Titre: Title:	Développement du système d'acquisition de données et de contrôle pour les boucles thermiques eau-vapeur et au fréon de l'IGN
Auteur: Author:	Ovidiu-Mihai Popescu
Date:	2009
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Popescu, O.-M. (2009). Développement du système d'acquisition de données et de contrôle pour les boucles thermiques eau-vapeur et au fréon de l'IGN [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/8441/

UNIVERSITÉ DE MONTRÉAL

DÉVELOPPEMENT DU SYSTÈME D'ACQUISITION DE DONNÉES ET DE CONTRÔLE POUR LES BOUCLES THERMIQUES EAU-VAPEUR ET AU FRÉON DE L'IGN

OVIDIU-MIHAI POPESCU
DÉPARTEMENT DE GÉNIE PHYSIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE ÉNERGÉTIQUE)
MAI 2009

Your file Votre référence
ISBN: 978-0-494-53920-0
Our file Notre référence
ISBN: 978-0-494-53920-0

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

DÉVELOPPEMENT DU SYSTÈME D'ACQUISITION DE DONNÉES ET DE CONTRÔLE POUR LES BOUCLES THERMIQUES EAU-VAPEUR ET AU FRÉON DE L'IGN

présenté par : POPESCU Ovidiu-Mihai

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de :

M. KOCLAS Jean, Ph. D., président

M. TEYSSEDOU Alberto, Ph. D., membre et directeur de recherche

M. OLEKHNOVITCH Andrei, Ph. D., membre et codirecteur de recherche

M. ROY Robert, Ph. D., membre

Remerciements

Je désire remercier l'Institut de Génie Nucléaire de l'École Polytechnique de Montréal pour me permettre de mettre en valeur mes capacités dans un domaine si captivant. Je tiens à remercier spécialement les professeurs Alberto Teyssedou et Andrei Olekhnovitch, mes directeurs de recherche, pour leur confiance en moi, leur appui constant dans mon travail et pour leur soutien moral et matériel. Leur passion pour les découvertes scientifiques est une forte inspiration pour moi.

Je voudrais remercier aussi tous les professeurs de l'Institut et tous mes collègues étudiants pour leur aide et assistance. Je tiens à remercier, spécialement, François-Xavier de Cordoüe pour ses commentaires très pertinents.

Je remercie ma famille pour son soutien, compréhension et patience.

Résumé

Un programme informatique pour effectuer l'acquisition de données pour les boucles thermiques de l'Institut de Génie Nucléaire a été développé. Une étude a été effectuée dans le but de rendre l'application facile à adapter, compte tenu qu'elle sera utilisée pour deux boucles (eau-vapeur et Fréon). Des séquences nécessaires pour chaque étape de fonctionnement ont été programmées dans le langage spécifique de l'équipement d'acquisition de données (Tempscan 1000). L'application principale a été programmée en Visual C++.NET. La synchronisation des éléments d'interface avec les flux de données en provenance de l'équipement d'acquisition de données a été assurée par l'utilisation des fils d'exécution multiples. L'application a été testée premièrement pendant l'état d'arrêt de la boucle. Les déterminations ont été comparées avec les mesures obtenues par l'application initiale et les résultats sont semblables. L'essai final de l'application a été effectué pendant des expériences réalisées avec la boucle eau-vapeur. Les résultats ont été considérés comme satisfaisants et l'application est utilisée actuellement par l'équipe du Laboratoire Thermo-Hydraulique de l'IGN.

Une application pour le contrôle et le suivi du fonctionnement de la boucle au Fréon a été réalisée. L'étape initiale a été l'étude du système existant pour déterminer les éléments à réutiliser de l'application initiale. Une distribution des tâches par composantes de l'environnement Paragon a été réalisée. La définition des processus et des fonctions correspondantes pour les serveurs a été introduite. Une nouvelle interface graphique a été conçue. Les spécifications de l'ancienne interface ont été respectées pour permettre le passage facile vers la nouvelle application pour l'operateur de la boucle. Les composantes utilisables de l'application ont été essayées dans des conditions d'arrêt de la boucle au Fréon.

Les deux applications sont utilisées actuellement par des professeurs, personnels et étudiants de l'IGN qui opèrent les boucles thermiques.

Abstract

A Data Acquisition System used in the thermal loops of the Institut de Génie Nucléaire (IGN) was developed. It uses a data acquisition unit (Tempscan 1000) as an intermediary device between measurement instruments and the user. A study was carried out in order to make the application more flexible, given that it will be used for data coming from two different loops (the water-steam and the Freon loops). Sequences for each stage of operation were programmed in the specific language of the data acquisition device. The main application was programmed in Visual C++.NET. Synchronization of the graphic interface elements with data flow from the data acquisition system was assured by multiple execution threads. The application was tested first with a shutted down loop. The results of this measurement were compared with those obtained using the initial application. In all the cases, the agreement was excellent. The final test of the application was performed by running the water-steam loop. The results were satisfactory and the application is currently used by the Laboratoire Thermo-hydraulique team of the IGN.

An application for controlling the Freon loop was also developed. It involved a detailed study of particular application requirements, including the definition of processes and functions for the system servers. A new graphic user interface was produced, while ensuring legacy compatibility. Working components of the application were tested satisfactorily in off operational Freon loop.

Both applications are presently used by IGN professors, personnel and students responsible of operating the thermal loops.

Table des matières

Liste de figures

Liste de tableaux

Liste d'abréviations

AM	Gérant de l'application (« Application Manager »)
CC	Courant continu
CRC	Contrôle de redondance cyclique
CS	Stratégie continue (« Continuous Strategy »)
DM	Gérant des données (« Data Manager »)
EEPROM	Mémoire effaçable électriquement et programmable (« Electrically-Erasable Programmable Read-Only Memory »)
EI	Interface pour ingénieur (« Engineering Interface »)
FCC	Flux de chaleur critique
IGN	Institut de Génie Nucléaire
IIOP	Processeur intelligent d'entrée/sortie (« Intelligent Input/Output Processor »)
OC	Contrôle ouvert (« Open Control »)
OI	Interface opérateur (« Operator Interface »)
PÉ	Pleine échelle
PIO	Processus d'entrée-sortie (« Process Input Output »)
PLC	Automate programmable industriel (« Programmable logic controller »)
SAD	Système d'acquisition des données

SCR	Redresseurs à semi-conducteurs (« Silicon-controlled rectifier »)
SLC	Contrôleur avec cycle unique (« Single loop controller »)

Introduction

Dans certains dispositifs industriels en général et dans la production d'énergie par les réacteurs CANDU en particulier, le transport de l'énergie thermique est assuré par un fluide caloporteur (i.e., de l'eau). Dans ce but, on utilise comme fluide caloporteur principal de l'eau à l'état mono et parfois diphasique. L'écoulement diphasique est beaucoup plus efficace que celui mono phasique au niveau de la quantité d'énergie transportée à cause de l'apport de la chaleur latente. Par contre, son principal désavantage est l'apparition du phénomène d'assèchement dans les canaux de combustible dans le cas d'un flux de la chaleur supérieure à la valeur critique (FCC). Il est donc très important de bien connaître les conditions du développement d'un tel phénomène (i.e., assèchement intermittent ou total).

Pour effectuer des recherches dans ce domaine, on utilise des boucles thermiques qui permettent de contrôler les paramètres de l'écoulement dans une section d'essais. Parmi ces paramètres, on a : la pression à la sortie, la température à l'entrée, le débit de l'écoulement, la puissance chauffante, etc. On peut aussi effectuer des études sur la variation de la température au long de la section d'essais chauffée, les conditions physiques d'apparition du FCC, la perte de pression, etc.

Les boucles thermiques représentent donc des dispositifs complexes qui impliquent, pendant leur construction et opération, la résolution des multiples problèmes. Parmi les problèmes rencontrés, on retrouve le contrôle du fonctionnement de la boucle pour pouvoir, par exemple, régler et maintenir les paramètres de l'écoulement dans la section d'essais pendant les expériences. Un autre problème important est l'acquisition de données lors d'expériences pour pouvoir les analyser et interpréter par la suite.

À l'Institut de Génie Nucléaire (IGN), le Laboratoire Thermo-Hydraulique dispose de deux boucles thermiques pour étudier les écoulements mono et diphasique (i.e., écoulements bouillants) :

- la boucle thermique eau-vapeur (pour des pressions allant jusqu'à 40 bar);
- la boucle thermique au Fréon (pour des pressions allant jusqu'à 25 bar, équivalent au plus de 100 bars en eau).

La boucle eau-vapeur est contrôlée à partir des contrôleurs individuels. Par contre, pour l'acquisition de données, le laboratoire dispose d'une application développée dans les années 1990 qui a été programmée dans le langage Visual Basic 2.0. Cette application ne correspond plus aux exigences minimales requises à cause de son instabilité fréquente de fonctionnement. De plus, certaines interventions pour effectuer des améliorations à ce programme sont empêchées à cause du manque, dans le cas du Visual Basic 2.0, des techniques développées ultérieurement en matière de programmation. Le compilateur VB 2.0 génère des exécutables 16 bits qui ne seront plus supportés dans les prochains systèmes d'exploitation de la famille Windows.

Dans le cas de la deuxième boucle, le contrôle est effectué à partir d'un système géré par ordinateur. À l'origine l'application utilisée, PARAGON, fonctionne sur un ordinateur équipé avec un système d'exploitation MS-DOS. La boucle au Fréon est en train d'être modifiée physiquement. L'application doit tenir compte de ces transformations aussi.
Dans le cadre de ce mémoire de maîtrise, on propose les deux objectifs suivants :

- Développement d'un logiciel pour l'acquisition de données pour les boucles thermiques eau-vapeur et au Fréon dans le langage de programmation Visual C++.Net version 2005. Le logiciel sera plus robuste et plus stable que l'application initiale. Il sera aussi plus flexible pour pouvoir être exécuté sur d'autres systèmes d'exploitation.

- Développement d'un logiciel pour le contrôle de la boucle au Fréon dans la dernière version du programme PARAGON. Cela suivra les changements intervenus dans la structure de la boucle au Fréon.

Le système d'acquisition de données, qui est géré par le logiciel développé en premier, sera soumit à des conditions d'opération extrêmes telles que : température supérieure à 40°C, champs électromagnétiques très intenses, transitoires de courants de l'ordre de 5000 A/ms, etc. L'utilisation du compilateur Visual C++.Net version 2005 assure la génération d'un exécutable 32 bits qui sera compatible avec les systèmes d'exploitation Windows XP, Windows Vista et aussi avec les prochaines systèmes d'exploitation de la même famille.

Nous poursuivrons ce document par une présentation générale, dans le premier chapitre, de la boucle eau-vapeur et de la boucle au Fréon de l'IGN. Dans le deuxième chapitre on présentera les composantes du système d'acquisition de données. Suivra, dans le troisième chapitre, une description des structures de commande utilisées dans la communication avec l'interface Personal488. Dans le quatrième chapitre on présentera le logiciel réalisé pour effectuer l'acquisition des données dans les boucles thermiques de l'IGN. Suivra une description du système de contrôle pour la boucle au Fréon. Dans le sixième chapitre sera décrie l'application Paragon. Le logiciel réalisé pour effectuer le contrôle de la boucle au Fréon sera présenté dans le chapitre 8. Tout cela sera suivi par une conclusion.

Chapitre 1

Les boucles thermiques

Dans le but d'accomplir sa mission de recherche dans le domaine du transfère de chaleur dans des conditions d'écoulement mono- et diphasique le Laboratoire Thermo-Hydraulique de l'IGN dispose de deux boucles thermiques : la boucle thermique eau-vapeur (pour pressions allant jusqu'à 40 bar) et la boucle thermique au Fréon (pour pressions allant jusqu'à 25 bar).

1.1 La boucle thermique eau-vapeur de l'IGN

Pour satisfaire la demande de données stables et de qualité qui sont utilisées pour mieux comprendre et prédire le FCC, une boucle thermique eau-vapeur a été construite à l'IGN en 1986.

1.1.1 Description et mode d'opération de la boucle

La boucle thermique de l'Institut de Génie Nucléaire de Montréal est utilisée pour l'étude des écoulements mono- et diphasique sous des conditions diabatiques. Ses principales caractéristiques sont :

- la puissance thermique : 10-400 kW;
- la pression d'opération : 3-40 bar;
- le débit massique : 0,05-1,7 kg s^{-1};
- le sous-refroidissement à l'entrée de section d'essais : 0-100°C.

