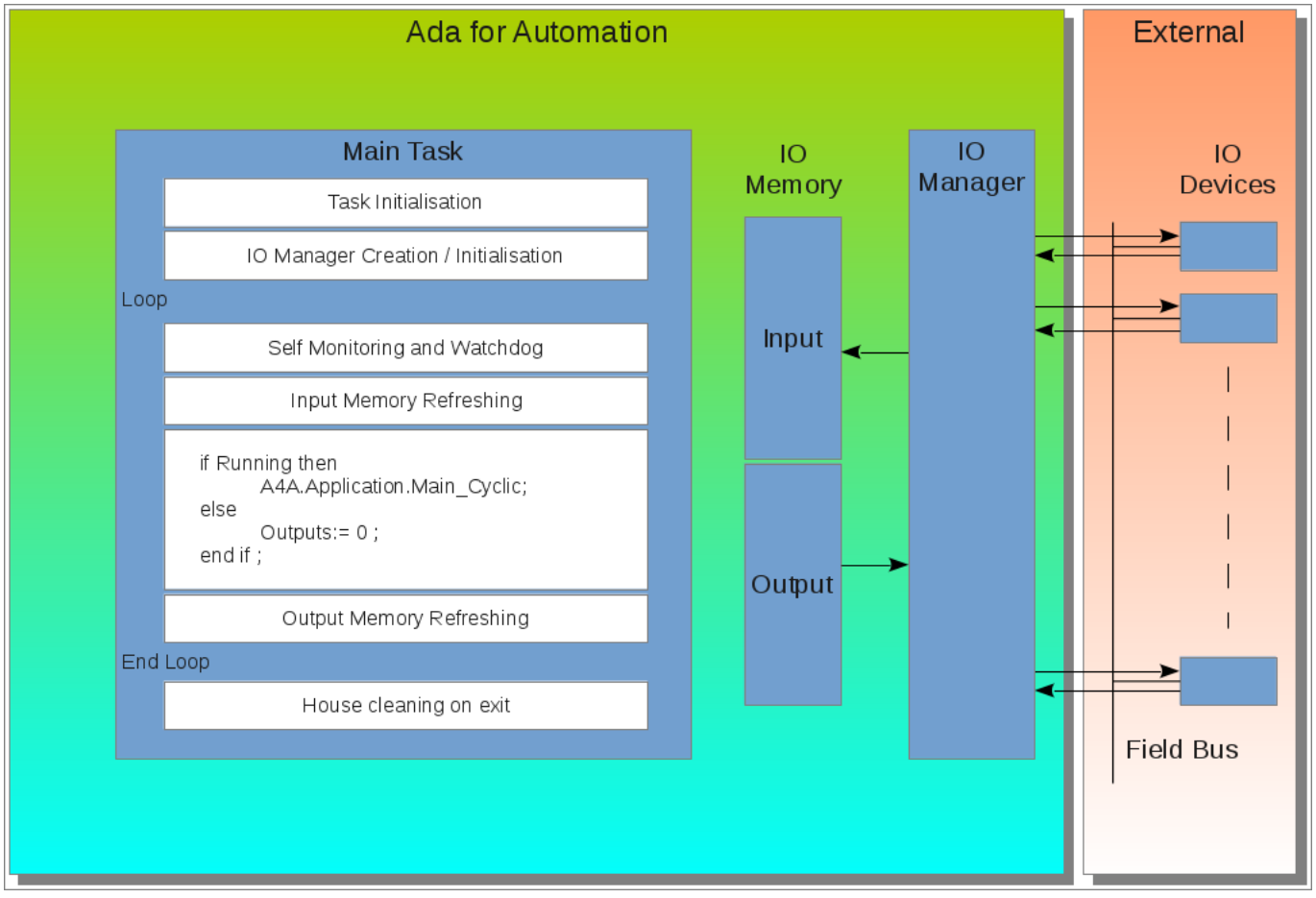

FIGURE 7.17 – A4A Tâche Principale

7.4.2 Periodic 1

La tâche "Periodic 1" peut être configurée pour s'exécuter périodiquement.

Cette configuration a lieu dans le paquetage "A4A.Application.Configuration".

Cette tâche appelle la procédure "Periodic_Run_1" définie par l'utilisateur dans le paquetage "A4A.Application".

Aujourd'hui, les données (DPM, objets utilisateur…) sont partagées entre les tâches "Main_Task" et "Periodic 1".

AVERTISSEMENT
Il serait préférable de gérer ceci via un objet protégé.

7.4.3 Sig_Handler

Cette tâche annexe est démarrée par l'application console.

Son seul travail consiste à attendre l'interruption déclenchée par l'utilisateur qui souhaite quitter l'application proprement avec la combinaison de touches Ctrl+C.

7.4.4 Clock_Handler

Cette tâche annexe est démarrée par la tâche "Main_Task".

En automatisme on utilise beaucoup les temporisations. Si chaque instance de temporisateur appelle la fonction "Ada.Real_Time.Clock" sur un programme utilisateur conséquent, l'auteur craint que cela ne soit pas souhaitable.

C'est une tâche qui s'exécute périodiquement, avec une période qui définit donc le rafraîchissement de la donnée horloge. C'est cette donnée qui est utilisée par les temporisateurs.

7.5 Scrutation sur Modbus TCP (IO Scanning)

Pour chaque serveur Modbus TCP, l'on utilise une tâche cliente qui sera créée automatiquement par la tâche "Main_Task" à partir de la configuration fournie. Ainsi, si un serveur est lent ou ne répond plus, la scrutation des autres est toujours assurée de façon optimale.

Chaque tâche cliente s'exécute périodiquement, cette période étant définie par le paramètre "Task_Period_MS" dans la configuration de la tâche.

Chaque commande possède un coefficient multiplicateur, le paramètre "Period_Multiple", qui définit donc la période de cette commande ("Task_Period_MS" x "Period_Multiple"), et un décalage, défini par le paramètre "Shift", qui permet d'échelonner dans le temps plusieurs commandes arrivant à échéance afin que le serveur ne soit pas saturé.

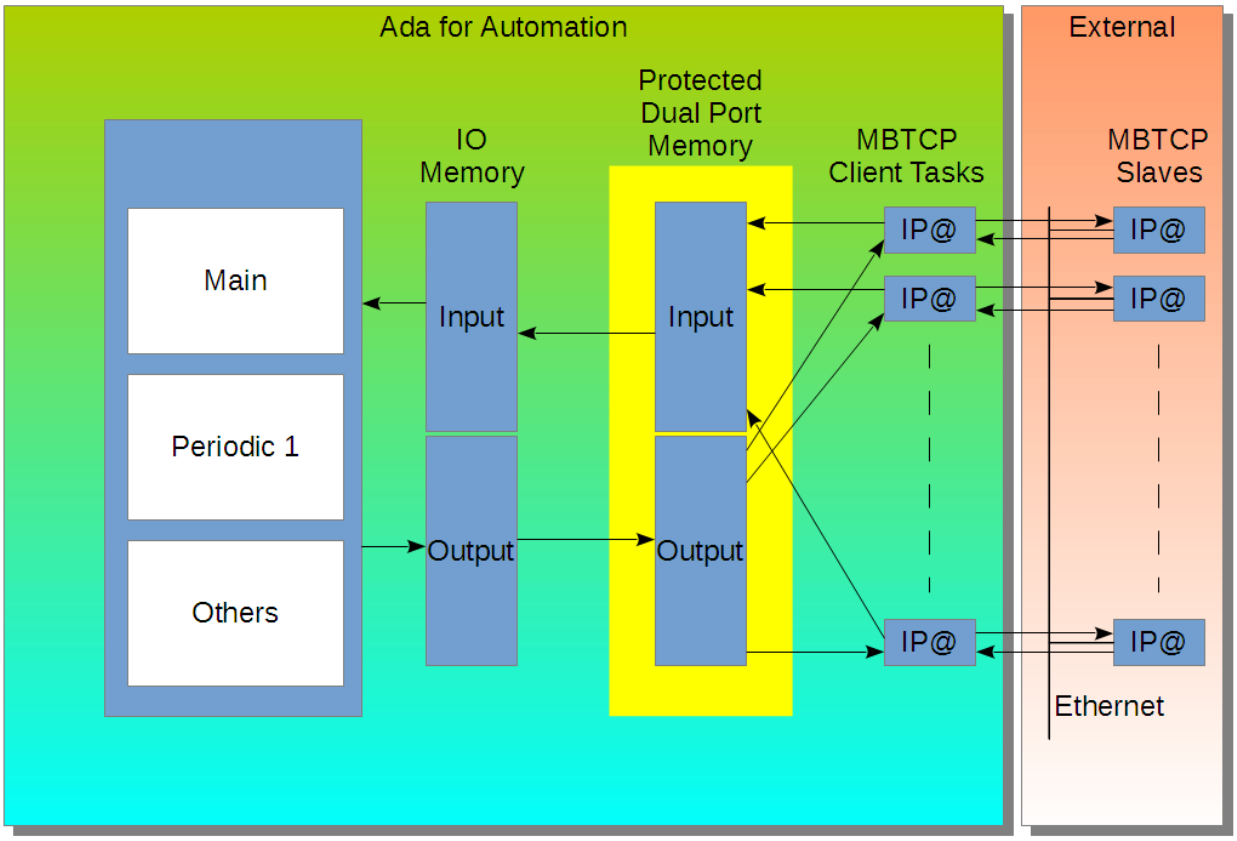

FIGURE 7.18 – A4A Architecture des Clients Modbus TCP

La "Dual Port Memory" est en fait un objet protégé qui encapsule un tableau de registres en entrée et un en sortie. La tâche "Main_Task" gère la mise à jour des zones mémoire Entrée / Sortie par rapport à la DPM, zones d'E/S auxquelles le programme utilisateur a accès.

Les types de données utilisés dans le protocole Modbus sont les registres (ou mot de 16 bits) et les booléens (bobines et bits d'état). Le protocole définit deux modèles de données, l'un dans lequel les données bits et registres sont disjointes, l'autre où il y a recouvrement. Nous préférons le premier modèle et il a donc été défini des zones registres et des zones booléens, respectivement des DPM registres et des DPM booléens.

Le schéma est ainsi incomplet car il y a en sus une construction identique pour les booléens.

Pour ce qui est de la configuration, celle-ci est réalisée dans le paquetage "A4A.Application.MBTCP_Clients_Config" comme ici pour l'application exemple "App1" :

```
with A4A.MBTCP_Client; use A4A.MBTCP_Client;
with A4A.Protocols; use A4A.Protocols.IP_Address_Strings;
with A4A.Protocols.LibModbus; use A4A.Protocols.LibModbus;

package A4A.Application.MBTCP_Clients_Config is

   ------------------------------------------------------------------
   -- Modbus TCP Clients configuration
   ------------------------------------------------------------------

   -- For each Modbus TCP Server define one client configuration task
```

```
Config1 : aliased Client_Configuration :=
  (Command_Number    => 9,          -- ❶
   Enabled           => True,       -- ❷
   Debug_On          => False,      -- ❸
   Task_Period_MS    => 10,         -- ❹
   Retries           => 3,          -- ❺
   Timeout           => 0.2,        -- ❻

   Server_IP_Address => To_Bounded_String("192.168.0.100"), -- ❼
   -- 127.0.0.1 / 192.168.0.100

   Server_TCP_Port   => 502,                                 -- ❽
   -- 502 Standard / 1502 PLC Simu / 1504 App1Simu
```

❶ **Command_Number** spécifie le nombre de commandes exécutées sur le serveur et dimensionne la table **Commands**.

❷ **Enabled** active ou désactive la scrutation sur le serveur.

❸ **Debug_On** active ou désactive le mode débogage de libmodbus, ce qui active ou non la trace des échanges.