La figure 1.1 (Olekhnovitch, 1997) montre un diagramme de la boucle eau-vapeur. Elle consiste en : un ballon de vapeur, une pompe de recirculation, un échangeur de chaleur, deux préchauffeurs et une section d'essais. La chaleur est directement produite dans la section d'essais par effet Joule en utilisant du courant continu (CC). La puissance du CC est contrôlée par le circuit primaire d'un transformateur de puissance avec des redresseurs à semi-conducteurs (« *Silicon-Controlled Rectifier* », *SCR*). Pour pouvoir fournir le voltage CC pour la section d'essais (jusqu'à 100 V) la tension de sortie du transformateur de puissance est redressée par un pont de diodes triphasé. En fonction de la résistance électrique de la section d'essais, l'intensité maximale du courant électrique peut aller jusqu'à 5000 A. Pour réduire les fluctuations de l'intensité du courant qui peut induire du bruit électrique dans les instruments, on a connecté un filtre passe-bas de grande puissance entre les redresseurs et la section d'essais. Ce système assure une réduction du bruit de 40 dB.

Le système condenseur-ballon de vapeur permet de contrôler avec précision la pression à la sortie de la section d'essais. Le degré de sous-refroidissement à l'entrée de la section d'essais est contrôlé par l'intermédiaire de deux préchauffeurs connectés en série qui peuvent produire jusqu'à 75 kW pour le premier et, respectivement, 15 kW pour le second (voir la figure 1.1).

Pour minimiser la dispersion d'oxygène dissolu, l'eau est traitée chimiquement, et sa qualité est vérifiée périodiquement. Étant donné que les températures et les pressions élevées tendent à détruire la qualité de l'eau après chaque essai celle-ci est remplacée.

Le contrôle de la boucle est assuré à partir d'un panneau de contrôle situé dans une chambre spécialement conçue à cet effet et en utilisant différents dispositifs de lecture et de commande situés un peu partout dans la boucle (i.e., capteurs, valves de contrôles, etc.). On utilise aussi une série de contrôleurs qui permet de fixer le débit, la pression dans le condenseur, la température du gicleur, etc.

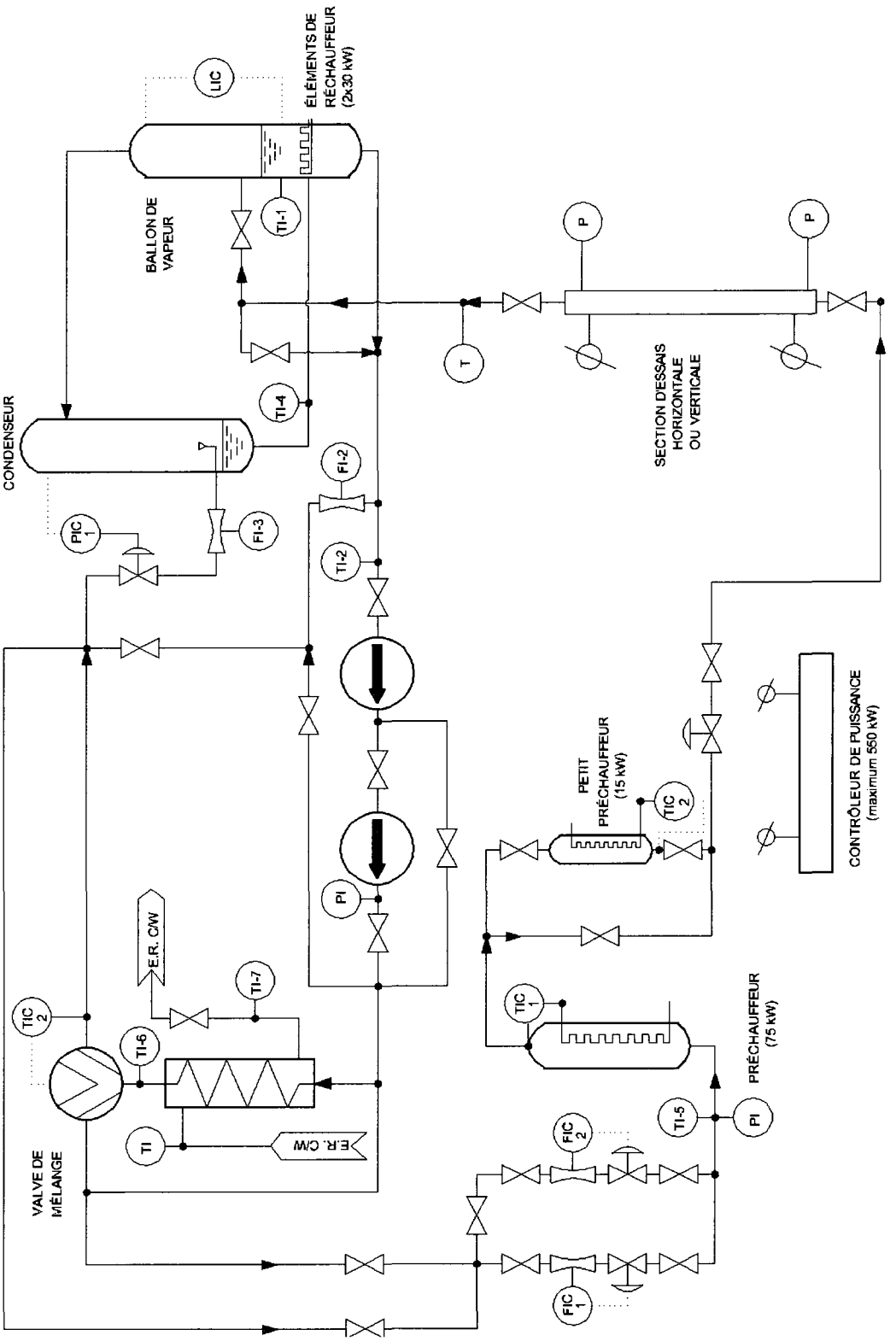

Figure 1.1 : Le schéma de la boucle thermique eau-vapeur.

1.1.2 La section d'essais

Le type de section d'essais peut varier en fonction des expériences qu'on doit effectuer. Pour pouvoir facilement la remplacer, on utilise une conception de connexion hydraulique de type « Swagelok » avec des brides standardisées faites en acier inoxydable. Le potentiel électrique est appliqué à la section d'essais par l'intermédiaire de deux serre-joints en cuivre. En bougeant un des serre-joints on peut varier la longueur chauffée. Pour assurer un contact électrique approprié on utilise des feuilles en argent d'une pureté de 99,99% placées entre les surfaces externes de la section d'essais et les barres de cuivre. La dilatation thermique de la section d'essais est absorbée par un dispositif spécialement conçu pour produire une tension mécanique constante en évitant toute déformation de celle-ci.

Pour assurer l'intégrité de la section d'essais et une bonne reproductibilité des données à la suite des expériences répétées, celle-ci est visuellement vérifiée de manière périodique et, si la moindre déformation est observée, alors elle est remplacée.

1.1.3 L'instrumentation de la boucle thermique eau-vapeur

La boucle thermique eau-vapeur est instrumentée avec les systèmes suivants :

- Des thermocouples pour déterminer la température de la paroi externe et pour la détection du FCC;
- Des capteurs de pression;
- Des systèmes pour la mesure du courant électrique dans les sections d'essais;
- Un système pour la mesure de la différence de potentiel électrique appliquée à la section d'essais;
- Un multiplicateur analogique;
- Une série de débitmètres volumiques;

- Un système d'acquisition de données Tempscan/1000;
- Une extension du système d'acquisition de données Exp/10;
- Un filtre anti-aliasing réalisé dans l'I.G.N. par le professeur Alberto Teyssedou.

Les thermocouples utilisés sont du type K isolés de la Terre. Pour ce type de thermocouple, la jonction est faite de Chromel et Alumel. Ils permettent des mesures dans une large gamme de température, s'étendant de 0 à 1100°C. La figure 1.2 montre l'emplacement des thermocouples dans le cas d'une expérience quelconque.

Les thermocouples sont connectés au Système d'acquisition de données (SAD) en passant le signal par une série d'amplificateurs qui sont aussi utilisés pour obtenir une isolation galvanique entre ce système et la section d'essais. Les signaux sont collectés en temps réel et, par l'intermédiaire de l'application qui fait le sujet du présent mémoire, on peut les enregistrer pour effectuer une analyse ultérieure.

Pour protéger la section d'essais, lorsque le FCC se produit, les signaux provenant des mesures de température de paroi sont simultanément analysés par SAD. Une valeur maximale de température est ainsi utilisée comme seuil. Lorsqu'une température quelconque dépasse cette limite, le SAD envoie un signal vers un système qui déclenche le contrôleur de puissance.

La puissance appliquée dans la section d'essais est simultanément mesurée par deux méthodes. La première consiste en un multiplicateur analogique qui donne la puissance comme le produit de la chute de tension et de l'intensité du courant électrique obtenue par un shunt de 50 mV – 5000 A avec une erreur de 1%. Le résultat du multiplicateur est utilisé comme rétroaction pour le contrôleur de puissance.

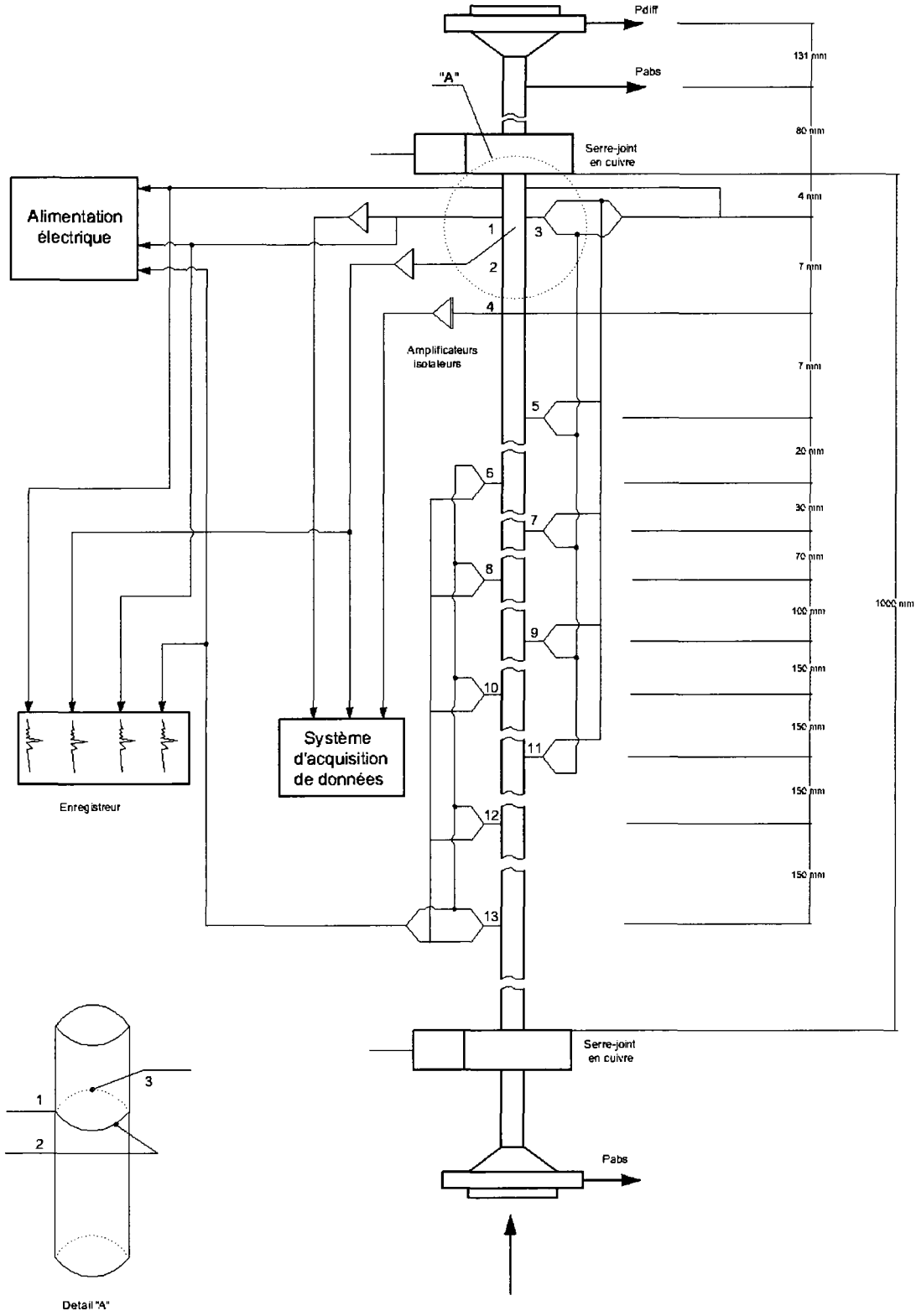

Figure 1.2 : Configuration possible de la section d'essais (Olekhnovitch, 1997).