❹ **Task_Period_MS** spécifie la période propre de la tâche cliente.

❺ **Retries** spécifie le nombre d'essais avant de déclarer la commande en défaut.

❻ **Timeout** spécifie le délai de réponse maximum.

❼ **Server_IP_Address** indique l'adresse du serveur auquel le client doit se connecter.

❽ **Server_TCP_Port** indique le port sur lequel le serveur attend les requêtes, 502 est le port standard.

```
   Commands =>
     (--                                Period              Offset Offset
      --               Action Enabled Multiple Shift Number Remote  Local
      1 =>
        (Read_Input_Registers,   True,      10,    0,    10,     0,     0),
      2 =>
        (      Read_Registers,   True,      20,    0,    10,     0,    10),
      3 =>
        (     Write_Registers,   True,      30,    0,    20,    20,     0),
      4 =>
        (           Read_Bits,   True,      30,    1,    16,     0,     0),
      5 =>
        (     Read_Input_Bits,   True,      30,    2,    16,     0,    32),
      6 =>
        (      Write_Register,   True,      30,    3,     1,    50,    30),
      7 =>
        (           Write_Bit,   True,      30,    4,     1,     0,     0),
      8 =>
        (          Write_Bits,   True,      30,    5,    15,     1,     1),

      9 => (Action              => Write_Read_Registers,    -- ❶
            Enabled             => True,
            Period_Multiple     => 10,
            Shift               =>  5,
            Write_Number        => 10,
            Write_Offset_Remote => 40,
            Write_Offset_Local  => 20,
            Read_Number         => 10,
            Read_Offset_Remote  => 10,
            Read_Offset_Local   => 20)
     )
  );

Config2 : aliased Client_Configuration :=                   -- ❷
```

```
      (Command_Number    => 2,
       Enabled           => False,
       Debug_On          => False,
       Task_Period_MS    => 100,
       Retries           => 3,
       Timeout           => 0.2,

       Server_IP_Address => To_Bounded_String("127.0.0.1"),
       Server_TCP_Port   => 1503, -- My own MBTCP server

       Commands =>
         (--                                Period              Offset Offset
          --               Action Enabled Multiple Shift Number Remote  Local
          1 =>
            (      Read_Registers,   True,      10,    0,    10,     0,     0),
          2 =>
            (     Write_Registers,   True,      30,    1,    10,     0,     0)
         )
      );

   -- Declare all clients configuration in the array
   -- The kernel will create those clients accordingly

   MBTCP_Clients_Configuration : Client_Configuration_Access_Array :=
     (1 => Config1'Access,      -- ❸
      2 => Config2'Access);

end A4A.Application.MBTCP_Clients_Config;
```

❶ Pour la commande "Write_Read_Registers" on remarque qu'elle nécessite quelques paramètres supplémentaires par rapport aux autres. Cela illustre d'une part que la structure des paramètres de commande est déterminée par l'identifiant de la commande, et d'autre part qu'il est possible de nommer les champs de la structure.

❷ Cette configuration n'est définie que pour l'exemple.

❸ Pour finir, on remplit le tableau des configurations clientes.

7.6 Serveur Modbus TCP

La spécification Modbus définit un modèle de données qui comporte quatre tables constituées de bits pour les entrées (Discrete Inputs) en lecture seule et sorties / mémentos (Coils) en lecture / écriture, ou de registres, mots de 16 bits, pour les entrées analogiques (Input Registers) en lecture seule et les registres internes (Holding Registers) en lecture / écriture.

Chaque table comporte 65536 éléments.

Il existe deux possibilités, l'une selon laquelle les quatres tables sont disjointes, l'autre les superposant.
L'implémentation dans "Ada for Automation" est du premier type.

TABLE 7.1: Modèle de données

Table	Type	Accès	Description
Input Bits	Bit	L	Entrées TOR
Coils	Bit	L / E	Sorties / Mémentos TOR
Input Registers	Registre	L	Entrées Analogiques
Holding Registers	Registre	L / E	Registres internes / Sorties Analogiques

Le serveur Modbus TCP est géré par une tâche dédiée qui répond aux requêtes des clients indépendamment des tâches traitant

l'automatisme.

Un mécanisme de verrouillage permet de gérer la cohérence des données en bloquant les lectures durant l'exécution d'une écriture.

L'architecture ressemble peu ou prou à celle organisée pour les clients. On y retrouve les DPM et les zones mémoire IO correspondantes.

La tâche "Main_Task" gère la mise à jour des zones mémoire Entrée / Sortie par rapport à la DPM, mais aussi recopie les données E/S dans les données du serveur de façon à permettre la visualisation de ces données brutes.

7.7 Scrutation sur Modbus RTU (IO Scanning)

La conception est très sensiblement identique à celle de "Scrutation sur Modbus TCP (IO Scanning)".

La configuration s'effectue de même et diffère seulement par les paramètres du port série utilisé par le maitre et l'adresse sur le bus de l'esclave dans la table de commandes.

Sur un bus Modbus il ne peut y avoir qu'un seul maitre. Le maitre interroge cycliquement les esclaves comme configuré dans la table.

Au lieu d'avoir une tâche cliente par serveur comme dans le cas Modbus TCP, une seule tâche remplit la fonction de Maitre.

La configuration est réalisée dans le paquetage "A4A.Application.MBRTU_Master_Config" comme ici pour l'application exemple "App3" :

```
with A4A.MBRTU_Master; use A4A.MBRTU_Master;
with A4A.Protocols; use A4A.Protocols.Device_Strings;
with A4A.Protocols.LibModbus; use A4A.Protocols.LibModbus;

package A4A.Application.MBRTU_Master_Config is

   ------------------------------------------------------------------
   -- Modbus RTU Master configuration
   ------------------------------------------------------------------

   Config : aliased Master_Configuration :=
     (Command_Number    => 9,

      Enabled           => True,
      Debug_On          => False,
      Task_Period_MS    => 10,
      Retries           => 3,
      Timeout           => 0.2,

      Device            => To_Bounded_String("/dev/ttyS0"),
      -- "COM1" on Windows
      -- "/dev/ttyS0" on Linux

      Baud_Rate         => BR_115200,
      Parity            => Even,
      Data_Bits         => 8,
      Stop_Bits         => 1,

      Commands =>
        (--                                Period                    Offset Offset
         --               Action Enabled Multiple Shift Slave Number Remote  Local
         1 =>
           (Read_Input_Registers,   True,      10,    0,    2,    10,     0,     0),
         2 =>
           (      Read_Registers,   True,      20,    0,    2,    10,     0,    10),
         3 =>
```

```
           (     Write_Registers,   True,       30,     0,     2,     20,     20,      0),
         4 =>
           (           Read_Bits,   True,       30,     1,     2,     16,      0,      0),
         5 =>
           (     Read_Input_Bits,   True,       30,     2,     2,     16,      0,     32),
         6 =>
           (      Write_Register,   True,       30,     3,     2,      1,     50,     30),
         7 =>
           (           Write_Bit,   True,       30,     4,     2,      1,      0,      0),
         8 =>
           (          Write_Bits,   True,       30,     5,     2,     15,      1,      1),

         9 => (Action              => Write_Read_Registers,
               Enabled             => False,
               Period_Multiple     => 10,
               Shift               =>  5,
               Slave               =>  2,
               Write_Number        => 10,
               Write_Offset_Remote => 40,
               Write_Offset_Local  => 20,
               Read_Number         => 10,
               Read_Offset_Remote  => 10,
               Read_Offset_Local   => 20)
       )
     );

end A4A.Application.MBRTU_Master_Config;
```

C'est similaire à la configuration du client Modbus TCP hormis les paramètres du port série et l'adresse de l'esclave en sus dans les commandes.

7.8 Flux de données

Le diagramme suivant montre les flux de données dans l'application exemple 1, détaillée au chapitre consacré.

Les zones mémoires sont définies dans le paquetage "A4A.Memory". La "Dual Port Memory" est allouée automatiquement dans le paquetage "A4A.Kernel" en fonction de la taille nécessaire.

Les flux en noir sont à l'initiative du ou des clients Modbus TCP.

Les flux en vert correspondent à la scrutation des entrées et sorties sur les serveurs Modbus TCP.

En rouge sont représentés les flux gérés par la tâche "Main_Task" du paquetage "A4A.Kernel".

Pour terminer, les flux en bleu sont organisés par l'utilisateur dans les fonctions du paquetage "A4A.User_Functions" "Map_Inputs, Map_HMI_Inputs, Map_Outputs, Map_HMI_Outputs" appelées dans la procédure principale "Main_Cyclic", elle-même appelée par la tâche "Main_Task".

Dans le cas de l'application exemple 2, "app2" la bien nommée, il suffit de remplacer dans ce schéma les clients Modbus TCP par la carte Hilscher cifX.

La DPM de la carte comporte également une mémoire image process avec une zone d'entrée et une zone de sortie.

Dans celui de l'application exemple 3, "app3", il suffit de remplacer les clients Modbus TCP par le maître Modbus RTU.

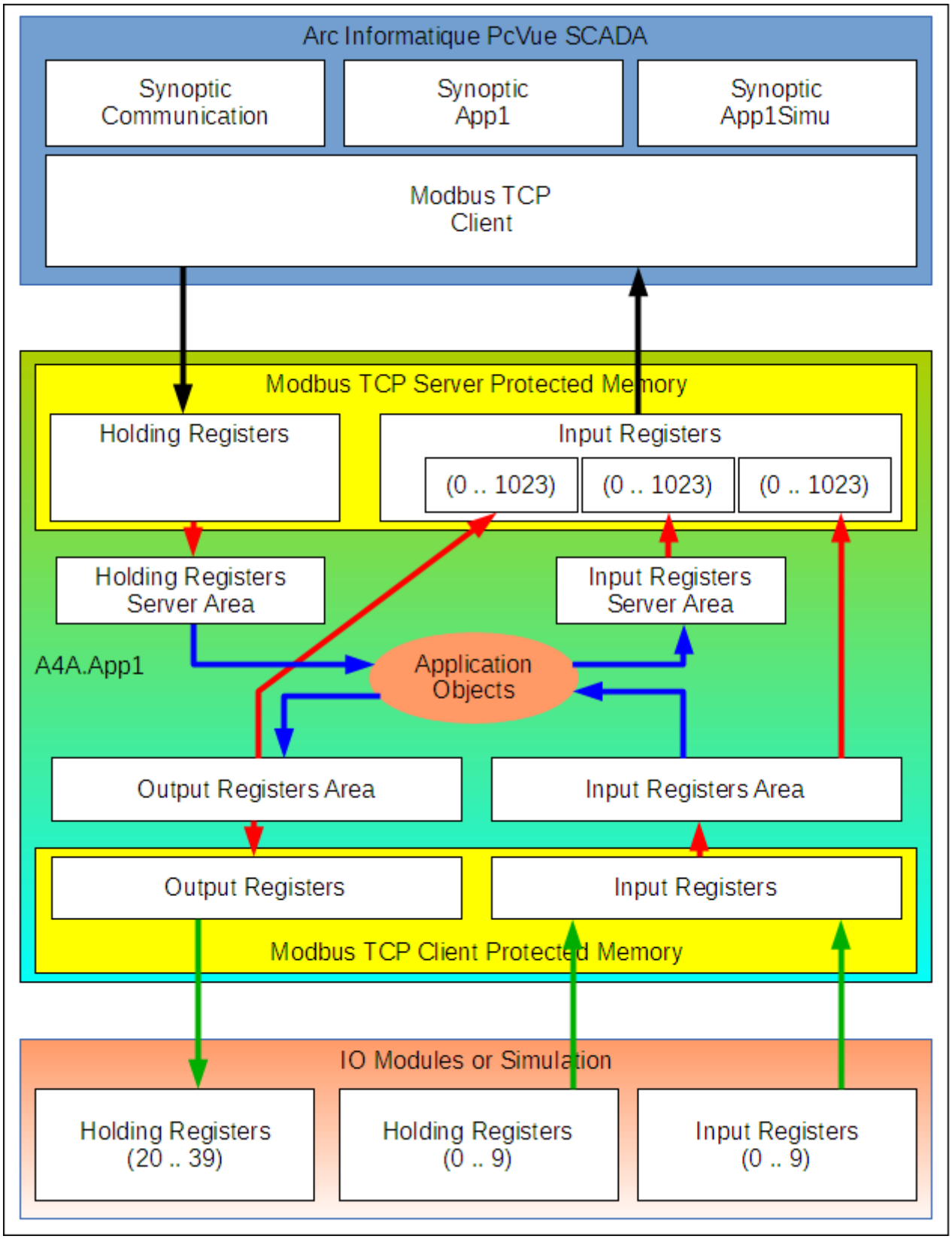

FIGURE 7.19 – A4A Flux de données d'une application nominale

7.9 Interface avec le programme utilisateur

Cette interface définit les interactions entre le framework et le programme utilisateur.

Le framework appelle les fonctions du programme utilisateur définies dans le paquetage "A4A.Application" comme déjà vu dans les paragraphes consacrés aux tâches "Main_Task" et "Periodic 1".

Nous avons déjà vu les procédures "Main_Cyclic" et "Periodic_Run_1".

La procédure "Cold_Start" est également appelée par la tâche "Main_Task" avant de rentrer dans la boucle d'exécution cyclique, lorsque l'initialisation de la tâche est terminée, afin que le programme utilisateur puisse procéder lui-même à quelque initialisation.

Un exemple d'utilisation est fourni par l'application "App2".

La procédure "Closing" est quant à elle appelée par la tâche "Main_Task" lorsque l'application est terminée, par le signal "Ctrl+C" dans le cas de l'application console ou par le bouton "Quit" de l'application GUI, avant que la tâche ne ferme les moyens de communication.

L'application "App2" fournit ici encore un exemple.

Les programmes utilisateur peuvent générer une exception, par exemple une division par 0. Cette exception est normalement gérée et un drapeau "Program_Fault" est alors levé.

La fonction "Program_Fault" permet au framework de connaître l'état de ce drapeau et cela a le même effet que le bouton "Stop".

A ceci près qu'il n'y a pas d'autre moyen pour redémarrer que de relancer l'application.

Chapitre 8

Exemple d'application 1

L'application "App1" met en œuvre uniquement le protocole Modbus TCP pour la communication tant avec les entrées et sorties du système qu'avec l'outil de supervision utilisé pour la démonstration, le produit PcVue® de Arc Informatique.

L'application tourne aussi bien sous Linux que sous Windows®, grâce à libmodbus et à Ada, cependant pour simplifier les choses nous nous en tiendrons à la plateforme majoritaire (pour l'instant), d'autant que le SCADA utilisé pour l'interface homme – machine tourne sous Windows®.

Un article présente également cette application et l'on y voit PcVue en action :
http://slo-ist.fr/ada4automation/a4a-exemple-dapplication-app1

L'exemple d'application 1 est fourni dans le répertoire "app1".

Il montre donc comment mettre en œuvre une application de base avec :

— des clients Modbus TCP,

— un serveur Modbus TCP pour y connecter une supervision par exemple,

— la mise en œuvre des fonctions d'automatisme classiques.

Pour ce qui est de la partie opérative, nous n'en avons pas. Aussi, il y a trois solutions :

— vous disposez de la partie opérative et avez un peu d'argent pour acheter le matériel électrique, les modules d'entrées / sorties, les capteurs et actionneurs...

— vous téléchargez un simulateur d'automate comme Modbus PLC Simulator,

— vous utilisez l'application de simulation de partie opérative développée concomitamment !

Pour notre part, nous préférons l'application de simulation et c'est pourquoi nous l'avons développée également.

L'application de simulation pour l'exemple d'application 1 est fournie dans le répertoire "app1simu".

8.1 Spécifications Fonctionnelles

Voici ci-après les spécifications fonctionnelles de cette application.

8.1.1 Objet

Cette application a pour objet la gestion automatique de l'arrosage du jardin de Beau Papa.

8.1.2 Description

Une cuve stocke les eaux pluviales récupérées du toit de la maison, de l'ordre de 700 à 900 litres par mètre carré et par an dans la Loire.

La cuve peut-être alimentée par le réseau municipal pour pallier un déficit de précipitations, une électrovanne normalement fermée commande le débit.

Le jardin est desservi par un système de tuyaux et une pompe de refoulement.

La cuve est équipée :

— d'un capteur de niveau fournissant une mesure analogique numérisée,

— d'un capteur de niveau TOR haut, dont l'information est utilisée pour couper l'électrovanne d'alimentation en eau, pour ne pas remplir l'étang en contrebas,

— d'un capteur de niveau TOR bas, dont l'information est utilisée pour couper l'alimentation de la pompe pour la protéger d'un fonctionnement à vide.

L'information de niveau analogique est traitée pour fournir les seuils d'automatisme, avec une hystérésis, les seuils étant utilisés pour la gestion des actionneurs :

— XHH : ferme l'électrovanne,

— XH :

— XL : ouvre l'électrovanne,

— XLL : arrête la pompe.

Les informations des capteurs de niveau TOR (SL et SH) sont disposés d'une part dans la chaine de commande des actionneurs (fonction de sécurité) et, d'autre part, remontent à l'automatisme pour la génération d'alarmes correspondantes.

On a : SL < XLL et SH > XHH

La vanne est instrumentée, deux fins de course, ouvert et fermé, sont remontés.

Le contacteur de la pompe dispose d'un retour d'information.

8.1.3 Modes de marche

L'installation possède deux modes de marche, le mode manuel et le mode automatique.

La sélection du mode de marche se fait sur le tableau de commande via un commutateur.

Le mode manuel permet le test de l'installation ou le remplissage de la cuve par anticipation.

8.1.3.1 Mode manuel

La vanne peut être pilotée par l'intermédiaire des commandes disponibles sur l'IHM. Le seuil XHH ferme la vanne.

La pompe peut également être pilotée par l'intermédiaire des commandes disponibles sur l'IHM. Le seuil XLL arrête la pompe.

8.1.3.2 Mode automatique

L'arrosage est démarré selon un horaire et en fonction de capteurs d'humidité. (en version ultérieure car nous ne disposons pas d'assez de variables au niveau IHM en version démo. . .)

Lorsque l'arrosage est en marche, le seuil XL déclenche l'ouverture de la vanne, le seuil XHH la fermeture de celle-ci.

La pompe est mise en service durant l'arrosage tant que le seuil XLL est couvert.

8.1.4 Interface Homme – Machine

L'IHM affiche les données remontées de l'automatisme, état des entrées et sorties, alarmes, courbe de tendance sur le niveau et permet le pilotage des organes en mode manuel.

Il permet également le réglage de l'horaire d'arrosage.

8.1.5 Tableau de commande

On trouve sur ce tableau un commutateur Auto / Manu et un bouton poussoir pour l'acquittement des défauts.

8.1.6 Instrumentation

— Un transmetteur de niveau ou LT (Level Transmitter, LT10)
— Un détecteur de niveau ou LS (Level Switch, LS11) avec un seuil haut SH,
— Un détecteur de niveau ou LS (Level Switch, LS12) avec un seuil bas SL.

8.1.7 Entrées / Sorties

8.1.7.1 Entrées Numériques

— Level Transmitter LT10

8.1.7.2 Entrées Tout ou Rien

— LS11_SH
— LS12_SL
— Pump13_FB, retour contacteur pompe
— V14_ZO, fin de course position ouverte électrovanne
— V14_ZF, fin de course position fermée électrovanne
— Auto, commutateur Auto / Manu
— Manu, commutateur Auto / Manu
— Ack_Fault, bouton poussoir Acquit Défaut

8.1.7.3 Sorties Tout ou Rien

— P13_Coil, bobine contacteur pompe
— V14_Coil, bobine électrovanne

8.1.8 Interface Homme – Machine

8.1.8.1 IHM ⇒ Automate

Les commandes Manu des pompe et vanne.

8.1.8.2 Automate ⇒ IHM

Les états des capteurs et actionneurs.

8.1.9 Simulation

A défaut de partie opérative, une application de simulation sera développée pour les tests, la réception et la formation des utilisateurs. :-)

8.2 Vue d'ensemble

Voici un schéma montrant cette architecture :

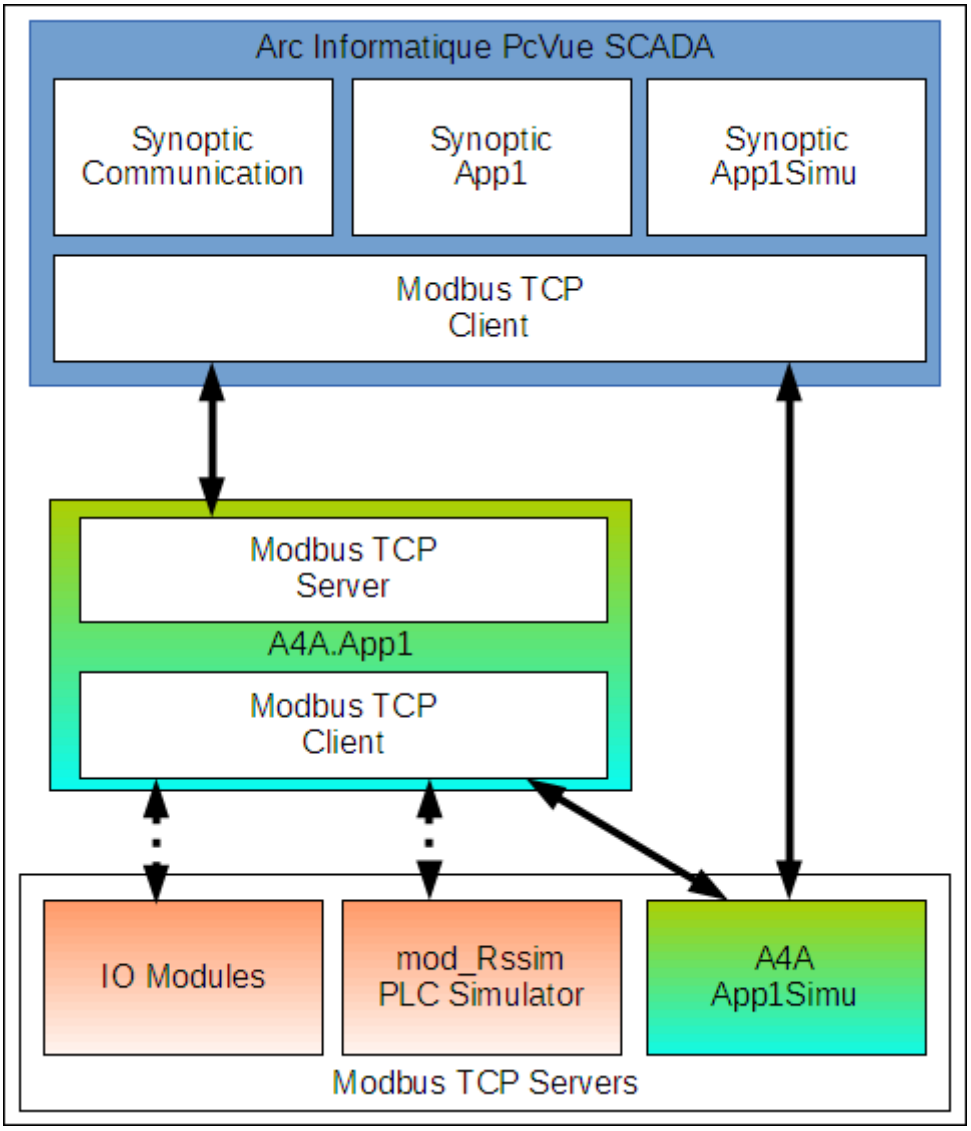

FIGURE 8.1 – app1 Vue d'ensemble

8.3 Spécifications Organiques

Après les spécifications fonctionnelles, ce que le système doit faire, on peut spécifier comment il peut le faire.