La deuxième méthode consiste en un prélèvement des données de l'intensité du courant et de la différence de potentiel électrique, puis de la multiplication des valeurs collectées par l'ordinateur.

Parallèlement au suivi de la température de surface d'autres paramètres sont aussi mesurés et sont éventuellement enregistrés. Parmi ces variables se trouve les températures à l'entrée et à la sortie qui sont mesurées avec des thermomètres à résistance qui ont une erreur de mesure de ± 1^0C dans le domaine d'étude. Pour la pression, on utilise trois capteurs produits par la compagnie Sensotec : deux de 5.17 MPa en absolu (avec une erreur de ± 0,1% PÉ) et un de 0.68 MPa différentiel (qui a une erreur de ±0,25% PÉ). Le débit à l'entrée est mesuré avec différents débitmètres produits par la compagnie « Flow Technology » qui ont des erreurs allant jusqu'à 1% de la valeur lue. Les débits sont corrigés par la lecture de la température obtenue à partir d'un thermomètre à résistance situé à la sortie du débitmètre.

1.2 La boucle thermique au Fréon

Cette boucle a été construite en 1995 grâce a une importante subvention obtenue par le Prof. A. Teyssedou. Elle a été conçue pour effectuer des expériences sur le FCC à des pressions équivalentes supérieures à celles qu'on peut obtenir avec la boucle eau-vapeur.

1.2.1 Description de la boucle au Fréon

La boucle thermique au Fréon de l'IGN fait partie d'un projet en développement. Cette boucle a été conçue pour effectuer de la recherche expérimentale dans les domaines suivants :

- L'étude du flux de chaleur critique dans des tubes de 8 mm diamètre intérieur;

- La validation du modèle « fluide à fluide » qui permet l'extrapolation des données obtenues pour le Fréon à l'eau. Même si le modèle a été développé il y a 30 ans, il n'a pas encore été validé pour toutes les conditions. Pour cette raison, on doit utiliser deux boucles, i.e., eau-vapeur et au Fréon;
- L'étude de la perte de charge dans des conditions d'écoulement monophasique et diphasique.

Cette boucle peut opérer avec trois fluides réfrigérants comme suit : Fréon 22, HCFC-123 et HFC-134a. La boucle est conçue pour une pression maximale d'opération de 30 bars à la sortie de la section d'essais et à un débit massique maximal de 4 kg/s. L'utilisation du Fréon permet d'effectuer des expériences à une pression équivalente pour l'eau plus élevée que 100 bars et à une puissance équivalente jusqu'à 1.5 MW. Cette boucle peut être représentée par deux circuits distincts, comme présente dans la figure 1.3 :

- Le circuit d'addition de la chaleur, formée par un préchauffeur, la section d'essais et le système de mesure d'écoulement de sortie;
- Le circuit de rejet de la chaleur qui consiste d'un échangeur de chaleur et d'un condensateur à contact direct.

Les deux branches se fusionnent au niveau du condensateur, et le fluide caloporteur est re-circulé par une pompe à couplage magnétique. L'échangeur de chaleur sert à obtenir le sous-refroidissement nécessaire à l'entrée de la section d'essais et à fournir le refroidissement requis par la douche de liquide réfrigérant utilisé par le condensateur. Le condensateur à douche est un appareil très efficace de transfert thermique; pour cette raison constitue un outil exceptionnel pour contrôler la pression de fonctionnement de la boucle. En commandant le débit de liquide réfrigérant qui passe par la douche, même les perturbations les plus petites de pression du système peuvent être rapidement contrôlées. Ainsi ce système permet de maintenir la pression de la boucle avec une erreur de 0.1 bars pour une gamme variant de 5 à 40 bars.

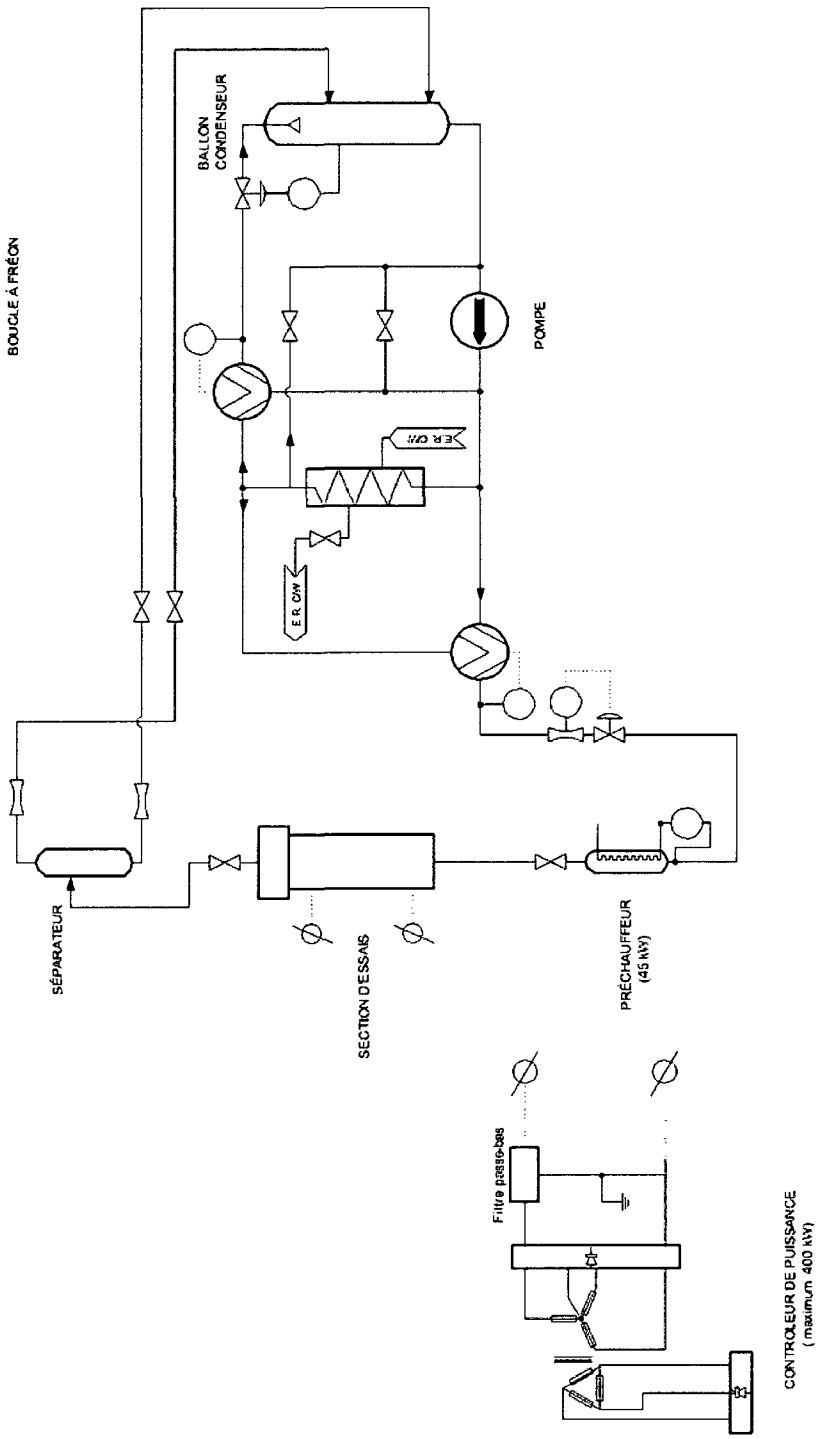

Figure 1.3 : Le schéma de la boucle au Fréon.

1.2.2 Instrumentation de la boucle au Fréon

Les instruments utilisés autant dans la section d'essais que pour garantir un contrôle efficace de cette boucle sont :

- Des thermocouples pour la mesure des différences de température de la surface de la section d'essais;
- Des RTD pour la mesure de la température du fluide dans certains emplacements importants de la boucle (i.e., entrée de la pompe, préchauffeur, écoulement de la douche, etc.);
- Des capteurs de pression absolue et différentielle;
- Des débitmètres volumiques à vortex;
- Des valves de contrôle;
- Un système de transmission de données pour le contrôle et commande.

Les mesures effectuées dans la section d'essais sont collectés par le même système d'acquisition de données que pour la boucle eau-vapeur. Les capteurs de pression et les débitmètres possèdent des sorties dans la gamme de 0 à 5 V, tandis que les dispositifs de contrôle et commande utilisent des entrées en courant dans la gamme de 4 à 20 mA.

Chapitre 2

Le système d'acquisition de données (SAD)

2.1 Les composantes du système

L'appareil utilisé pour l'acquisition de données est un « TempScan/1000 High-Speed Measurement System » produit par la compagnie IOtech. Une extension a été ajoutée pour pouvoir avoir accès à plus de canaux et à un filtre anti-aliasing conçu et réalisé à l'Institut de Génie Nucléaire est utilisé avant la conversion analogique-digitale. La communication entre le SAD et l'ordinateur est assurée par l'intermédiaire d'une carte « Personal488 » produite par la même compagnie.

Le SAD dispose d'entrées et sorties digitales et analogiques. Les sorties sont utilisées pour communiquer avec les dispositifs d'alarme qui déclenche le contrôleur de puissance. Pour pouvoir augmenter le nombre des détecteurs et pour pouvoir les diversifier, on utilise une extension du système d'acquisition de données « Exp/10 » produite par la même compagnie. L'extension permet d'ajouter jusqu'à 64 canaux de lecture supplémentaire. Pour brancher l'instrumentation on utilise des cartes spécialisées du type TempV/32A pour les voltages et TempTc/32B pour les thermocouples. Chaque carte permet le branchement d'un maximum de 32 instruments en mode différentiel ou 64 en mode commune. L'appareil TempScan/1000 possède une mémoire tampon de 4 Mb dans laquelle on peut sauvegarder les données pendant l'acquisition. Cela nous permet de stocker les données pour une durée approximative de 20 minutes (ceci dépend du nombre de canaux). Pour les expériences effectuées dans la boucle thermique de l'IGN, cette période de temps est suffisante.

Comme montré dans la figure 2.1, on communique avec le SAD à partir de l'ordinateur en utilisant une carte PCI qui utilise le standard « IEEE-488 ».

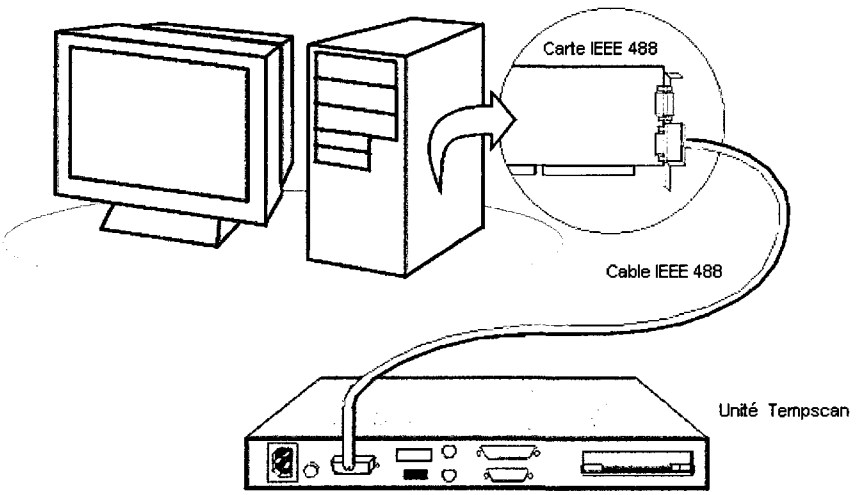

Figure 2.1 : Branchement de l'unité Tempscan à ordinateur par l'intermédiaire de la carte IEEE-488.

Ceci est un standard de communication numérique à petite échelle. Il a été créé pour un usage avec des équipements d'essais automatisés, et il est toujours largement utilisé pour la lecture des instruments de mesure (i.e., multimètres, oscilloscopes, etc.). IEEE-488 est également généralement connu sous le nom de HP-IB (standard de communication d'instrument de Hewlett-Packard) ou de GPIB (standard de communication pour les interfaces). L'IEEE-488 permet d'utiliser jusqu'à 15 dispositifs et de partager une interface électronique parallèle de 8 bits. Le débit maximal d'information digitale est d'environ 8 MB/s.