Ainsi, l'on a ramené toutes les informations, capteurs et actionneurs dans le coffret électrique et l'on a câblé tout ceci sur un module d'entrées / sorties déporté, serveur Modbus TCP.

8.3.1 Affectation des Entrées / Sorties

Convenons que le module approvisionné permet :

— d'accéder à l'état des entrées TOR câblées en interrogeant les "Discrete Inputs",

— de piloter les sorties câblées en écrivant les "Coils", que l'on peut aussi relire,

— de lire les mesures analogiques en interrogeant les "Input Registers".

TABLE 8.1: Entrées TOR, du point de vue de l'application "App1"

No	Mnémonique	Description
0	Auto	Commutateur en position Auto
1	Manu	Commutateur en position Manu
2	Ack_Faults	BP Acquittement défauts
3	Level_Switch_11	Niveau Haut
4	Valve_14_Pos_Open	Vanne 14, position ouverte
5	Valve_14_Pos_Closed	Vanne 14, position fermée
6	Level_Switch_12	Niveau Bas
7	MyPump13_FeedBack	Pompe 13, retour contacteur

TABLE 8.2: Sorties TOR, du point de vue de l'application "App1"

No	Mnémonique	Description
0	MyPump13_Coil	Pompe 13, bobine contacteur
1	Valve_14_Coil	Vanne 14, solénoïde

TABLE 8.3: Entrées ANA, du point de vue de l'application "App1"

No	Mnémonique	Description
0	Level_Transmitter_10_Measure	Mesure de niveau LT10

8.4 Le projet app1

Comme évoqué un peu plus en amont :

GNAT Pro Studio permet d'organiser les projets en une arborescence de projets avec des projets héritant d'autres. Cela permet d'étendre les projets en ajoutant ou substituant des fichiers de code source. Vous voudrez bien vous référer à la documentation de GPRbuild pour ce qui est des projets.

A4A/app1.gpr Ceci est le fichier projet de l'application exemple 1, il étend le projet principal "a4a.gpr".
Seul le programme utilisateur diffère de celui inclus dans le projet principal.

Ainsi chaque fichier listé dans les sources du répertoire "app1/src" et qui porte le même nom que l'un de ceux qui existent dans les sources du projet principal, dans le répertoire "src", se substitue ou masque le fichier du projet principal.

Alors qu'un fichier n'existant que dans le répertoire "app1/src" étend le projet principal.

Dans ce qui suit, nous nous attachons à montrer ce qui détermine la logique de cette application app1.

8.5 Zones mémoire

La mémoire image des entrées et sorties est définie dans le paquetage "A4A.Memory" dans le répertoire "src".

Si la taille ne vous suffit pas, changez la tout simplement.

```
package A4A.Memory is

   ---------------------------------------------------------------------
   -- IO Areas
   ---------------------------------------------------------------------

end A4A.Memory;

package A4A.Memory.MBTCP_IOServer is

   ---------------------------------------------------------------------
   -- IO Areas
   ---------------------------------------------------------------------

   Bool_IO_Size  : constant := 1024;
   subtype Bool_IO_Range is Integer range 0 .. Bool_IO_Size - 1;

   Bool_Inputs: Bool_Array (Bool_IO_Range) := (others => False);
   Bool_Coils : Bool_Array (Bool_IO_Range) := (others => False);

   Word_IO_Size  : constant := 1024;
   subtype Word_IO_Range is Integer range 0 .. Word_IO_Size - 1;

   Input_Registers : Word_Array (Word_IO_Range) := (others => 0);
   Registers       : Word_Array (Word_IO_Range) := (others => 0);

end A4A.Memory.MBTCP_IOServer;

package A4A.Memory.MBTCP_IOScan is

   ---------------------------------------------------------------------
   -- IO Areas
   ---------------------------------------------------------------------

   Bool_IO_Size  : constant := 1024;
   subtype Bool_IO_Range is Integer range 0 .. Bool_IO_Size - 1;

   Bool_Inputs  : Bool_Array (Bool_IO_Range) := (others => False);
   Bool_Outputs : Bool_Array (Bool_IO_Range) := (others => False);

   Word_IO_Size  : constant := 1024;
   subtype Word_IO_Range is Integer range 0 .. Word_IO_Size - 1;

   Word_Inputs  : Word_Array (Word_IO_Range) := (others => 0);
   Word_Outputs : Word_Array (Word_IO_Range) := (others => 0);

end A4A.Memory.MBTCP_IOScan;
```

8.6 Configuration de la scrutation sur Modbus TCP (IO Scanning)

Une description détaillée de cette scrutation est disponible au chapitre Conception.

La configuration a lieu dans le fichier "a4a-application-mbtcp_clients_config.ads" du répertoire "app1/src".

Pour chaque serveur Modbus TCP, l'on utilise une tâche qui sera créée automatiquement à partir de la configuration saisie ici.

Donc, pour chaque serveur Modbus TCP ou module d'E/S, créer une configuration puis l'insérer dans le tableau des configurations "MBTCP_Clients_Configuration".

Voilà ! C'est fait.

La configuration ci-dessous déclare deux serveurs sur le même PC, celui de simulation et celui de l'application app1 elle-même. Comme ce dernier rebouclage ne sert qu'à des fins de tests, il est désactivé.

Remarquons également que seules les commandes 1 à 3 sont utiles dans cette application, les autres sont là pour la démonstration et le test.

```
with A4A.MBTCP_Client; use A4A.MBTCP_Client;
with A4A.Protocols; use A4A.Protocols.IP_Address_Strings;
with A4A.Protocols.LibModbus; use A4A.Protocols.LibModbus;

package A4A.Application.MBTCP_Clients_Config is

   ------------------------------------------------------------------------
   -- Modbus TCP Clients configuration
   ------------------------------------------------------------------------

   -- For each Modbus TCP Server define one client configuration task

   Config1 : aliased Client_Configuration :=
     (Command_Number    => 11,
      Enabled           => True,
      Debug_On          => False,
      Task_Period_MS    => 10,
      Retries           => 3,
      Timeout           => 0.2,

      Server_IP_Address => To_Bounded_String("192.168.0.100"),
      -- 127.0.0.1 / 192.168.0.100

      Server_TCP_Port   => 502,
      -- 502 Standard / 1502 PLC Simu / 1504 App1Simu

      Commands =>
        (--                                  Period              Offset Offset
         --               Action Enabled Multiple Shift Number Remote  Local
         1 =>
           (Read_Input_Registers,   True,      10,    0,    10,     0,     0),
         2 =>
           (      Read_Registers,   True,      20,    0,    10,     0,    10),
         3 =>
           (     Write_Registers,   True,      30,    0,    20,    20,     0),
         4 =>
           (           Read_Bits,   True,      30,    1,    16,     0,     0),
         5 =>
           (     Read_Input_Bits,   True,      30,    2,    16,     0,    32),
         6 =>
           (      Write_Register,   True,      30,    3,     1,    50,    30),
         7 =>
           (           Write_Bit,   True,      30,    4,     1,     0,     0),
         8 =>
           (          Write_Bits,   True,      30,    5,    15,     1,     1),

         9 => (Action              => Write_Read_Registers,
               Enabled             => True,
               Period_Multiple     => 10,
               Shift               =>  5,
               Write_Number        => 10,
               Write_Offset_Remote => 40,
               Write_Offset_Local  => 20,
               Read_Number         => 10,
               Read_Offset_Remote  => 10,
```

```
               Read_Offset_Local   => 20),
         10 =>
           (      Read_Registers,   True,      50,    0,    10,   100,   100),
         11 =>
           (      Read_Registers,   True,      50,    1,    10,   110,   110)
        )
     );

   Config2 : aliased Client_Configuration :=
     (Command_Number    => 2,
      Enabled           => False,
      Debug_On          => False,
      Task_Period_MS    => 100,
      Retries           => 3,
      Timeout           => 0.2,

      Server_IP_Address => To_Bounded_String("127.0.0.1"),
      Server_TCP_Port   => 1503, -- My own MBTCP server

      Commands =>
        (--                                Period              Offset Offset
         --               Action Enabled Multiple Shift Number Remote  Local
         1 =>
           (      Read_Registers,   True,      10,    0,    10,     0,     0),
         2 =>
           (     Write_Registers,   True,      30,    1,    10,     0,     0)
        )
     );

   -- Declare all clients configuration in the array
   -- The kernel will create those clients accordingly

   MBTCP_Clients_Configuration : Client_Configuration_Access_Array :=
     (1 => Config1'Access,
      2 => Config2'Access);

end A4A.Application.MBTCP_Clients_Config;
```

Dans les cuisines de "Ada for Automation", les tâches clients Modbus TCP sont créées.

Elles communiquent avec le reste du système au travers d'une émulation de mémoire double accès qui permet d'assurer la cohérence des données.

Les données sont échangées avec la mémoire image des entrées et sorties, soit deux tableaux de booléens et deux de registres.

Les curieux peuvent aller voir le paquetage "A4A.Kernel" dans le répertoire "src".

8.7 Configuration du Serveur Modbus TCP

Cette configuration a lieu dans le fichier "a4a-application-mbtcp_server_config.ads" du dossier "app1/src".

On instancie le paquetage générique "A4A.MBTCP_Server" en lui fournissant les quantités d'items souhaitées et on lui affecte une configuration.

```
with A4A.MBTCP_Server;
with A4A.Protocols; use A4A.Protocols.IP_Address_Strings;

package A4A.Application.MBTCP_Server_Config is

   ------------------------------------------------------------------------
   -- Modbus TCP Server configuration
```

```
   ------------------------------------------------------------------

   package Server is new A4A.MBTCP_Server
     (
      Coils_Number           => 65536,
      Input_Bits_Number      => 65536,
      Input_Registers_Number => 65536,
      Registers_Number       => 65536
     );

   Config1 : aliased Server.Server_Configuration :=
     (Server_Enabled           => True,
      Debug_On                 => False,
      Retries                  => 3,
      Server_IP_Address        => To_Bounded_String("127.0.0.1"),
      Server_TCP_Port          => 1502); -- 1503

end A4A.Application.MBTCP_Server_Config;
```

On a bien sûr défini les zones d'échange correspondantes, cf. ci-dessus le paquetage "A4A.Memory".

8.8 Objets utilisateur

Les entrées et sorties sont donc connectées sur la mémoire image des entrées et sorties.

Avec un automate classique, vous définiriez une table de variables pour jouer avec vos capteurs et actionneurs durant la phase de synchronisation par exemple.

Un tel concept n'existe pas pour le moment dans "Ada for Automation". Ce n'est pas très grave, vous pouvez utiliser le serveur Modbus TCP qui dispose des données d'E/S. Il ne vous reste plus qu'à créer le synoptique de vos équipements ou une simple table...

Vous disposez également du débogueur de GNAT Pro Studio (GDB).

L'étape suivante est la définition des objets de votre application.

Dans cette application exemple, cela est réalisé dans le paquetage "A4A.User_Objects".

Ce n'est en aucun cas une obligation et vous pouvez structurer votre application comme bon vous semble.

Le code suivant déclare donc des entrées, des sorties, des variables internes, des instances d'objets.

C'est somme toute équivalent à ce que vous feriez dans un programme automate, sans les boites de saisie.