2.2 Instructions et structures de commande

Le système d'acquisition de données permet de communiquer en utilisant un ensemble de commandes. La structure de la commande dans ce langage respecte les caractéristiques suivantes :

- Les instructions qui produisent des réglages commencent en général par une lettre (l'équivalent d'une fonction) suivie par des paramètres;
- Les instructions qui font la restauration des réglages à la forme préétablie ont la forme *<la lettre correspondant à l'opération à exécuter> (Par exemple *C va effacer les réglages pour les canaux);
- Les instructions qui interrogent l'équipement sur l'état d'un paramètre ont la forme < la lettre correspondant à l'opération à exécuter>? (par exemple A? c'est la commande pour déterminer l'état des alarmes);
- Il y a aussi une instruction qui change l'état d'acquisition (démarre ou arrête l'acquisition), il s'agit du caractère '@'.

L'exécution de la séquence des instructions est faite au moment où l'équipement reçoit l'instruction 'X'. Les paramètres sont séparés par des caractères virgule « , ». On va seulement présenter ici les instructions considérées comme les plus importantes pour ce projet. La première étape après le démarrage de l'équipement est d'effectuer les configurations suivantes:

- La configuration des canaux;
- Le réglage du format des données;
- La configuration des intervalles de temps entre les lectures;
- La configuration de l'acquisition de données dans la mémoire tampon de l'équipement;
- La configuration des alarmes;
- La détermination de l'état du système.

Ces items seront décrits d'une manière détaillée dans les sections suivantes.

I) <u>La configuration des canaux</u>

L'instruction qui gère cette étape a une structure donnée par :

C<canal (aux)>, <type>, [<minimum_du_domaine>, <maximum_du_domaine>, <valeur_écart>]

Si on utilise l'instruction pour un seul canal le premier paramètre sera son numéro. Si on utilise l'instruction pour une séquence de canaux, le même paramètre sera : <le numéro du premier canal> - <numéro du dernier canal>. Pour la variable *type,* on a des valeurs correspondant à une série de 11 thermocouples préétablis, en ordre J, K, T, E, R, S, B, N (14 gauge), N (28 gauge), 100 mV et aussi 100 mV. Pour les dispositifs qui ont à la sortie des différents voltages, on nous propose les échèles suivantes pour la variable *type* :

- 11 pour les voltages variant entre -0.1V à 0.1V;
- 12 pour les voltages variant entre -1V à 1V;
- 13 pour les voltages variant entre -5V à 5V;
- 14 pour les voltages variant entre -10V à 10V;

Les variables *minimum_du_domaine, maximum_du_domaine, valeur_écart* sont utilisées pour définir les alarmes. La variable *valeur_ écart* représente la valeur de la différence entre la limite dépassée du domaine pour laquelle le système sort de l'état d'alarme. Ceci est schématisé à la figure 2.2.

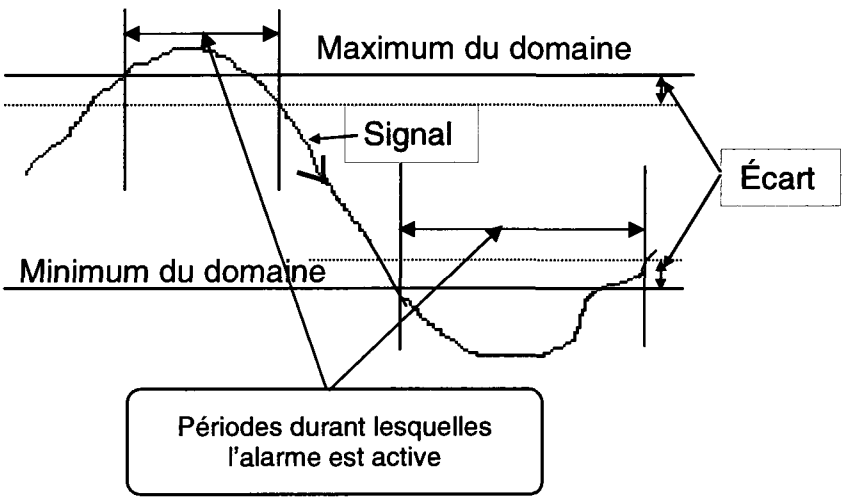

Figure 2.2 : Exemple qui montre le principe de fonctionnement des alarmes.

II) Le réglage du format des données

L'instruction qui permet le réglage pour les données d'entrée et de sortie a la structure suivante :

F<type_des_données>, <format_des_données>

La variable *type_des_données* peut avoir les valeurs suivantes :

- 0 - degrés Celsius (par défaut),
- 1 – degrés Fahrenheit,
- 2 - degrés Rankine,
- 3 - degrés Kelvin,
- 4 - Volts.

La variable *format_des_données* peut prendre les valeurs suivantes :

- 0 – Unités techniques (par défaut),
- 1 – Binaire (du bit le plus grand au plus petit),
- 2 – Binaire (du bit le plus petit au plus grand),
- 3 – Compteur (ASCII).

III) Le réglage des séparateurs

L'instruction qui établit les valeurs des séparateurs a la structure suivante :

Q< séparateur_de_réponse>, < séparateur_de_canal>,
<séparateur_de_lecture>, < séparateur_de_block>, <séparateur>

Les valeurs permises pour les quatre premiers paramètres sont présentées dans le tableau 2.1.

Le paramètre *séparateur_de_réponse* indique le séparateur de réponse pour les réponses de tout usage qui ne demandent pas des données provenant de la mémoire tampon ou des données provenant des registres *minimum_du_domaine/*

maximum_du_domaine/ valeur_écart (MMÉ). Le deuxième argument *séparateur_de_canal* indique le séparateur pour les réponses qui demandent des données provenant des registres *MMÉ.* Le troisième paramètre *séparateur_de_lecture* spécifie le séparateur entre deux ensembles successifs de réponses.

Tableau 2.1 Paramètres pour le réglage des séparateurs

Valeur du paramètre	*Séparateur*
0	Aucun
1	CR LF /EOI
2	CR LF
3	LF CR /EOI
4	LF CR
5	CR /EOI
6	CR
7	LF /EOI
8	LF
9	USER /EOI
10	USER

Le quatrième paramètre *séparateur_de_block* indique le séparateur entre deux blocs successifs de données qui se trouvent dans la mémoire tampon. Le dernier paramètre *séparateur* détermine si un caractère séparateur devrait être utilisé. Les options valides sont : 0 – ne placer aucun séparateur dans les données qui proviennent de la mémoire tampon quand on les lit, et 1 - placer un séparateur qui va avoir la valeur du séparateur réglé par l'usager. Pour régler la valeur de ce séparateur on utilise la commande Set User Terminator (V).

La fonction qui établit la valeur du séparateur au choix de l'usager a la structure suivante :

V<valeur>

Le paramètre **valeur** est le code ASCII du caractère qui sera utilisé comme séparateur.

IV) La configuration des intervalles de temps entre les lectures

Pour réaliser cette fonction on utilise la commande suivante :

I<normal_scan>, <acq_scan>

Les deux paramètres doivent avoir le format *hh:mm:ss.s.* Le terme *normal_scan* spécifie l'intervalle de temps quand l'équipement ne fait pas l'acquisition de données. *L'acq_scan*, qui est important pour nous, spécifie l'intervalle entre deux mesures successives. Si on met comme valeur 00:00:00.0, le système entre dans le mode rapide de mesure qui est de 0.2 secondes pour notre dispositif. L'intervalle maximal qui peut être spécifié est de 24h.

V) La configuration de l'acquisition de données dans la mémoire tampon de l'équipement

La fonction utilisée est :

T<start>, <stop>, <re-arm>, <sync>

Toutes les variables spécifient comment l'événement va être déclenché. Il y a plusieurs options mais pour nous les valeurs importantes sont :

- 0 – événement non spécifié et
- 1 – quand la commande de déclenchement d'acquisition est envoyée.

re-arm détermine si le système d'acquisition va se réarmer après la fin de l'acquisition. La variable *sync* établit si le déclenchement de l'acquisition est synchronisé ou non avec l'acquisition normale.

VI) La configuration des alarmes

La fonction qui gère les alarmes est :

A<canal (aux)>, <no_output>

La variable *canal (aux)* spécifie le canal ou la série de canaux qui seront assignés pour cette alarme. *no_output* spécifie le numéro du canal digital qui sera déclenché s'il y a une alarme. Chaque unité TempScan est prévue avec 32 sorties d'alarme digitale.

VII) La détermination de l'état du système

Cette fonction est, peut-être, la plus ample parmi les instructions du langage. On va seulement présenter l'utilité pour laquelle nous l'avons utilisée, qui est la structure suivante :

U<option>

La variable *option* peut prendre plusieurs valeurs en fonction du paramètre avec lequel on interroge le système. Nous avons utilisé cette commande pour déterminer la valeur de la dernière lecture (*option*=13), pour lire l'état de la mémoire tampon (*option*=6, on en a besoin pour pouvoir extraire le numéro des enregistrements) et pour lire les données contenues dans la mémoire tampon.

VIII) Autres instructions utilisées

Le changement de l'état d'acquisition est réalisé en utilisant la commande '@'. Si le système se trouve dans l'état d'acquisition, il arrête la collection des données sinon il recommence à stocker des données dans la mémoire tampon.

***R** – réalise le redémarrage « à froid » du système. Tous les réglages du système à la suite de l'exécution de cette commande seront les réglages préétablis du système. La commande a besoin de 3-4 secondes pour pouvoir s'accomplir. Au total 5 secondes sont nécessaires pour recommencer à envoyer des commandes vers l'unité TempScan.

***B** – instruction qui vide la mémoire tampon.

***C** – instruction qui efface les réglages pour les canaux.

Chapitre 3

Structures de commande pour l'interface IEEE-488

3.1 La configuration de l'interface IEEE-488

Tel que mentionné à la section 2.1, la carte utilisée pour la communication entre le dispositif d'acquisition de données et l'ordinateur en utilisant le protocole « IEEE488.2 handshake » est la carte « Personal488 ». Cette carte assure un débit maximal de 1MByte/s, elle est du type « Plug & Play » et assure une communication entrée/sortie digitale. La communication entre la carte et l'ordinateur est assurée à partir du pilote « Driver488 » fourni par la compagnie IOtech. Le pilote contient aussi une application nommée « IEEE-488 Configuration Utility », schématisée dans la figure 3.1. Elle permet la gestion des appareils qui sont connectés aux cartes Personal488, elles-mêmes connectées à l'ordinateur. Cette application permet à l'usager de donner des noms génériques aux cartes et aux équipements qui sont branchés, d'établir les séparateurs nécessaires dans la communication, ainsi que d'établir le temps d'attente maximal en cas d'erreur (« Timeout »).

Chaque système branché à cette carte est identifié par un nom générique. Les noms génériques sont très importants parce que la fonction qui permet l'initialisation du dialogue à partir du langage de programmation les prend comme paramètre. Ils doivent respecter les mêmes règles que celles des noms de variable. Ces noms ne doivent pas dépasser la longueur maximale permise qui est de 32 caractères.

Les séparateurs spécifieront le caractère et/ou le signal « End-Of-Identify » qui sera attaché à la fin des données qui seront elles-mêmes envoyées aux dispositifs, ou qui marquent la fin des données reçues de la part des équipements attachés au système

d'acquisition. Le temps maximal d'attente jusqu'à la déclaration de l'état du « Timeout » sera aussi spécifié.

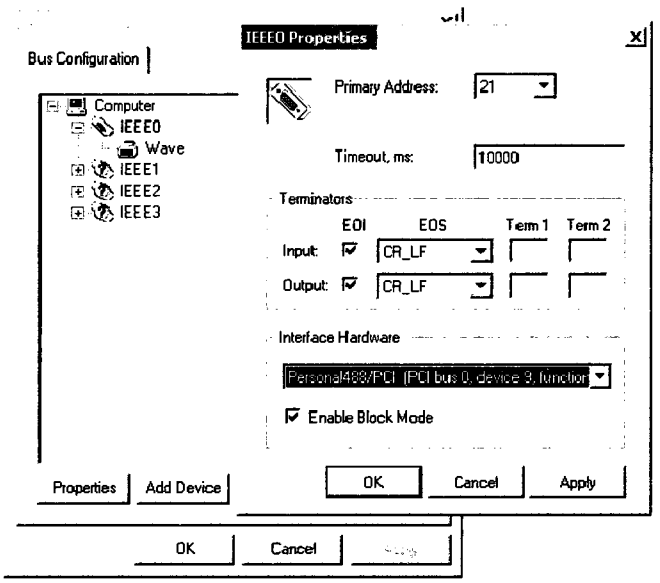

Figure 3.1 : Interface de configuration des cartes IEEE-488 et des équipements branchées.

Le pilote informatique offre aussi une application qui permet d'interroger la carte « Personal488 » en utilisant des fonctions intégrées. Ceci est expliqué dans la section suivante.

3.2 Les fonctions du pilote informatique

Le pilote offre la possibilité de communiquer directement avec la carte « Personal 488 » ou à partir des langages de programmation Microsoft Visual C++, Visual Basic, Borland C++ et Borland Delphi. À vrai dire le pilote informatique est une bibliothèque de fonctions précompilée (nommée « drvr488.dll »). Pour pouvoir accéder à ses fonctions directement il y a aussi un fichier correspondant qui contient les déclarations de chaque fonction.

Nous avons utilisé les fonctions du pilote à partir du langage Microsoft Visual C++. Pour pouvoir faire cela, on a dû inclure dans notre projet les fichiers « header » suivants : « Iotieee.h », « Ioterror.h », « Iotslpib.h » ainsi que le fichier « Iotslpib.h » également inclus dans la librairie du « pilote ». On présentera les fonctions utilisées dans le présent projet seulement, suivant leur apparition dans le programme.

- *int* ***OpenName****(LPSTR name)*

  Cette fonction ouvre la communication avec un dispositif extérieur branché à la carte « Personal488 ». Elle retourne une valeur numérique entière unique qui identifiera l'appareil dans le reste du programme et dont on aura besoin chaque fois que nous l'appelons. L'allocation des valeurs n'est pas aléatoire mais ordonnée à partir du 0. Donc le premier appel de la fonction va retourner 0, le deuxième va retourner 1, etc. *name* est une chaîne de caractères qui représente le nom du dispositif dans l'application de configuration présentée plus tôt.

- *long* ***Output****(int devHandle, LPBYTE data)*

  Cette fonction envoie une chaîne de caractères vers un appareil branché à l'interface par l'intermédiaire du deuxième paramètre. La chaîne de caractères représente une commande. Le résultat de la fonction est du type long et représente un code d'erreur suite à l'exécution de la commande. Si le code de l'erreur est différent de 0, la commande n'a pas été effectuée avec succès. Le premier paramètre représente la valeur entière qui identifie uniquement l'appareil et qui a été retournée par la fonction ***OpenName***;

- *int* ***EnterN****(int devHandle, LPBYTE data, int count)*

  Cette fonction effectue la lecture des données de l'appareil branché à l'interface IEEE-488. Comme pour la fonction précédente, le résultat est de type long et représente un code d'erreur suite à l'exécution de la commande. Si le code de l'erreur est différent de 0, la commande n'a pas été effectuée avec succès. Le premier

paramètre représente la valeur entière qui identifie uniquement l'appareil et qui a été retournée par la fonction ***OpenName***. Le deuxième paramètre représente la chaîne de caractère dans laquelle la fonction mettra les données lues. Le troisième paramètre spécifie le nombre de caractères qui seront lus.

- *long* ***Enter****(int devHandle, LPBYTE data)*

  Cette fonction lit les données provenant de SAD de la même façon que la fonction ***EnterN***, mais la lecture s'arrête à la fin de la séquence actuelle de données. Toutes les autres caractéristiques sont identiques à la fonction antérieure.

- *int* ***UnTalk*** *(int devHandle)*

  Cette fonction annule l'état de transmission de données par l'interface IEEE-488 de l'équipement. Comme pour toutes les autres fonctions le paramètre représente la valeur entière qui identifie uniquement le SAD et qui a été retournée par la fonction ***OpenName***.

- *int* ***Close****(int devHandle)*

  Cette fonction attend que la communication avec le SAD finisse, vide la mémoire tampon et invalide l'identificateur. Le paramètre représente l'identificateur de l'équipement.

Chapitre 4

Le programme d'acquisition de données pour la boucle eau-vapeur et pour la boucle au Fréon

L'objectif de ce projet a été de renouveler et d'améliorer une application développée antérieurement qui gérait l'acquisition de données pour la boucle eau-vapeur d'IGN. L'ancienne application était écrite en Visual Basic 2.0 et la principale erreur produite était l'arrêt de l'exécution et le gel de l'ordinateur au moment où l'on appliquait de la puissance dans la section d'essais.

Pour garantir une meilleure stabilité de l'application, il a été décidé de réécrire ce logiciel a été prise. Le nom du nouvel logiciel est SADLT (Système d'Acquisition de Données – Laboratoire Thermo-hydraulique) version 1.0. Le langage de programmation utilisé est Microsoft Visual C++ .NET, édition 2005. Pour effectuer cette tâche, le type de projet informatique choisi a été « Windows forms application ». De plus, pour assurer le bon fonctionnement des différents processus de calcul, ceux-ci ont été distribués sur trois fils d'exécution. La figure 4.1 représente un schéma simplifié de la philosophie utilisée. Les fils d'exécution constituent ainsi trois parties de l'application qui fonctionnent en parallèle; ils sont les suivants :

1. Le fil principal qui commande l'interface graphique sauf la fenêtre de message (même si cette variable n'existe pas explicitement, on va l'appeler ***mainThread***);
2. Le fil de cycle qui interagit avec le système d'acquisition (nom de variable pointer **cycleThead**);
3. Le fil qui commande la fenêtre de messages (nom de variable pointer **messageThread**). Ce fil d'exécution contrôle une fenêtre qui montre l'avancement du système et son état dans différentes situations (en particulier, quand on

sauvegarde les données à partir de la mémoire tampon sur le disque dur de l'ordinateur).

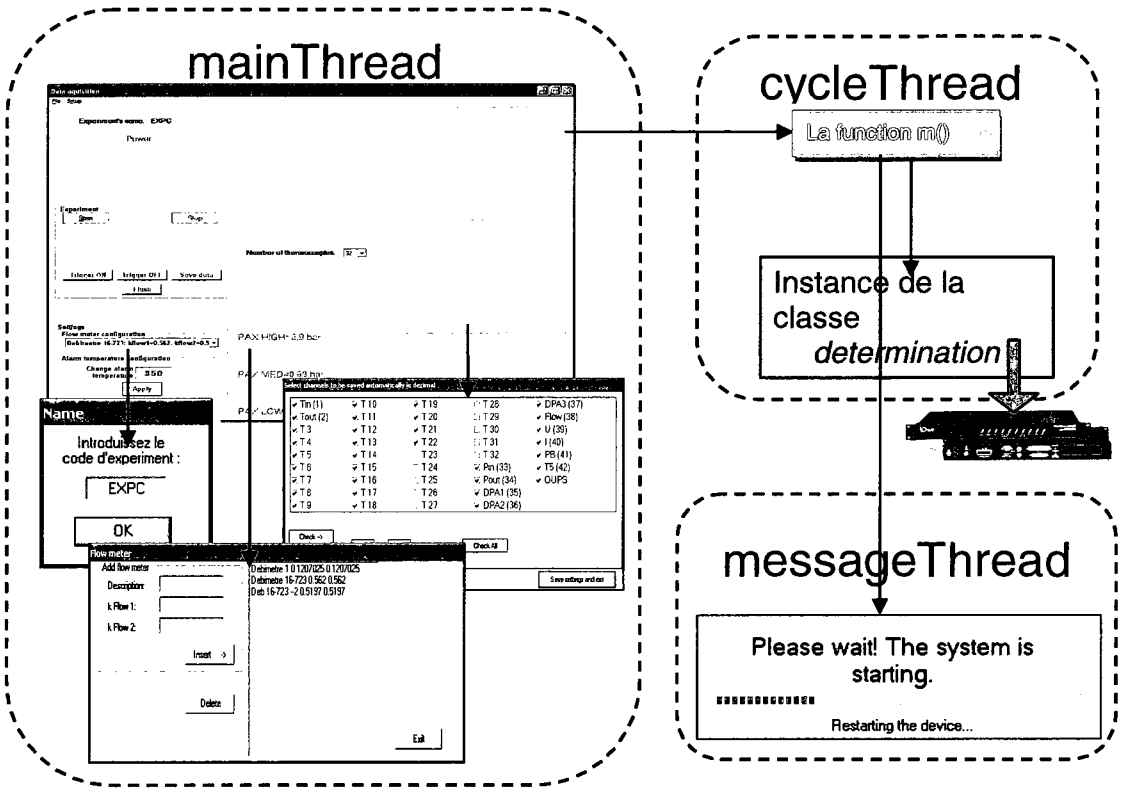

Figure 4.1 : Les fils d'exécution de l'application.

La structure des fichiers « header » et de leurs fils d'exécution correspondants, est la suivante :

- *Form1.h* : implémente une classe qui hérite de la classe System::Windows::Forms::Form. Cette classe implémente la fenêtre principale de l'application. Cette classe s'exécute dans le fil principal d'exécution. À partir de cette classe, on lance la fonction qui contrôle le cycle dans le fil d'exécution **cycleThread**;
- *Name.h* : cette classe implémente aussi la classe System::Windows::Forms::Form et est une fenêtre de dialogue par

l'intermédiaire de laquelle l'usager établit le code de l'expérience (quatre caractères alphanumériques). Cette fenêtre fonctionne également dans le fil principal;

- *Debimeter.h* : implémente aussi la classe System::Windows::Forms::Form. Pendant les expériences sur la boucle, il y a parfois des changements de débitmètre. Cette fenêtre permet à l'usager de gérer les débitmètres utilisés. Elle fonctionne dans le fil principal d'exécution;
- *Wait.h* : fenêtre utilisée pour montrer l'avancement des différentes opérations effectuées (exemple : la proportion dans laquelle un fichier est sauvegardée). Elle fonctionne dans le fil d'exécution **messageThread**;
- *SetupDecimal.h* : fenêtre utilisée pour établir quels seront les paramètres sauvegardés automatiquement dans un fichier texte;
- *Determination.h* : cette classe groupe les instructions envoyées vers l'appareil, en utilisant le format IEEE-488.

Dans les sections suivantes, une analyse détaillée de chaque classe est présentée.

4.1 La classe « Form1.h »

Ce « header » gère la fenêtre principale de l'application par l'intermédiaire de la classe *Form1*. Elle contient les éléments d'interface qui permettent à l'usager d'établir en premier les réglages comme le nom de l'expérience (une séquence de 4 lettres qui sera utilisée dans les noms des fichiers dans lesquels on va sauvegarder les données), le débitmètre qui sera utilisé (avec ses caractéristiques). L'étape suivante est de commencer le suivi des données qui seront affichées dans les éléments de l'interface graphique. Pendant le suivi des expériences, l'usager peut choisir de commencer le stockage des données dans la mémoire tampon du Tempscan/1000 et, après l'arrêt de l'acquisition, de sauvegarder les données sur le disque dur de l'ordinateur ou d'effacer la mémoire tampon. L'usager peut

aussi établir le nombre des canaux de lecture des températures qui seront affichées sur l'écran.

Dans le cadre de la classe *Form1,* on a implémenté la fonction *m* () qui est très importante pour le fonctionnement général de l'application et qui sera aussi présentée en détail plus tard. En principe cette fonction contient un cycle qui effectue principalement le suivi des données et l'affichage dans les étiquettes de la fenêtre principale et, en plus, toutes les opérations qui sont effectuées pendant le suivi des données sont implémentées dans le cadre de ce cycle.

Pour conserver les différentes valeurs des constantes dans l'application, on utilise un fichier de réglage nommé « settings.ini ». Ce fichier se trouve dans la racine du disque en occurrence C :. La figure 4.2 montre le menu de l'application :

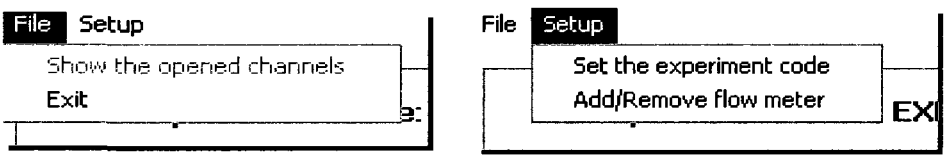

Figure 4.2 : Le menu de la fenêtre principale.

Cette figure montre l'accès aux deux fonctions, *File* et *Setup* qui ont les caractéristiques suivantes :

- *File→Show the opened channels* est utilisé pendant le suivi des données pour montrer les canaux qui n'ont pas de senseurs branchés ou qui ont des senseurs défectueux;
- *File→Exit* est utilisé pour sortir de l'application;
- *Setup→Set the experiment code* est utilisé pour effectuer des réglages sur le code de l'expérience et pour établir aussi quels seront les canaux sauvegardés en format décimal de façon automatique;

- *Setup→Add/remove flow meter* est utilisé pour ouvrir la fenêtre de réglages des débitmètres.