Plus que des fichiers texte, certains lecteurs sont déjà partis ?

```
with A4A.Library.Timers.TON; use A4A.Library.Timers;
with A4A.Library.Devices.Contactor; use A4A.Library.Devices;
with A4A.Library.Devices.Alarm_Switch; use A4A.Library.Devices;
with A4A.Library.Devices.Valve; use A4A.Library.Devices;
with A4A.Library.Analog.PID; use A4A.Library.Analog;
with A4A.Library.Analog.Threshold; use A4A.Library.Analog;

package A4A.User_Objects is

   ------------------------------------------------------------------
   -- User Objects creation
   ------------------------------------------------------------------

   ------------------------------------------------------------------
   -- Inputs
   ------------------------------------------------------------------
```

```
   Auto                : Boolean := False;
   Manu                : Boolean := False;

   Ack_Faults          : Boolean := False;
   -- Faults Acknowledgement

   Level_Transmitter_10_Measure : Word    := 0;

   Level_Switch_11     : Boolean := False;
   Level_Switch_12     : Boolean := False;
   MyPump13_FeedBack   : Boolean := False;
   Valve_14_Pos_Open   : Boolean := False;
   Valve_14_Pos_Closed : Boolean := False;

   ----------------------------------------------------------------------
   -- Outputs
   ----------------------------------------------------------------------
   MyPump13_Coil       : Boolean := False;
   Valve_14_Coil       : Boolean := False;

   ----------------------------------------------------------------------
   -- HMI Inputs
   ----------------------------------------------------------------------

   MyPump13_Manu_Cmd_On     : Boolean := False;
   Valve_14_Manu_Cmd_Open   : Boolean := False;

   ----------------------------------------------------------------------
   -- Internal
   ----------------------------------------------------------------------

   Tempo_TON_1      : TON.Instance;
   -- My Tempo TON 1

   Tempo_TON_2      : TON.Instance;
   -- My Tempo TON 2

   TON_2_Q          : Boolean := False;

   Mode_Auto        : Boolean := False;
   -- Working Mode is Auto

   Mode_Manu        : Boolean := False;
   -- Working Mode is Manu

   Level_Transmitter_10_Value : Float    :=  0.0;
   Level_Transmitter_10_Hyst  : Float    :=  5.0;
   Level_Transmitter_10_HHH_T : Float    := 99.0;
   Level_Transmitter_10_HH_T  : Float    := 90.0;
   Level_Transmitter_10_H_T   : Float    := 85.0;
   Level_Transmitter_10_L_T   : Float    := 15.0;
   Level_Transmitter_10_LL_T  : Float    := 10.0;
   Level_Transmitter_10_LLL_T : Float    :=  1.0;

   Level_Transmitter_10_InitDone : Boolean  := False;

   Level_Transmitter_10_XHHH : Boolean  := False;
   Level_Transmitter_10_XHH  : Boolean  := False;
   Level_Transmitter_10_XH   : Boolean  := False;
   Level_Transmitter_10_XL   : Boolean  := False;
   Level_Transmitter_10_XLL  : Boolean  := False;
   Level_Transmitter_10_XLLL : Boolean  := False;
```

```
   Valve_14_Condition_Perm : Boolean := False;
   Valve_14_Condition_Auto : Boolean := False;
   Valve_14_Condition_Manu : Boolean := False;

   Valve_14_Auto_Cmd_Open  : Boolean := False;

   Valve_14_Cmd_Open: Boolean := False;
   -- Valve Command

   MyPump13_Condition_Perm   : Boolean := False;
   MyPump13_Condition_Auto   : Boolean := False;
   MyPump13_Condition_Manu   : Boolean := False;

   MyPump13_Auto_Cmd_On  : Boolean := False;

   MyPump13_Cmd_On  : Boolean := False;
   -- Pump Command

   MyPump13_Is_On   : Boolean := False;
   -- Pump Status

   My_PID_1         : PID.Instance;
   -- My PID Controller 1

   Level_Transmitter_10_Thresholds  : Threshold.Instance;
   -- Level_Transmitter_10 Thresholds Box

   ------------------------------------------------------------------
   -- Devices Instances
   ------------------------------------------------------------------

   MyPump13         : Contactor.Instance :=
     Contactor.Create (Id => "Pump13");
   -- Pump Instance

   LS11_AH          : Alarm_Switch.Instance :=
     Alarm_Switch.Create
       (Id         => "LS11",
        TON_Preset => 2000);
   -- Level Alarm Switch Instance

   LS12_AL          : Alarm_Switch.Instance :=
     Alarm_Switch.Create
       (Id         => "LS12",
        TON_Preset => 2000);
   -- Level Alarm Switch Instance

   Valve_14         : Valve.Instance :=
     Valve.Create
       (Id         => "XV14",
        TON_Preset => 5000); -- a slow valve
   -- Valve Instance

end A4A.User_Objects;
```

8.9 Mapping des E/S

Il faut établir la correspondance de vos objets d'entrée et sortie avec la mémoire image.

Ceci est fait à votre initiative dans les fonctions utilisateurs que vous organiserez comme vous l'entendrez.

Par exemple ainsi :

```
with A4A.Memory.MBTCP_IOServer;
with A4A.Memory.MBTCP_IOScan;
use A4A.Memory;

with A4A.Library.Conversion; use A4A.Library.Conversion;

with A4A.User_Objects; use A4A.User_Objects;

package body A4A.User_Functions is

   ----------------------------------------------------------------------
   -- User functions
   ----------------------------------------------------------------------

   procedure Map_Inputs is
--        Temp_Bools : array (0..15) of Boolean := (others => False);
   begin

--        Word_To_Booleans
--          (Word_in       => MBTCP_IOScan.Word_Inputs(0),
--           Boolean_out00 => Auto,
--           Boolean_out01 => Manu,
--           Boolean_out02 => Ack_Faults,
--           Boolean_out03 => Level_Switch_11,
--           Boolean_out04 => Valve_14_Pos_Open,
--           Boolean_out05 => Valve_14_Pos_Closed,
--           Boolean_out06 => Level_Switch_12,
--           Boolean_out07 => MyPump13_FeedBack,
--
--           Boolean_out08 => Temp_Bools(8) , -- Spare
--           Boolean_out09 => Temp_Bools(9) , -- Spare
--           Boolean_out10 => Temp_Bools(10), -- Spare
--           Boolean_out11 => Temp_Bools(11), -- Spare
--           Boolean_out12 => Temp_Bools(12), -- Spare
--           Boolean_out13 => Temp_Bools(13), -- Spare
--           Boolean_out14 => Temp_Bools(14), -- Spare
--           Boolean_out15 => Temp_Bools(15)  -- Spare
--          );

      Auto                := MBTCP_IOScan.Bool_Inputs (32);
      Manu                := MBTCP_IOScan.Bool_Inputs (33);
      Ack_Faults          := MBTCP_IOScan.Bool_Inputs (34);
      Level_Switch_11     := MBTCP_IOScan.Bool_Inputs (35);
      Valve_14_Pos_Open   := MBTCP_IOScan.Bool_Inputs (36);
      Valve_14_Pos_Closed := MBTCP_IOScan.Bool_Inputs (37);
      Level_Switch_12     := MBTCP_IOScan.Bool_Inputs (38);
      MyPump13_FeedBack   := MBTCP_IOScan.Bool_Inputs (39);

      Level_Transmitter_10_Measure := MBTCP_IOScan.Word_Inputs(0);

   end Map_Inputs;

   procedure Map_Outputs is
   begin

--        Booleans_To_Word
--          (Boolean_in00 => MyPump13_Coil,
--           Boolean_in01 => Valve_14_Coil,
--           -- others => Spare
```

```
--           Word_out     => MBTCP_IOScan.Word_Outputs(0)
--          );

      MBTCP_IOScan.Bool_Outputs (0) := MyPump13_Coil;
      MBTCP_IOScan.Bool_Outputs (1) := Valve_14_Coil;

   end Map_Outputs;

   procedure Map_HMI_Inputs is
--        Temp_Bools : array (0..15) of Boolean := (others => False);
   begin

--        Word_To_Booleans
--          (Word_in       => MBTCP_IOServer.Registers(0),
--           Boolean_out00 => MyPump13_Manu_Cmd_On,
--           Boolean_out01 => Valve_14_Manu_Cmd_Open,
--           Boolean_out02 => Temp_Bools(2) , -- Spare
--           Boolean_out03 => Temp_Bools(3) , -- Spare
--           Boolean_out04 => Temp_Bools(4) , -- Spare
--           Boolean_out05 => Temp_Bools(5) , -- Spare
--           Boolean_out06 => Temp_Bools(6) , -- Spare
--           Boolean_out07 => Temp_Bools(7) , -- Spare
--
--           Boolean_out08 => Temp_Bools(8) , -- Spare
--           Boolean_out09 => Temp_Bools(9) , -- Spare
--           Boolean_out10 => Temp_Bools(10), -- Spare
--           Boolean_out11 => Temp_Bools(11), -- Spare
--           Boolean_out12 => Temp_Bools(12), -- Spare
--           Boolean_out13 => Temp_Bools(13), -- Spare
--           Boolean_out14 => Temp_Bools(14), -- Spare
--           Boolean_out15 => Temp_Bools(15)  -- Spare
--          );

      MyPump13_Manu_Cmd_On   := MBTCP_IOServer.Bool_Coils (0);
      Valve_14_Manu_Cmd_Open := MBTCP_IOServer.Bool_Coils (1);

   end Map_HMI_Inputs;

   procedure Map_HMI_Outputs is
   begin

--        Booleans_To_Word
--          (Boolean_in00 => MyPump13.is_On,
--           Boolean_in01 => MyPump13.is_Faulty,
--           Boolean_in02 => Valve_14.is_Open,
--           Boolean_in03 => Valve_14.is_Closed,
--           Boolean_in04 => Valve_14.is_Faulty,
--           Boolean_in05 => LS11_AH.is_On,
--           Boolean_in06 => LS11_AH.is_Faulty,
--           Boolean_in07 => LS12_AL.is_On,
--
--           Boolean_in08 => LS12_AL.is_Faulty,
--           -- others => Spare
--           Word_out     => MBTCP_IOServer.Input_Registers(0)
--          );

      MBTCP_IOServer.Bool_Inputs (0) := MyPump13.is_On;
      MBTCP_IOServer.Bool_Inputs (1) := MyPump13.is_Faulty;
      MBTCP_IOServer.Bool_Inputs (2) := Valve_14.is_Open;
      MBTCP_IOServer.Bool_Inputs (3) := Valve_14.is_Closed;
      MBTCP_IOServer.Bool_Inputs (4) := Valve_14.is_Faulty;
      MBTCP_IOServer.Bool_Inputs (5) := LS11_AH.is_On;
```

```
      MBTCP_IOServer.Bool_Inputs (6) := LS11_AH.is_Faulty;
      MBTCP_IOServer.Bool_Inputs (7) := LS12_AL.is_On;
      MBTCP_IOServer.Bool_Inputs (8) := LS12_AL.is_Faulty;

      MBTCP_IOServer.Input_Registers(0) := Level_Transmitter_10_Measure;

   end Map_HMI_Outputs;

end A4A.User_Functions;
```

8.10 La fonction principale

La boucle principale dans "A4A.Kernel" appelle la procédure "Main_Cyclic" du paquetage "A4A.Application".

C'est assez classique en automatisme, convenons en.

On y trouve les procédures définies au-dessus : "Map_Inputs" et "Map_Outputs".