Dans la figure 4.3, on présente un imprimé-écran de la fenêtre implémentée par le fichier *Form1.h.*

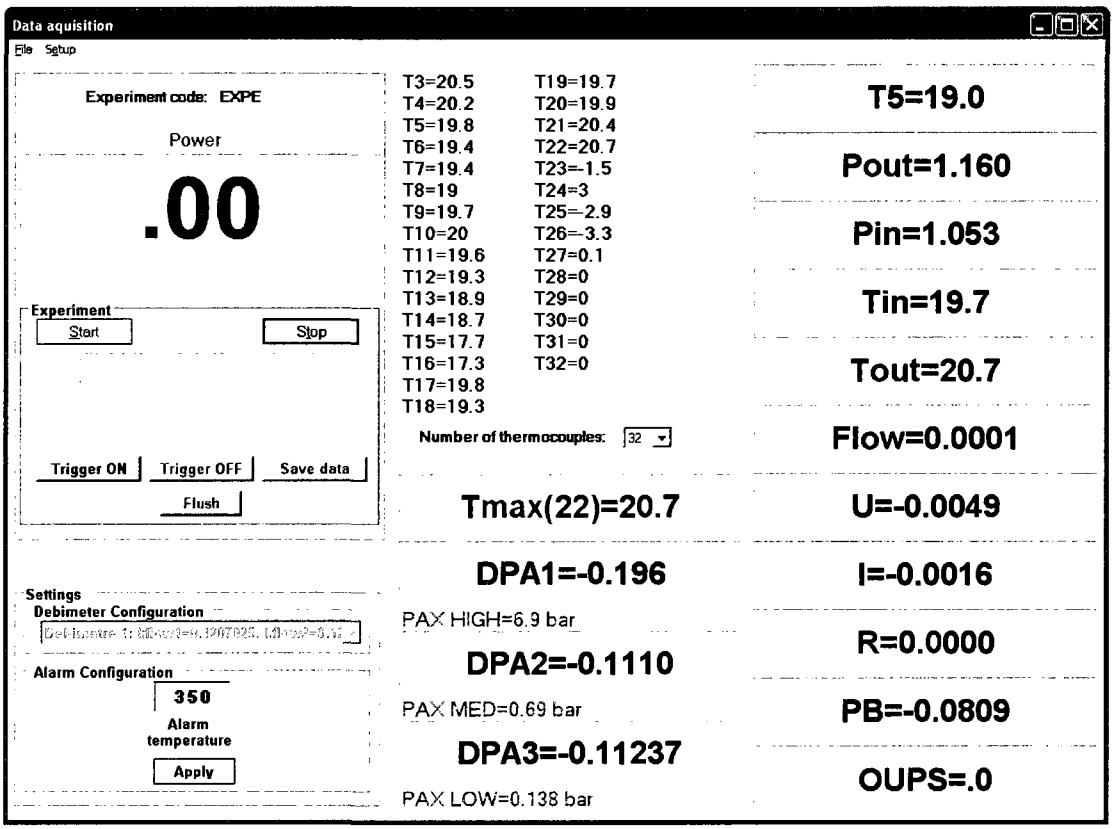

Figure 4.3 : Fenêtre principale de l'application implémentée par la classe *Form1*

En conformité avec cette figure, les objets d'interface graphique sont distribués sur trois colonnes. Les objets d'interface seront identifiés et le rôle de chacun sera présenté en partant de la première colonne.

Comme le montre la figure 4.4, les variables pour la première colonne sont :

- **lCode** : étiquette qui affiche le nom de l'expérience;
- **labelPower** : étiquette qui affiche la puissance consommée;
- **commence** : bouton qui démarre l'équipement Tempscan/100, effectue tous les réglages requis et commence le cycle qui fait la suivie des données;
- **fin** : bouton qui arrête le cycle et redémarre l'équipement;
- **labelTime** : étiquette qui affiche le temps pendant l'acquisition de données dans la mémoire tampon;

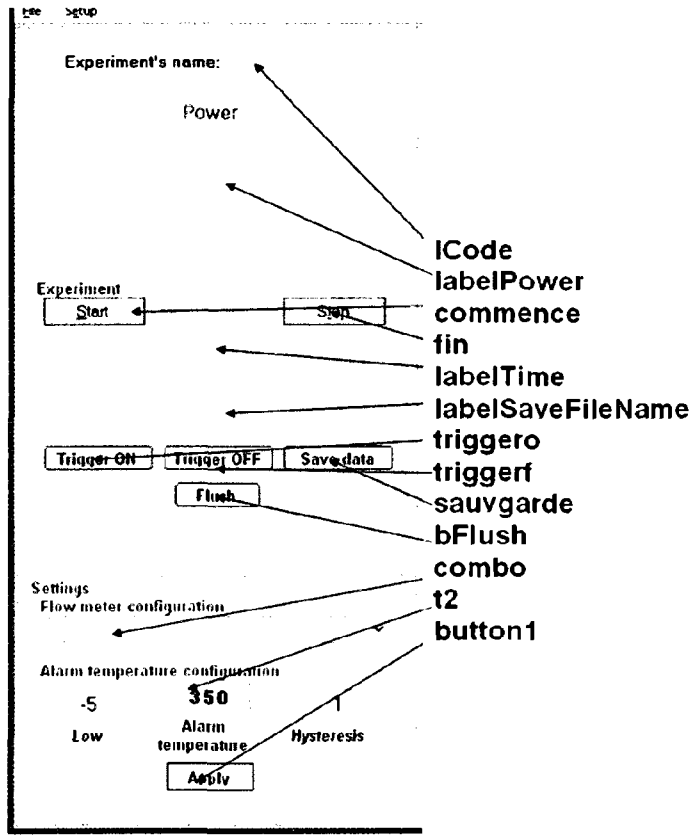

Figure 4.4 : Les objets d'interface graphique de la première colonne pour la fenêtre principale

- **labelSaveFileName** : étiquette qui montre le nom du fichier binaire dans lequel les données ont été sauvegardées;
- **triggero** : bouton qui envoie à l'équipement Tempscan/1000 la commande de commencement de sauvegarde des données dans la mémoire tampon;

- **triggerf** : bouton qui envoie à l'équipement Tempscan/1000 la commande d'arrêt de l'acquisition de données dans la mémoire tampon;
- **sauvgarde** : bouton qui commande la sauvegarde des données qui se trouvent dans la mémoire tampon vers le disque dur;
- **bFlush** : bouton qui vide la mémoire tampon;
- **combo** : boîte combo qui permet de choisir le débitmètre qui sera utilisé pendant les expériences. Cette propriété doit être établie avant que le cycle ne commence;
- **t2** : boîte de texte qui contient la température d'alarme;
- **button1** : bouton qui applique, pendant l'expérience, la température d'alarme spécifiée par **t2**.

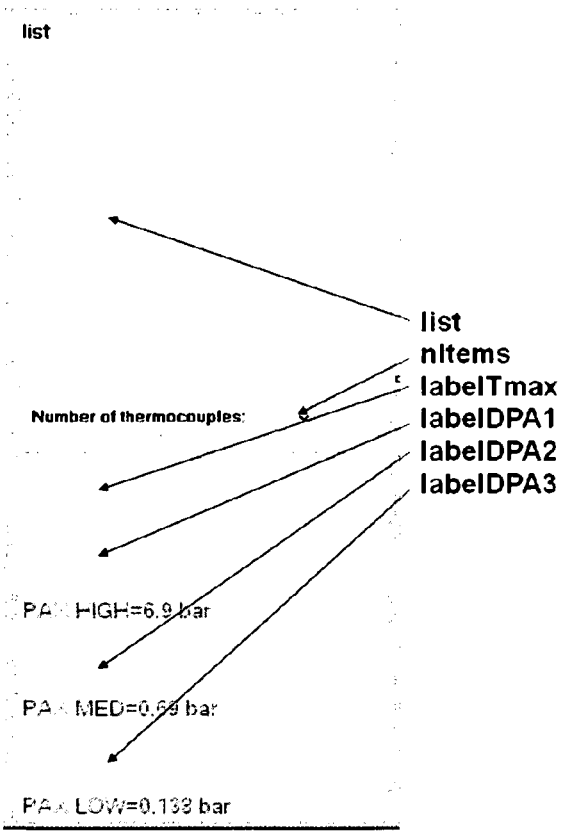

Figure 4.5 : Les objets d'interface graphique de la deuxième colonne pour la fenêtre principale.

La figure 4.5 contient les variables utilisées pour la deuxième colonne :

- **list** : la liste des températures mesurées;
- **nItems** : boîte combo qui établit le numéro des températures affichées;
- **labelTmax** : étiquette dans laquelle on affiche la température maximale déterminée dans la boucle;
- **labelDPA1** : étiquette dans laquelle on affiche la valeur du DPA1
- **labelDPA2** : étiquette dans laquelle on affiche la valeur du DPA2
- **labelDPA3** : étiquette dans laquelle on affiche la valeur du DPA3

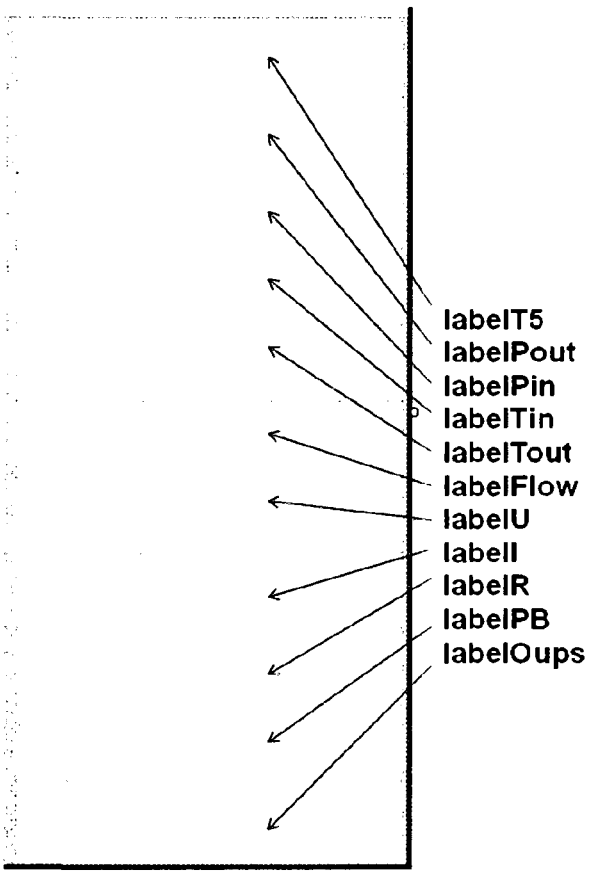

Figure 4.6 : Les objets d'interface graphique de la troisième colonne pour la fenêtre principale

La figure 4.6 montre les positions des objets d'interface pour la troisième colonne. Ces objets sont décrits comme suit :

- **labelT5** : étiquette qui affiche la valeur du T5;
- **labelPout** : étiquette qui affiche la valeur du Pout;
- **labelPin** : étiquette qui affiche la valeur du Pin;
- **labelTin** : étiquette qui affiche la valeur du Tin;
- **labelTout** : étiquette qui affiche la valeur du Tout;
- **labelFlow** : étiquette qui affiche la valeur du Flow;
- **labelU** : étiquette qui affiche la valeur du U;
- **labelI** : étiquette qui affiche la valeur du I;
- **labelR** : étiquette qui affiche la valeur du R;
- **labelPB** : étiquette qui affiche la valeur du power beel;
- **labelOups** : étiquette qui affiche la valeur du bouton oups. Ceci envoi un signal de 9 volts dans un canal pour faciliter la localisation où le chercheur a observé la présence du FCC (i.e., premières fluctuations de température).

Les fils d'exécution gèrent des objets (voir la figure 4.3) qui leur sont propres. À cause de l'utilisation de multiples fils d'exécution, pour pouvoir changer les valeurs du texte contenues à l'intérieur des certains éléments de l'interface, on a dû les déclarer comme « *static* ». Cela signifie que ces variables sont accessibles à partir d'un autre fils d'exécution que celui qui les gère en mode direct.

Par la suite, on va décrire les fonctions de la classe *Form1* :

- *private : System::Void button1_Click (System::Object* sender, System:: EventArgs* e).* Représente le récepteur d'évènements pour le bouton qui commence le suivi des données. Premièrement, la fonction active des boutons et désactive d'autres, pour pouvoir permettre à l'usager d'effectuer seulement certaines opérations, par exemple l'opérateur ne peut pas démarrer le suivi des données après que celui-ci ait déjà démarré. Il va pouvoir le faire seulement après l'arrêt du suivi. Par la suite, si le répertoire dans lequel les données devraient être

sauvegardées n'existe pas, il sera créé. Dans la même fonction, le nom du fichier de sortie sauf son extension, a été construit. Il a été convenu que le nom du fichier va contenir la désignation de l'expérience et la date. L'extension représentera le numéro de l'acquisition effectué dans la journée, mais l'extension est établie au moment de la sauvegarde des données. Par la suite, le fil d'exécution du cycle ayant comme paramètre la fonction *m ()* est déclenché.