Ces procédures sans paramètres et sans parenthèses ressemblent fortement aux sous-routines.

```
...
package body A4A.Application is
...

   procedure Main_Cyclic is
      My_Ident : String := "A4A.Application.Main_Cyclic";
      Elapsed_TON_2 : Ada.Real_Time.Time_Span;
--        Bug : Integer := 10;
   begin
--        A4A.Log.Logger.Put (Who  => My_Ident,
--                            What => "Yop ! ********************** "
--                            & Integer'Image(Integer(MBTCP_IOScan_Inputs(0))));

--        Bug := Bug / (Bug - 10);
      Map_Inputs;

      Map_HMI_Inputs;

      -- Working mode
      Mode_Auto := Auto and not Manu;
      Mode_Manu := not Auto and Manu;

      -- Level Transmitter 10
      Level_Transmitter_10_Value := Scale_In
        (X    => Integer(Level_Transmitter_10_Measure),
         Xmin => 0,
         Xmax => 65535,
         Ymin => 0.0,
         Ymax => 100.0);

      if not Level_Transmitter_10_InitDone then
         Level_Transmitter_10_Thresholds.Initialise
           (Hysteresis => Level_Transmitter_10_Hyst,
            HHH_T      => Level_Transmitter_10_HHH_T,
            HH_T       => Level_Transmitter_10_HH_T,
            H_T        => Level_Transmitter_10_H_T,
            L_T        => Level_Transmitter_10_L_T,
            LL_T       => Level_Transmitter_10_LL_T,
            LLL_T      => Level_Transmitter_10_LLL_T);

         Level_Transmitter_10_InitDone := True;
```

```
   end if;

   Level_Transmitter_10_Thresholds.Cyclic
     (Value => Level_Transmitter_10_Value,
      HHH   => Level_Transmitter_10_XHHH,
      HH    => Level_Transmitter_10_XHH,
      H     => Level_Transmitter_10_XH,
      L     => Level_Transmitter_10_XL,
      LL    => Level_Transmitter_10_XLL,
      LLL   => Level_Transmitter_10_XLLL);

   LS11_AH.Cyclic(Alarm_Cond => not Level_Switch_11,
                  Ack        => Ack_Faults,
                  Inhibit    => False);

   LS12_AL.Cyclic(Alarm_Cond => not Level_Switch_12,
                  Ack        => Ack_Faults,
                  Inhibit    => False);

   -- Valve_14 Command
   Valve_14_Condition_Perm := not LS11_AH.is_Faulty;
   Valve_14_Condition_Auto := Mode_Auto;
   Valve_14_Condition_Manu := Mode_Manu;

   if Level_Transmitter_10_XL then
      Valve_14_Auto_Cmd_Open := True;
   elsif Level_Transmitter_10_XHH then
      Valve_14_Auto_Cmd_Open := False;
   end if;

   Valve_14_Cmd_Open := Valve_14_Condition_Perm and
     ((Valve_14_Condition_Manu and Valve_14_Manu_Cmd_Open)
     or (Valve_14_Condition_Auto and Valve_14_Auto_Cmd_Open));

   Valve_14.Cyclic (Pos_Open   => Valve_14_Pos_Open,
                    Pos_Closed => Valve_14_Pos_Closed,
                    Ack        => Ack_Faults,
                    Cmd_Open   => Valve_14_Cmd_Open,
                    Coil       => Valve_14_Coil);

   -- MyPump13 Command
   MyPump13_Condition_Perm := not LS12_AL.is_Faulty;
   MyPump13_Condition_Auto := Mode_Auto;
   MyPump13_Condition_Manu := Mode_Manu;

   if Level_Transmitter_10_XHH then
      MyPump13_Auto_Cmd_On := True;
   elsif Level_Transmitter_10_XLL then
      MyPump13_Auto_Cmd_On := False;
   end if;

   MyPump13_Cmd_On := MyPump13_Condition_Perm and
     ((MyPump13_Condition_Auto and MyPump13_Auto_Cmd_On)
     or (MyPump13_Condition_Manu and MyPump13_Manu_Cmd_On));

   MyPump13.Cyclic(Feed_Back => MyPump13_FeedBack,
                   Ack       => Ack_Faults,
                   Cmd_On    => MyPump13_Cmd_On,
                   Coil      => MyPump13_Coil);

   -- Status use example
   MyPump13_Is_On := MyPump13.is_On;
```

```
--         A4A.Log.Logger.Put (Who  => My_Ident,
--                             What => "Here is MyPump Id : " & MyPump13.Get_Id);
      -- A little test
      Tempo_TON_2.Cyclic (Start   => not TON_2_Q,
                          Preset  => Ada.Real_Time.Milliseconds (10000),
                          Elapsed => Elapsed_TON_2,
                          Q       => TON_2_Q);

      if TON_2_Q then
         A4A.Log.Logger.Put (Who  => My_Ident,
                             What => "Tempo_TON_2 elapsed!");
      end if;

      -- Modbus TCP IO Scanning test
      for Index in 10 .. 19 loop
         A4A.Memory.MBTCP_IOScan.Word_Outputs (Index) := Word (Index);
      end loop;

      for Index in 20 .. 29 loop
         A4A.Memory.MBTCP_IOScan.Word_Outputs (Index) :=
           A4A.Memory.MBTCP_IOScan.Word_Inputs (Index);
      end loop;

      A4A.Memory.MBTCP_IOScan.Word_Outputs (30) :=
        A4A.Memory.MBTCP_IOScan.Word_Outputs (30) + 1;

      A4A.Memory.MBTCP_IOScan.Bool_Outputs (0 .. 15) :=
        A4A.Memory.MBTCP_IOScan.Bool_Inputs (32 .. 47);

      -- Modbus TCP Server test
      A4A.Memory.MBTCP_IOServer.Input_Registers (5 .. 19) :=
        A4A.Memory.MBTCP_IOServer.Registers (5 .. 19);

      Map_Outputs;

      Map_HMI_Outputs;

   exception     -- ❶

      when Error: others =>
         A4A.Log.Logger.Put (Who  => My_Ident,
                             What => Exception_Information(Error));

         Program_Fault_Flag := True;

   end Main_Cyclic;

...
end A4A.Application;
...
```

❶ En cas d'erreur de programmation, par exemple une division par 0, une exception est levée et l'information d'erreur est tracée dans le journal tandis que le drapeau **Program_Fault_Flag** est brandi.

On admettra sans peine que le code présenté ci-dessus se lit avec la même facilité que du "Structured Text".

8.11 Boucle du noyau

La boucle principale qui gère le tout, cf. "task body Main_Task" dans "A4A.Kernel.Main" est présentée ci-dessous en version simplifiée pour illustrer comment le tout s'articule.

Vous pouvez naturellement la modifier à votre convenance, comme le reste du code de ce projet, mais ce n'est absolument pas nécessaire et vous pouvez sans doute créer de nombreuses applications avec ce code.

Vous pouvez y trouver :

— l'initialisation du serveur Modbus TCP,

— l'initialisation des clients Modbus TCP,

— l'organisation des flux de données évoqués ici.

```
...
package body A4A.Kernel.Main is
...

   begin

      Log_Task_Start;

   ------------------------------------------------------------------------
   -- Modbus TCP Server Management
   ------------------------------------------------------------------------

      MBTCP_Server_Task := new Server.Periodic_Task
           (Task_Priority  => System.Default_Priority,
            Configuration  => A4A.Application.MBTCP_Server_Config.Config1'Access
           );

      A4A.Log.Logger.Put (Who  => My_Ident,
                          What => "Modbus TCP Server created...");

   ------------------------------------------------------------------------
   -- Modbus TCP Clients Management
   ------------------------------------------------------------------------

      for Index in MBTCP_Clients_Tasks'Range loop
         MBTCP_Clients_Tasks(Index) := new A4A.MBTCP_Client.Periodic_Task
           (Task_Priority  => System.Default_Priority,
            Configuration  => MBTCP_Clients_Configuration(Index),
            DPM_Access     => My_DPM'Access);
      end loop;

      A4A.Log.Logger.Put (Who  => My_Ident,
                          What => "Modbus TCP Clients created...");

   ------------------------------------------------------------------------
   -- Main loop
   ------------------------------------------------------------------------

      loop
         A4A.Memory.MBTCP_IOScan_Inputs := My_DPM.Inputs.Get_Data
           (Offset => A4A.Memory.MBTCP_IOScan_Inputs'First,
            Number => A4A.Memory.MBTCP_IOScan_Inputs'Length);

         Server.Registers_Read
           (Outputs => MBTCP_IOServer_Registers,
            Offset  => 0);

         A4A.Application.Main_Cyclic;
```

```
         My_DPM.Outputs.Set_Data
           (Data_In => A4A.Memory.MBTCP_IOScan_Outputs,
            Offset  => A4A.Memory.MBTCP_IOScan_Outputs'First);

         Server.Inputs_Registers_Write
           (Inputs => A4A.Memory.MBTCP_IOScan_Inputs,
            Offset => 0);

         Server.Inputs_Registers_Write
           (Inputs => A4A.Memory.MBTCP_IOScan_Outputs,
            Offset => A4A.Memory.MBTCP_IOScan_Inputs'Length);

         Server.Inputs_Registers_Write
           (Inputs => MBTCP_IOServer_Input_Registers,
            Offset => (A4A.Memory.MBTCP_IOScan_Inputs'Length
                       + A4A.Memory.MBTCP_IOScan_Outputs'Length));

         exit when Quit;

         case Application_Main_Task_Type is
         when Cyclic =>
            delay My_Delay;
         when Periodic =>
            Next_Time := Next_Time + My_Period;
            delay until Next_Time;
         end case;
      end loop;

...
end A4A.Kernel.Main;
...
```

Chapitre 9

Hilscher

La société Hilscher GmbH est une société Allemande qui a été créée en 1986 et développe des produits de communication depuis plus de vingt ans.

Hilscher conçoit et produit des solutions de communication industrielle depuis le "System on Chip - SoC" jusqu'à l'équipement complet.

Nous ne présenterons brièvement ici que les produits conçus autour de la nouvelle génération de processeur netX.

9.1 Les composants

9.1.1 La gamme netX

Les SoC Hilscher netX sont donc une famille de "System on Chip", système sur puce, qui intègrent autour d'un processeur ARM un certain nombre de périphériques et notamment un ou plusieurs canaux de communication.

Ces canaux de communication ont la particularité de recevoir leur fonction par micro-code et sont de ce fait bien plus flexibles que s'ils étaient ... comment dire ... figés... comme des ASICs.

Ainsi les netX peuvent être utilisés pour gérer toute la panoplie de protocoles du marché.

Les netX sont conçus par Hilscher SoC, filiale de Hilscher GmbH à Berlin. Hilscher SoC peut concevoir également votre puce sur Cahier Des Charges.

Les netX sont durement éprouvés puisqu'ils sont la base même des produits dérivés Hilscher, utilisés dans quantité d'applications industrielles.

Vous pouvez également utiliser le netX dans vos propres produits, selon les termes d'un contrat à passer avec la société.

9.1.2 Le système d'exploitation Temps Réel rcX

Pour la gamme netX, Hilscher a développé un système d'exploitation temps réel optimisé pour la communication industrielle, et libre de droits.

Ce système d'exploitation intègre les pilotes nécessaires à l'utilisation des différents composants du netX.

Le système rcX peut être utilisé sur toute plateforme matérielle basée sur le netX.

Il est possible notamment de développer sa propre plateforme ou d'utiliser l'un des matériels proposés par Hilscher. Vous pourriez par exemple créer un firmware spécifique pour une passerelle netTAP gérant votre protocole propriétaire.

9.1.3 Les couches d'abstraction matérielle

Les HAL, Hardware Abstraction Layer, définissent l'interface entre les pilotes et le matériel. En fournissant ces couches, Hilscher permet le portage d'autres systèmes d'exploitation sur le netX et le développement de vos propres piles de protocoles.

A ce jour, des BSP (Board Support Package) pour Linux et d'autres OS sont disponibles.

9.1.4 Les piles de protocoles

Hilscher est membre des diverses organisations qui régissent les protocoles de communication industrielle et participe à l'élaboration des normes.

Hilscher a développé un portefeuille très large de protocoles. Qu'ils soient traditionnels comme Profibus, DeviceNet, CANopen, AS-i, CC-Link, ou basés sur Ethernet Temps Réel, tel Profinet, Sercos III, EtherCAT, PowerLink, Ethernet/IP ou Open Modbus TCP, en version Maitre ou en version Esclave, tous les protocoles standard du marché sont disponibles sur le netX.

9.2 Les produits conçus autour du netX

Les besoins en communication sont extrêmement divers et Hilscher a créé toute une gamme de produits couvrant les besoins de l'électronicien, de l'automaticien, de l'informaticien chez l'OEM, l'intégrateur ou l'ensemblier.

On trouve donc des solutions pour l'embarqué pour permettre à des équipements électroniques comme des variateurs, des codeurs, des appareils de mesure, des pupitres opérateur, etc… de communiquer en utilisant des composants éprouvés.

Pour le monde PC, Hilscher propose une gamme de cartes dans tous les formats. Ces cartes sont utilisées pour des bancs de test et de mesure ou des solutions d'automatisme évoluées intégrant un automate conforme à la norme IEC 61131-3, voire IEC 61499.

La gamme passerelle permet quant à elle l'intégration de matériels divers, issus d'une opération de renouvellement ou présentant un intérêt particulier pour l'application, dans une architecture réseau mettant en œuvre un protocole non supporté par ces matériels.

Enfin, l'activité "produits standard" est complétée dans une forte proportion de produits OEM (Original Equipment Manufacturer).

9.3 Les outils de configuration, paramétrage et diagnostic

Pour configurer les piles de protocole, Hilscher a développé SYCON.net, un outil basé sur la technologie FDT/DTM.

Ce logiciel permet de configurer tous les produits de la gamme construite autour du netX.

Un outil plus léger a été développé, le netX Configuration Tool, qui permet de configurer de façon simple les produits en version esclave, la configuration des piles de protocoles esclaves étant notablement réduite par rapport aux versions maitre.

9.4 Les pilotes

Pour les cartes PC, les cifX, Hilscher a développé des pilotes pour les systèmes d'exploitation du marché.

Ces pilotes sont basés sur un ToolKit en C, dont le code source est fourni, qui permet donc de porter le pilote sur un système d'exploitation non supporté par Hilscher.

Quelle que soit la plateforme matérielle, le système d'exploitation choisi ou le protocole mis en œuvre, l'interface est commune. Votre application bénéficie d'une connectivité maximale pour un travail de portage minimal.

Il existe une émulation de ce pilote pour le développement embarqué, ce qui permet de commencer le développement d'un équipement sur base PC, en bénéficiant du confort de cette plateforme, et de porter sans trop d'effort le projet sur l'équipement embarqué.

Au dessus de ces pilotes génériques, des pilotes spécifiques ont été développés pour les principaux automates logiciels du marché.

Note
Hilscher GmbH collabore avec notamment les sociétés 3S, qui développe CoDeSys, KW Software avec ProConOs, IBH Softec qui propose des solutions compatibles Siemens.
Chez Hilscher France, nous avons par exemple développé le pilote pour ISaGRAF sur tous les systèmes d'exploitation supportés et nous avons également porté ISaGRAF sur le netX.
Nous avons également développé le pilote pour l'automate WinAC / RTX de Siemens, pilote qui a depuis été repris par la maison mère.

Chapitre 10

Exemple d'application 2

Cette application met en œuvre une carte Hilscher cifX PROFIBUS DP Maître et un serveur Modbus TCP.

Cf. : http://slo-ist.fr/hilscher/cifx/a4a-exemple-dapplication-app2-hilscher-cifx

Chapitre 11

Exemple d’application 3

Cette application met en œuvre un serveur Modbus TCP et un Maître Modbus RTU.

Cf. : http://slo-ist.fr/ada4automation/a4a-exemple-dapplication-app3-modbus-rtu-maitre

Chapitre 12

La bibliothèque

Nous discutons ici de la bibliothèque de "Ada for Automation" comprenant des types de données élémentaires, des fonctions, des procédures et des objets utiles dans une application d'automatisme.

Une application "Ada for Automation" bénéficie de cet environnement mais également de la pléthore de bibliothèques utilisables en Ada, écrites en Ada ou en C moyennant un binding, comme libmodbus ou GtkAda.

12.1 Types élémentaires

Ada définit un certains nombre de types élémentaires et offre les moyens de créer ses propres types de données. Ce n'est pas le sujet de ce livre.

Dans le paquetage père A4A, l'on trouve ainsi définis quelques types et fonctions élémentaires que l'automaticien peut utiliser dans tous les fichiers descendant de ce père sans avoir à le "wither" ni à le "user".

12.1.1 Types

```
with Interfaces;

with Ada.Text_IO;
with Ada.Characters.Latin_1; use Ada.Characters.Latin_1;

with Ada.Unchecked_Conversion;

package A4A is

   ------------------------------------------------------------------
   -- Elementary types
   ------------------------------------------------------------------

   type Byte is new Interfaces.Unsigned_8;
   --  8-bit unsigned integer

   type Word is new Interfaces.Unsigned_16;
   -- 16-bit unsigned integer

   type DWord is new Interfaces.Unsigned_32;
   -- 32-bit unsigned integer

   type LWord is new Interfaces.Unsigned_64;
   -- 64-bit unsigned integer
```

```
type SInt is new Interfaces.Integer_8;
--  8-bit signed integer

type Int is new Interfaces.Integer_16;
-- 16-bit signed integer

type DInt is new Interfaces.Integer_32;
-- 32-bit signed integer

type LInt is new Interfaces.Integer_64;
-- 64-bit signed integer
```

12.1.2 Décalage et rotation

On y trouve les fonctions élémentaires de décalage et de rotation, vers la droite ou vers la gauche, et pour chacun des types élémentaires ou presque :

```
---------------------------------------------------------------------
-- Elementary functions
---------------------------------------------------------------------

function SHL
  (Value  : Byte;
   Amount : Natural) return Byte renames Shift_Left;
-- <summary>Shift Left Byte</summary>

function SHR
  (Value  : Byte;
   Amount : Natural) return Byte renames Shift_Right;
-- <summary>Shift Right Byte</summary>

function SHL
  (Value  : Word;
   Amount : Natural) return Word renames Shift_Left;
-- <summary>Shift Left Word</summary>

function SHR
  (Value  : Word;
   Amount : Natural) return Word renames Shift_Right;
-- <summary>Shift Right Word</summary>

function SHL
  (Value  : DWord;
   Amount : Natural) return DWord renames Shift_Left;
-- <summary>Shift Left DWord</summary>

function SHR
  (Value  : DWord;
   Amount : Natural) return DWord renames Shift_Right;
-- <summary>Shift Right DWord</summary>

function ROL
  (Value  : Byte;
   Amount : Natural) return Byte renames Rotate_Left;
-- <summary>Rotate Left Byte</summary>

function ROR
  (Value  : Byte;
   Amount : Natural) return Byte renames Rotate_Right;
-- <summary>Rotate Right Byte</summary>
```

```
function ROL
  (Value  : Word;
   Amount : Natural) return Word renames Rotate_Left;
-- <summary>Rotate Left Word</summary>

function ROR
  (Value  : Word;
   Amount : Natural) return Word renames Rotate_Right;
-- <summary>Rotate Right Word</summary>

function ROL
  (Value  : DWord;
   Amount : Natural) return DWord renames Rotate_Left;
-- <summary>Rotate Left DWord</summary>

function ROR
  (Value  : DWord;
   Amount : Natural) return DWord renames Rotate_Right;
-- <summary>Rotate Right DWord</summary>
```

12.1.3 Tableaux d'octets et de mots

Également y figurent les types tableaux non contraints, compatibles avec les appels de fonctions en C :

```
-------------------------------------------------------------------
-- Unconstrained Arrays of Elementary types
-------------------------------------------------------------------

type Byte_Array is array (Integer range <>) of aliased Byte;
for Byte_Array'Component_Size use Byte'Size;
pragma Pack(Byte_Array);
pragma Convention (C, Byte_Array);
type Byte_Array_Access is access all Byte_Array;

type Word_Array is array (Integer range <>) of aliased Word;
for Word_Array'Component_Size use Word'Size;
pragma Pack(Word_Array);
pragma Convention (C, Word_Array);
type Word_Array_Access is access all Word_Array;
```

12.1.4 Text_IO

Afin de faciliter l'usage de la console pour le débogage et la trace, on procure aussi les instances du paquetage générique Text_IO pour chaque type élémentaire :

```
-------------------------------------------------------------------
-- Instanciation of Generic Text_IO package for each Elementary type
-------------------------------------------------------------------

package Byte_Text_IO is
  new Ada.Text_IO.Modular_IO (Byte);

package Word_Text_IO is
  new Ada.Text_IO.Modular_IO (Word);

package DWord_Text_IO is
  new Ada.Text_IO.Modular_IO (DWord);
```

```
package SInt_Text_IO is
  new Ada.Text_IO.Integer_IO (SInt);

package Int_Text_IO is
  new Ada.Text_IO.Integer_IO (Int);

package DInt_Text_IO is
  new Ada.Text_IO.Integer_IO (DInt);
```

12.1.5 Conversions non vérifiées

Ada est un langage très à cheval sur les types et on ne peut pas faire n'importe quoi sans indiquer que telle est réellement notre intention. Lorsque l'on veut convertir un type dans un autre et que cela peut poser un problème, il est nécessaire de le faire explicitement, d'où les fonctions de conversions suivantes, à utiliser avec précaution donc :

```
function SInt_To_Byte is new Ada.Unchecked_Conversion
  (Source => SInt,
   Target => Byte);

function Byte_To_SInt is new Ada.Unchecked_Conversion
  (Source => Byte,
   Target => SInt);

function Int_To_Word is new Ada.Unchecked_Conversion
  (Source => Int,
   Target => Word);

function Word_To_Int is new Ada.Unchecked_Conversion
  (Source => Word,
   Target => Int);

function DInt_To_DWord is new Ada.Unchecked_Conversion
  (Source => DInt,
   Target => DWord);

function DWord_To_DInt is new Ada.Unchecked_Conversion
  (Source => DWord,
   Target => DInt);

function LInt_To_LWord is new Ada.Unchecked_Conversion
  (Source => LInt,
   Target => LWord);

function LWord_To_LInt is new Ada.Unchecked_Conversion
  (Source => LWord,
   Target => LInt);
```

12.2 Conversions

En automatisme comme en informatique industrielle, on passe son temps à rassembler des informations booléennes dans des octets ou des mots, de 16, 32 voire 64 bits parce que c'est plus facile à traiter ou inversement plus facile à transporter, et bien évidemment le contraire.

En Ada, il est possible de créer une structure contenant des informations booléennes selon une représentation bien définie avec ce que l'on appelle les clauses de représentation, et de la convertir dans le contenant ad hoc avec les fonctions de conversions non vérifiées, qu'il aura fallu dériver du paquetage générique idoine.

C'est bien, et utilisé à de multiples occasions dans "Ada for Automation", mais l'auteur ne sait pas si c'est efficace de traiter de telles structures quand les données sont utilisées dans des équations booléennes multiples, et cela parait complexe pour l'automaticien qu'il reste.