- *static String* readItemValue(String *name,bool stopApplicationIfNotFound).* Cette fonction effectue la lecture d'un paramètre à partir du fichier de réglages. Le nom du paramètre est spécifié par le premier argument. Si le paramètre est introuvable, on peut décider si on arrête l'application par l'intermédiaire du deuxième argument ou non.

- *static void readOffsetAndKThcouple(double __nogc *offset,double&kthcouple).* La fonction effectue la lecture des paramètres qui permettent de déterminer les températures. On utilise des fonctions linéaires qui ont la forme suivante :

$$T(i) = (ValeurLue\ (i) + Offset\ (i)) \cdot K_{thetmocoup\ le} \tag{4.1}$$

où : *i* est le numéro du canal lu et *Offset* sont des corrections pour chaque chaine de lecture. Les valeurs de ceux-ci ont été déterminées expérimentalement par les professeurs Alberto Teyssedou et Andrei Olekhnovitch.

- *static void showWait().* Cette fonction démarre le fil d'exécution *messageThread* en montrant la fenêtre de message. Cette fenêtre permet à l'usager d'être informé à tout moment de l'état du système et d'estimer le temps restant. Elle est implémentée dans la classe *wait*.

- *static void setAndShowWait(String *s).* Cette fonction fait la même chose que la fonction précédente et affiche en plus dans la fenêtre de message le texte transmis par le paramètre *s*.

- *static void closeWait ()*. La fonction ferme la fenêtre de message et arrête le fil d'exécution correspondant.

- *static void m ()*. Dans la figure 4.7 sont présentées l'ensemble les opérations effectuées par cette fonction.

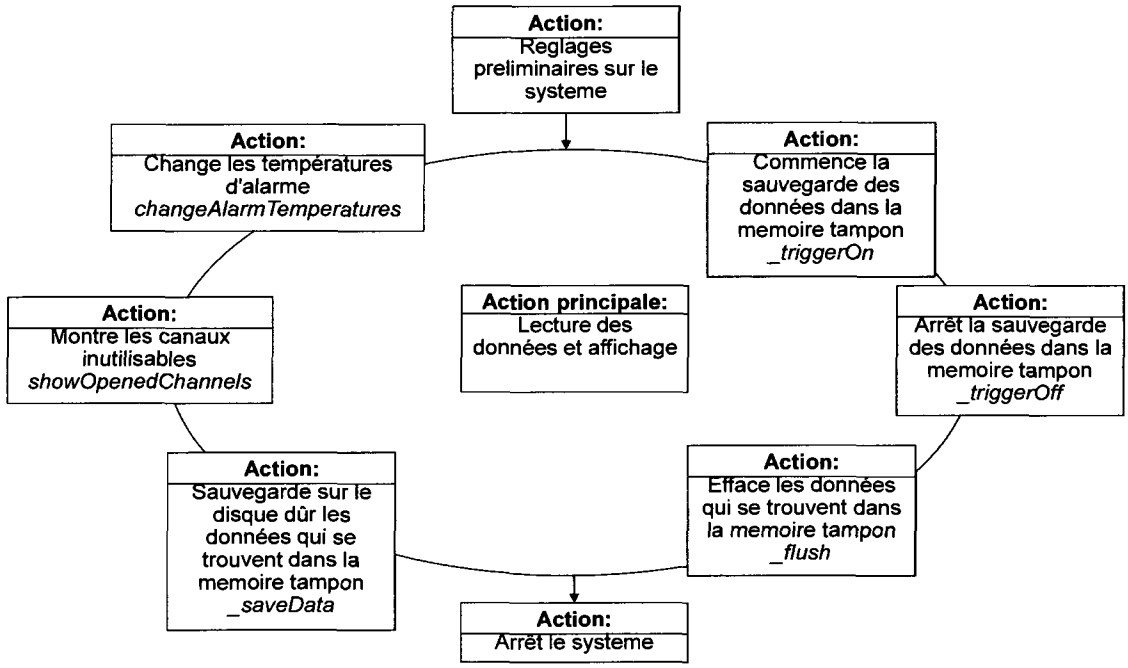

Figure 4.7 : Représentation schématique de la fonction *m ()*.

Cette fonction réalise la mise à jour des données dans les étiquettes situées dans la fenêtre. Elle assure aussi l'exécution des opérations décidées par l'intermédiaire des éléments d'interface pendant la suivie des données. Parce que les fonctions qui interagissent avec l'appareil d'acquisition de données ont parfois un temps de réponse de l'ordre de quelques secondes et pour informer l'usager de l'état du système pendant ces périodes, une fenêtre de message sera démarrée. Par la suite, les numéros des canaux qui seront sauvegardés sont lus (dans le fichier où se trouvent les réglages, on trouve aussi l'état de chaque canal). L'étape suivante est de lire la valeur des différents paramètres qui seront utilisés par la suite pour

déterminer les valeurs des températures ou d'autres caractéristiques suivies par le système d'acquisition de données.

L'objet qui contient les fonctions et les réglages pour commander le système d'acquisition de données est créé par la suite. Celui-ci représente une instance de la classe *determination* qui est implémentée dans le fichier « determination.h ». Par la suite, le système est redémarré et reconfiguré, en utilisant la fonction *start* implémentée dans la classe *détermination.* On fait cela pour effacer les éventuels réglages antérieurs. Il suit la fermeture de la fenêtre des réglages pour permettre à l'usager d'accéder à la fenêtre principale de l'application. À partir de ce moment, dans un cycle on décide quels sont les réglages à faire pour le système et après cela, on lit et affiche les valeurs des paramètres mesurés dans la boucle. Les réglages à faire sont décidés en utilisant des variables bit. Ces variables sont les suivantes :

- *_triggerOn* – pour déclencher l'acquisition de données dans la mémoire tampon du système;
- *_flush* – pour effacer les données sauvegardées dans la mémoire tampon du système;
- *_triggerOff* – pour arrêter l'acquisition de données dans la mémoire tampon;
- *_saveData* – pour sauvegarder les données de la mémoire tampon du système vers le disque dur de l'ordinateur. Les données sont sauvegardées sous leur forme intégrale en format binaire. Les données sélectionnées sont sauvegardées automatiquement dans un sous-répertoire en forme décimale pour pouvoir être vérifiées, interprétées et par la suite utilisées dans d'autres applications;
- *showOpenedChannels* – pour déterminer quels sont les canaux qui présentent des valeurs qui sortent du domaine. Les valeurs lues par le

système d'acquisition de données sont en réalité des tensions. Une valeur trop élevée montrera que, pour ce canal le senseur n'est pas branché;

- *_changeAlarmTemperatures* – pendant les expériences, on arrive souvent à des situations limites où les différentes températures dans la boucle dépassent les valeurs permises. Pour éviter les situations dangereuses, par l'intermédiaire du déclencheur digital inclus dans l'équipement Tempscan/1000, on arrête le système de chauffage de la boucle. La variable est utilisée pour déclencher la mise en marche du changement de la température d'alarme.

- *private : System::Void onstart(System::Object * sender, System:: EventArgs * e).* La fonction s'exécute à l'ouverture de la fenêtre. La première action effectuée est de sélectionner la valeur maximale de la boîte combinée qui établit le numéro des températures affichées. Si le fichier de réglages n'existe pas, ou qu'il existe mais qu'il n'a pas de débitmètres enregistrés à l'intérieur, la fenêtre qui gère les débitmètres sera ouverte. On fait cela pour pouvoir avoir au moins un débitmètre disponible dans la liste. Pour le nom de l'expérience, le procédé est répété comme il a été présenté à l'étape antérieure. Les caractéristiques des débitmètres seront lues à partir du fichier de réglages.

- *private : System::Void fin_Click(System::Object* sender, System::EventArgs * e).* Cette fonction représente le récepteur des évènements pour le bouton *fin* qui achève le suivi des expériences. Premièrement, certains éléments d'interface seront activés et désactivés pour ne pas pouvoir effectuer par la suite des opérations spécifiques à l'acquisition de données. Par la suite, le variable « bit », qui contrôle l'exécution du cycle contenu dans la fonction *m ()*, sera mise à la valeur d'arrêt (=1). Ce changement sera visible dans le fil d'exécution *cycleThread* et arrêtera le cycle.

- *private : System::Void combo_SelectedIndexChanged(System::Object * sender, System::EventArgs * e).* Représente le récepteur des évènements pour la boîte combo *nItems*. Elle est appelée quand on choisit une autre valeur. La valeur du paramètre sera changée pour la nouvelle valeur choisie et le fichier de réglages sera lui aussi modifié;

- *private : System::Void f1(System::Object * sender, System::EventArgs * e).* Représente le récepteur des évènements pour plusieurs boîtes de texte qui accepte seulement des valeurs numériques réelles. Elle est appelée quand on change le texte contenu par ces boîtes. Si le dernier caractère ne permet pas par la suite d'obtenir un numéro réel dans le format anglais, l'usager sera informé de cela par l'intermédiaire d'une fenêtre de message automatiquement générée par le compilateur et le caractère en question sera éliminé.

- *private:System::Void triggero_Click(System::Object * sender, System::EventArgs * e).* La fonction représente le récepteur des événements pour le bouton *triggero.* On active et on désactive certains éléments d'interface pour ne pas pouvoir effectuer par la suite des opérations qui peuvent empêcher l'acquisition de données. Par la suite, on met la variable bit *_triggerOn* à la valeur qui va pouvoir déclencher dans la fonction *m ()* l'acquisition de données dans la mémoire tampon du système.

- *private: System::Void triggerf_Click(System::Object * sender, System::EventArgs * e).* Cette fonction représente le récepteur des événements pour le bouton *triggerf.* On active et on désactive des éléments d'interface pour ne pas pouvoir effectuer par la suite des opérations spécifiques à l'acquisition de données. On met la variable bit *_triggerOn* à la valeur qui pourra déclencher, dans la fonction *m (),* l'arrêt du processus d'acquisition de données dans la mémoire tampon du système.

- *private : System::Void f2 (System::Object * sender, System::EventArgs * e).* Représente le récepteur des évènements pour plusieurs boîtes de texte qui acceptent seulement des valeurs numériques réelles. Elle est appelée quand on quitte la boîte. Si le champ de saisie est vide, on affiche dans la boîte la valeur « 0 ».

- *private : System::Void sauvgarde_Click(System::Object * sender , System:: EventArgs * e).* Cette fonction représente le récepteur des événements pour le bouton *sauvgarde.* La variable qui déclenchera la sauvegarde des données dans la fonction *m ()* est mise à la valeur correspondante à cette action (=1). Par la suite, on active et on désactive des éléments d'interface pour mettre les boutons dans le même état qu'avant le commencement de l'acquisition de données.

- *private : System::Void menuItem2_Click_1(System::Object * sender, System::EventArgs * e).* Cette fonction est exécutée quand on choisit à partir du menu l'option *Setup->Set the experiment code.* Elle ouvre une instance de la fenêtre *Name* qui est implémentée dans le fichier *Name.h*, qui fera par elle-même le réglage du nom de l'expérience. On établit par la suite quels sont les canaux qui seront sauvegardés automatiquement en créant un objet de type *SetupDecimal.*

- *void extractFlow(String *in,String*&name,double& k1,double& k2).* Cette fonction réalise l'extraction à partir de la variable *in* du nom du débitmètre et de ses coefficients. Ceux-ci permettent de calculer le débit. Le débit est calculé à partir de la formule suivante :

$$débit=(VoltageLu+Offset)\cdot kflow1-kflow2 \qquad (4.2)$$

- *void fillCombo().* Cette fonction remplit l'élément de l'interface *combo* par les débitmètres qui se trouvent dans le fichier de réglages.

- *private : System::Void menuItem2_Click_2(System::Object * sender , System:: EventArgs * e).* Cette fonction est appelée quand on utilise l'item du menu Setup--> *Add/Remove flow meter.* On ouvre une fenêtre de réglages sur les débitmètres, qui est implémentée dans la classe *Debitmetre.*

- *private : System::Void bFlush_Click(System::Object * sender, System::EventArgs * e).* Cette fonction représente le récepteur des événements pour le bouton *bFlush.* On active et on désactive des éléments d'interface pour mettre les boutons dans le même état qu'avant le commencement de l'acquisition de données.