"Ada for Automation" offre donc les procédures de conversion suivantes :

```
package A4A.Library.Conversion is
   procedure Bytes_To_Word (LSB_Byte : in Byte;
                            MSB_Byte : in Byte;
                            Word_out : out Word);

   procedure Word_To_Bytes (Word_in  : in Word;
                            LSB_Byte : out Byte;
                            MSB_Byte : out Byte);

   procedure Words_To_DWord (LSW_Word  : in Word;
                             MSW_Word  : in Word;
                             DWord_out : out DWord);

   procedure DWord_To_Words (DWord_in  : in DWord;
                             LSW_Word  : out Word;
                             MSW_Word  : out Word);

   procedure Byte_To_Booleans (Byte_in : in Byte;
                               Boolean_out00 : out Boolean;
                               Boolean_out01 : out Boolean;
                               Boolean_out02 : out Boolean;
                               Boolean_out03 : out Boolean;
                               Boolean_out04 : out Boolean;
                               Boolean_out05 : out Boolean;
                               Boolean_out06 : out Boolean;
                               Boolean_out07 : out Boolean
                              );

   procedure Booleans_To_Byte (
                               Boolean_in00 : in Boolean := false;
                               Boolean_in01 : in Boolean := false;
                               Boolean_in02 : in Boolean := false;
                               Boolean_in03 : in Boolean := false;
                               Boolean_in04 : in Boolean := false;
                               Boolean_in05 : in Boolean := false;
                               Boolean_in06 : in Boolean := false;
                               Boolean_in07 : in Boolean := false;
                               Byte_out : out Byte
                              );

   procedure Word_To_Booleans (Word_in : in Word;...);
   procedure Booleans_To_Word (...);
   procedure DWord_To_Booleans (DWord_in : in DWord;...);
   procedure Booleans_To_DWord (...);

end A4A.Library.Conversion;
```

12.3 Traitement Analogique

La mesure acquise, température, pression, vitesse, etc. est fournie sous forme numérique le plus souvent entière, un nombre de points représentant l'étendue de la mesure, qu'il faut mettre à l'échelle pour obtenir, par exemple, une valeur normalisée dans l'intervalle [-100%, 100%] ou [0, 100%], ou une valeur à l'échelle en unités physiques, [-30, +100°C], [0, 50 Bars], [-10, +300 km/h]…

12.3.1 Scale

On trouve donc des fonctions de mise à l'échelle.

```
package A4A.Library.Analog is

   function Scale_In
     (X    : Integer;
      Xmin : Integer;
      Xmax : Integer;
      Ymin : Float;
      Ymax : Float)
      return Float;
   -- returns the value scaled :
   -- Y = Ymax + ((X - Xmax) * (Ymax - Ymin) / (Xmax - Xmin))

   function Scale_Out
     (X    : Float;
      Xmin : Float;
      Xmax : Float;
      Ymin : Integer;
      Ymax : Integer)
      return Integer;
   -- returns the value scaled :
   -- Y = Ymax + ((X - Xmax) * (Ymax - Ymin) / (Xmax - Xmin))
```

12.3.2 Limits

Il est souvent intéressant de limiter les valeurs que peuvent prendre les grandeurs de commande.

Ainsi par exemple, sur une régulation de chaud-froid, tandis que le régulateur PID peut produire des valeurs normalisées dans l'intervalle [-100%, +100%], l'on enverra à la vanne proportionnelle vapeur un consigne limitée dans l'intervalle [0%, +100%]. Pour le froid, c'est laissé en exercice…

```
   function Limits
     (X    : Float;
      Ymin : Float;
      Ymax : Float)
      return Float;
   -- returns the value limited to the bounds
```

12.3.3 Ramp

De même, il peut être souhaitable de limiter la vitesse de variation de la grandeur de commande qui pourrait sans cela avoir des effets délétères.

On peut aussi avoir besoin de générer une rampe à partir d'un échelon en entrée.

La procédure Ramp, appelée dans une tâche périodique remplit cette fonction. Le gradient doit être calculé en fonction de la période de la tâche.

```
   procedure Ramp
     (Value_In    : in Float;
      Gradient    : in Gradient_Type;
      Value_Out   : in out Float
     );
   -- has to be called in a periodic task
   -- the gradient has to be calculated accordingly
```

12.3.4 PID

Cet objet implémente l'algorithme bien connu, puisque disponible sur Wikipedia, du régulateur PID.

La fonction cyclique doit bien sûr être appelée dans une tâche périodique et la période doit être renseignée.

Le booléen "Initialize" remet à 0 les variables internes.

Instantiation

```
with A4A.Library.Analog.PID; use A4A.Library.Analog;

package A4A.User_Objects is

   My_PID_1          : PID.Instance;
   -- My PID Controller 1
```

Utilisation

```
      My_PID_1.Cyclic
        (Set_Point              => My_PID_1_SP,
         Process_Value          => My_PID_1_PV,
         Kp                     => My_PID_1_Kp,
         Ki                     => My_PID_1_Ki,
         Kd                     => My_PID_1_Kd,
         Initialise             => My_PID_1_Init,
         Period_In_Milliseconds => 100,
         Manipulated_Value      => My_PID_1_MV
        );
```

12.3.5 Thresholds

Cet objet est une boite à seuils, que l'on peut utiliser pour de la commande, de la signalisation, de l'alarme. Elle dispose d'une hystérésis et de six seuils réglables.

Instantiation

```
with A4A.Library.Analog.Threshold; use A4A.Library.Analog;

package A4A.User_Objects is

   My_Thresholds_1   : Threshold.Instance;
   -- My Thresholds Box 1
```

Utilisation La procédure "Initialise" permet d'affecter Hystérésis et seuils.

```
```

12.4 Temporisateurs

Les temporisateurs sont des objets, qu'il faut d'abord instancier avant de pouvoir les utiliser.

12.4.1 TON

Temporisation à la montée.

Instantiation

```
with A4A.Library.Timers.TON; use A4A.Library.Timers;

package A4A.User_Objects is

   Tempo_TON_1      : TON.Instance;
   -- My Tempo TON 1
```

Utilisation

```
with Ada.Real_Time;

      Elapsed_TON_1 : Ada.Real_Time.Time_Span;

      -- Could be simulate
      Tempo_TON_1.Cyclic (Start   => MyPump13Coil,
                          Preset  => Ada.Real_Time.Milliseconds (500),
                          Elapsed => Elapsed_TON_1,
                          Q       => MyPump13FeedBack);
```

12.4.2 TOFF

Temporisation à la retombée.

Instantiation

```
with A4A.Library.Timers.TOFF; use A4A.Library.Timers;

package A4A.User_Objects is

   Tempo_TOFF_1      : TOFF.Instance;
   -- My Tempo TOFF 1
```

Utilisation

```
with Ada.Real_Time;

      Elapsed_TOFF_1 : Ada.Real_Time.Time_Span;

      -- TOFF little test
      Tempo_TOFF_1.Cyclic (Start   => Test_TOFF,
                           Preset  => Ada.Real_Time.Milliseconds (500),
                           Elapsed => Elapsed_TOFF_1,
                           Q       => Test_TOFF_Q);
```

12.4.3 TPULSE

Temporisation d'impulsion.

Instantiation

```
with A4A.Library.Timers.TPULSE; use A4A.Library.Timers;

package A4A.User_Objects is

   Tempo_TPULSE_1      : TPULSE.Instance;
   -- My Tempo TPULSE 1
```

Utilisation

```
with Ada.Real_Time;

      Elapsed_TPULSE_1 : Ada.Real_Time.Time_Span;

      -- TPULSE little test
      Tempo_TPULSE_1.Cyclic (Start   => Test_TPULSE,
                             Preset  => Ada.Real_Time.Milliseconds (500),
                             Elapsed => Elapsed_TOFF_1,
                             Q       => Test_TPULSE_Q);
```

12.5 Composants

Les composants sont des objets, qui héritent de la classe "Device", et qu'il faut d'abord instancier avant de pouvoir les utiliser.

12.5.1 Device

Les composants représentent des entités physiques du système à automatiser. Ce sont les pompes, les vannes, les capteurs divers et variés, les régulateurs, etc.

Lorsque le Process Man ou le mécanicien a bien fait son travail, ces composants sont identifiés par un indicatif de leur fonction et un numéro d'identification.

Le composant "Device" est le père de tous les composants possédant une identification.

Cette identification peut être alors utilisée pour tracer une alarme ou un changement d'état par exemple.

C'est un composant virtuel qui ne peut donc pas être instancié en tant que tel.

12.5.2 Alarm Switch

Ce composant est utilisé pour déclencher une alarme si la condition d'alarme est vérifiée.

— il faut d'une part filtrer les alarmes intempestives, c'est le rôle de la temporisation,

— d'autre part mémoriser l'alarme.

L'entrée "Ack" permet l'acquittement de l'alarme.

L'entrée "Inhibit" permet l'inhibition de l'alarme.

Instantiation

```
with A4A.Library.Devices.Alarm_Switch; use A4A.Library.Devices;

package A4A.User_Objects is

   LS12_AL          : Alarm_Switch.Instance :=
     Alarm_Switch.Create
       (Id         => "LS12",
        TON_Preset => 2000);
   -- Level Alarm Switch Instance
```

Utilisation

```
      LS12_AL.Cyclic(Alarm_Cond => not Level_Switch_12,
                     Ack        => Ack_Faults,
                     Inhibit    => False);

      -- MyPump13 Command
      Condition_Auto := not LS12_AL.is_Faulty and Valve_14.is_Open;
```

12.5.3 Contactor

Ce composant représente un contacteur ou l'organe piloté par ce contacteur.
— une alarme temporisée est élaborée si le retour contacteur ne confirme pas l'état commandé.
L'entrée "Ack" permet l'acquittement de l'alarme.

Instantiation

```
with A4A.Library.Devices.Contactor; use A4A.Library.Devices;

package A4A.User_Objects is

   MyPump13         : Contactor.Instance :=
     Contactor.Create (Id => "Pump13");
   -- Pump Instance
```

Utilisation

```
      -- MyPump13 Command
      Condition_Auto := not LS12_AL.is_Faulty and Valve_14.is_Open;

      Condition_Manu := True;

      MyPump13Cmd_On :=
        (Auto and Condition_Auto)
        or (Manu and Condition_Manu);

      MyPump13.Cyclic(Feed_Back => MyPump13FeedBack,
                      Ack       => Ack_Faults,
                      Cmd_On    => MyPump13Cmd_On,
                      Coil      => MyPump13Coil);

      -- Status
      MyPump13IsOn := MyPump13.is_On;
```

12.5.4 Valve

Ce composant représente une vanne normalement fermée.
— une alarme temporisée est élaborée si les retours de position ne confirment pas l'état commandé.
L'entrée "Ack" permet l'acquittement de l'alarme.

Instantiation

```
with A4A.Library.Devices.Valve; use A4A.Library.Devices;

package A4A.User_Objects is

   Valve_14         : Valve.Instance :=
     Valve.Create
       (Id         => "XV14",
        TON_Preset => 5000); -- a slow valve
   -- Valve Instance
```

Utilisation

```
      Valve_14.Cyclic (Pos_Open   => Valve_14_Pos_Open,
                       Pos_Closed => Valve_14_Pos_Closed,
                       Ack        => Ack_Faults,
                       Cmd_Open   => Valve_14_Cmd_Open,
                       Coil       => Valve_14_Coil);
```

```
   -- MyPump13 Command
   Condition_Auto := not LS12_AL.is_Faulty and Valve_14.is_Open;
```

Chapitre 13

Bibliographie

13.1 Livres

[1] [barnes12] John Barnes. Programming in Ada 2012. Cambridge University Press. ISBN 978-1107424814.

[2] [barnes05] John Barnes. Programming in Ada 2005. Addison-Wesley. ISBN 0-32-134078-7.

[3] [burns-wellings] Alan Burns & Andy Wellings. Concurrent and Real-Time Programming in Ada. Cambridge University Press. ISBN 978-0-521-86697-2.

[4] [pert] McCormick, Singhoff & Hughes. Building Parallel, Embedded, and Real-Time Applications with Ada. Cambridge University Press. ISBN 978-0-521-19716-8.

Chapitre 14

Glossaire

PLC
: Programmable Logic Controller ou Automate Programmable Industriel.

Bus de terrain
: Un moyen de communication entre équipements répartis.

Colophon

Ce livre est composé en texte brut dans le format Asciidoc et traité pour fournir des fichiers HTML ou PDF.