- *private : System::Void form1_closing(System::Object * sender , System:: ComponentModel::CancelEventArgs * e).* Cette fonction s'effectue avant que la fenêtre principale se ferme. On a dû implémenter cette fonction à cause du comportement différent de la fenêtre si on pousse le bouton qui devrait fermer la fenêtre (le X du coin droite en haut). Si on appuie sur le bouton quand on fait le suivi des données, on devrait d'abord annuler le fil d'exécution qui gère le dialogue avec le système d'acquisition de données et seulement après la fermeture la fenêtre principale. Si par contre, on ferme la fenêtre quand on ne fait pas le suivi des données, on laisse les évènements se dérouler de façon normale.

- *private : System::Void menuItem2_Click_3(System::Object * sender, System:: EventArgs * e).* Cette fonction est exécutée quand on choisit à partir du menu l'option *File-> Show the opened channels.* Elle met la variable qui déclenchera dans la fonction *m*() la procédure de détermination des canaux, qui sont ouverts ou qui sont défectueux à la valeur correspondante (= true).

- *private : System::Void button1_Click_3(System::Object * sender, System :: EventArgs * e).* Cette fonction représente le récepteur d'événement pour le bouton

button1 correspondant à l'action où le bouton est pressé. Elle met la variable *_changeAlarmTemperatures* à la valeur correspondante au déclenchement dans la fonction *m()* du changement en marche de la température d'alarme.

- *static void replaceOrAddValue(String*name,String *value).* Cette fonction est utilisée pour effectuer des opérations dans le fichier de réglages. Elle remplace la valeur spécifiée dans la variable d'entrée *name,* par la valeur spécifiée du deuxième paramètre. S'il n'y a pas de variable avec ce nom dans le fichier de réglages, la fonction ajoutera à la fin du fichier une nouvelle ligne qui va contenir le nom de la variable de réglage et sa valeur. Si le fichier n'existe pas, on le créera et on effectuera les opérations antérieurement spécifiées.

- *static void readChannelsToSave().* Cette fonction lit à partir du fichier de réglages quels seront les canaux qui seront sauvegardés automatiquement dans le format décimal.

- *private :System::Void size_changed(System::Object* sender, System::débitmètre* e).* Cette fonction représente un récepteur de l'événement pour le changement de forme de la fenêtre principale. Les éléments d'interface sont ajoutés sur la superficie de la fenêtre en respectant une politique de positionnement, qui permettra de les distribuer de manière uniforme verticalement et horizontalement. La grandeur des éléments par contre ne varie pas et, à cause de cela, on doit régler celle-ci chaque fois que l'usager change la grandeur de la fenêtre.

- *private : System::Void on_enter_commence(System::Object* sender, System:: débitmètre* e).* Cette fonction représente le récepteur d'événement du bouton *commence,* correspondant au moment où la souris entre sur la superficie du bouton. On a considéré qu'il est extrêmement important d'attirer l'attention de l'usager sur le bon choix de la température d'alarme. Pour faire cela, chaque fois

que l'usager se prépare à commencer le suivi des données un texte apparaît à côté du champ de saisie *t2* pour le renseigner.

- *private : System::Void on_leave_commence(System::Object* sender, System::EventArgs* e)*
  Cette fonction représente le récepteur événement du bouton *commence,* correspondant au moment où la souris sort de la superficie du bouton. Elle cache seulement le texte avertisseur.

- *private : static void saveToDecimal(double *offset,double kthcouple,String *binaryFileName).* Cette fonction est utilisée pour sauvegarder en format décimal les données qui sont initialement stockées dans un fichier binaire. Les trois paramètres sont :

  - *double *offset* sont les paramètres de correction appliqués aux lectures de thermocouples;
  - *double kthcouple* représente le facteur de multiplication nécessaire pour déterminer les températures;
  - *String binaryFileName* c'est le fichier d'entrée.

  Dans le répertoire où se trouve le fichier binaire, s'il n'existe pas, on crée un répertoire nommé "*decimal*". Dans ce répertoire sera écrit le fichier texte. Le nom du fichier commencera par le préfixe *dec_*.

4.2 La classe « determination.h »

Cette classe implémente la communication avec le système d'acquisition de données. Les variables de la classe sont les suivants :

- *int disp* spécifie l'identificateur du système d'acquisition;
- *double off[43]* c'est le vecteur des valeurs de correction sur les températures et les autres paramètres déterminés dans la boucle de l'eau;
- *double kth* est le facteur de multiplication nécessaire à la détermination de la température mesurée par un thermocouple quelconque;
- *double tmin, tmax, thisteresis* sont les températures utilisées dans le réglage des alarmes. Le premier est la température inférieure d'alarme, le deuxième est la température supérieure et le troisième est l'écart nécessaire pour sortir de l'état d'alarme, comme expliqué précédemment.

Par la suite, on va expliquer de manière plus détaillée les fonctions de la classe :

- *détermination ()*. Représente le constructeur de la classe par défaut;

- *determination(double s1,double s2,double s3,double k,double offset[43]).* Représente elle aussi un constructeur de la classe qui a les paramètres_suivants : la température minimale d'alarme, la température maximale d'alarme, l'écart nécessaire pour sortir de l'état d'alarme, la valeur du facteur de proportionnalité pour déterminer les températures, l'ensemble des valeurs de corrections sur les températures.

- *int Err(int disp).* Cette fonction détermine le code d'erreur du système d'acquisition de données. Le paramètre représente l'identificateur du système (dans la quasi-totalité des cas cette variable a la valeur 1).

- *void OutputNew(int ieee, char* command).* Cette fonction envoie au système d'acquisition de données, qui est spécifié par le premier paramètre, l'instruction spécifiée dans le deuxième paramètre. Elle vérifie au début si le système ne se trouve pas dans un état d'erreur et s'il peut donc effectuer la commande.

- *double *decomp(char *s,int &ne).* Cette fonction est utilisée pour transformer le message reçu de la part du système d'acquisition quand on lui demande de nous envoyer les données de la dernière lecture. Les paramètres sont donnés dans l'ordre suivant : une chaîne de caractères contenant l'ensemble des données reçues et le nombre de chaînes lues (valeur qui sera déterminée par cette fonction). On retourne un vecteur qui contient les valeurs réelles.

- *void putFloatToString(double in, int nr_digits,char* &c).* Cette fonction est utilisée pour pouvoir manipuler le format des valeurs réelles. Elle met sous le format d'une chaîne de caractère un nombre réel (double), en gardant seulement le nombre indiqué par les chiffres décimaux. Les paramètres sont dans l'ordre suivant : le nombre réel initial, le nombre de décimales, la chaîne de caractères dans laquelle on met la réponse.

- *void setupAlarms(double T_low, double T_high, double T_histeresis).* Cette fonction réalise le réglage des canaux et des alarmes en parallèle avec une information dans la fenêtre des messages. Les trois paramètres sont les valeurs en format double des températures normalisées dans le domaine du thermocouple. Pour certaines instructions on a dû laisser du temps à l'équipement pour les exécutions.

- *void beforesetupAlarms().* On utilise cette fonction seulement quand on change les valeurs d'alarme pendant le suivi des données. Elle redémarre le système à froid et effectue des réglages sur la représentation des données. En parallèle, on informe l'usager de l'étape dans laquelle se trouve le système, en utilisant la fenêtre des messages.

- *void start()*. Cette fonction effectue les réglages initiaux du système d'acquisition de données. On obtient un numéro d'identification pour notre système, en utilisant la fonction *OpenName()* du pilote informatique. Cette valeur est attribuée à la variable *disp* de la classe *détermination* pour pouvoir, par la suite, retrouver le système. On effectue également tous les autres réglages comme le format de données en sortie, on fait le choix sur le type de chaque canal et on attribue aussi les alarmes. En parallèle, on informe l'usager de l'étape dans laquelle se trouve le système, en utilisant la fenêtre des messages.

- *double * shot(int &n)*. Cette fonction est utilisée pour récupérer les données utilisées pour le suivi. Comme première étape, la fonction envoie la commande vers le système pour obtenir les dernières données. Après cela, elle utilise la fonction *Enter* du pilote informatique pour récupérer les données sous la forme d'une chaîne de caractères. Ces données seront retournées par la fonction.

- *void stop ()*. Cette fonction effectue les opérations d'arrêt du suivi des données. La première opération effectuée est de redémarré à froid le système. Par la suite, la communication avec celui-ci est interrompue et l'identificateur du système est désactivé en utilisant la fonction *Close* du pilote informatique. En parallèle, l'usager est informé sur l'opération courante par l'intermédiaire de la fenêtre des messages.

- *static void movingBar(int min=0,int max=100,int nr=100,int tt=6000)*. Cette fonction est utilisée pour bouger la barre qui montre, dans la fenêtre des messages, l'avancement du processus.

- *void emptyBuffer()*. Cette fonction efface les données qui se trouvent dans la mémoire tampon du système. En parallèle, l'usager est informé sur l'opération courante par l'intermédiaire de la fenêtre des messages.

- *void triggerChange().* Cette fonction est également utilisée au début de l'acquisition de données et à la fin.

- *void saveBuffer(String* path_file, int &ne,double kflow1, double kflow2,char fN[100]).* Cette fonction sauvegarde les données qui se trouvent dans la mémoire tampon sur le disque dur de l'ordinateur. Les paramètres de la fonction sont dans l'ordre suivant : le nom du fichier de sortie sauf l'extension (l'extension représente le numéro du fichier sauvegardé dans la même journée), le numéro du fichier sauvegardé pour l'expérience effectuée dans la même journée, les deux paramètres qui permettront de déterminer le débit, une variable de type chaîne de caractères qui va garder le nom du fichier de sortie. Si on lit une donnée à la fois, le processus sera très lent parce que les fonctions du pilote informatique qui seront utilisées ont un temps de réponse assez long. La fonction déboute par la détermination du nombre des enregistrements sauvegardés dans la mémoire tampon et, parce que les données peuvent être très nombreuses (ce qui peut générer des problèmes d'espace de stockage dans la mémoire pendant l'exécution – *stack overflow*), des paquets de 10000 données réelles binaires seront lues. Ces données seront redirigées par la suite vers le fichier de sortie. L'usager sera informé en parallèle de la proportion dans laquelle les données sont enregistrées dans la fenêtre des messages.

4.3 La classe « Debitmetre.h »

Cette classe représente une fenêtre, donc est une extension de la classe *Form,* qui permet à l'usager d'effectuer des réglages sur la liste des débitmètres. On peut donc introduire et sortir des débitmètres de la liste des débitmètres disponibles. On considère que les caractéristiques des débitmètres sont : le nom, *kflow1* et *kflow2*. Ces deux coefficients ont été expliqués plus tôt dans la section 4.1. Cette classe contient les variables suivantes :

- *private : System::Windows::Forms::Button * button1*; bouton qui est utilisée pour fermer la fenêtre.
- *private : System::Windows::Forms::Button * button2*; bouton qui est utilisée pour introduire un nouveau débitmètre dans le fichier et dans la liste qui les affiche dans la fenêtre.
- *private : System::Windows::Forms::ListBox * lb*; liste qui permet l'affichage des débitmètres sur la superficie de la fenêtre.
- *private : System::Windows::Forms::TextBox * description*; champ de saisie utilisé pour spécifier un nom générique pour le débitmètre, qui par la suite sera écrit dans le fichier de réglages et dans la liste *lb*.
- *private : System::Windows::Forms::TextBox * kf1*; champ de saisie utilisé pour spécifier le premier coefficient pour le débitmètre, qui par la suite sera écrit dans le fichier de réglages et dans la liste *lb*.
- *private : System::Windows::Forms::TextBox * kf2*; champ de saisie utilisé pour spécifier le deuxième coefficient pour le débitmètre, qui par la suite sera écrit dans le fichier de réglages et dans la liste *lb*;
- *private : System::Windows::Forms::Label * label1;*
- *private : System::Windows::Forms::Label * label2;*
- *private : System::Windows::Forms::Label * label3;*
  Les trois étiquettes sont utilisées pour spécifier le contenu des boîtes de texte *description*, *kf1* et *kf2*.
- *private : System::Windows::Forms::Button * button3*; ce bouton est utilisé pour effacer le débitmètre sélectionné dans la liste *lb* du fichier de réglages et de la liste.

On va présenter par la suite les principales fonctions de cette classe :

- *public : static String*getFlowMeters()[].* La fonction exécute la lecture des données des débitmètres qui se trouvent dans le fichier de réglages. Elle retourne

un vecteur de pointers vers des variables du type String contenant le nom du débitmètre, les premier et deuxième coefficients qui seront utilisés pour déterminer le débit.

- *private :System::Void button3_Click(System::Object * sender, System::EventArgs * e).* Cette fonction représente le récepteur des événements pour le bouton *button3*. On efface le débitmètre sélectionné dans la liste *lb* du fichier de réglages. L'opération s'exécute par l'intermédiaire d'un fichier temporaire où on copie le contenu du fichier initial de réglages sauf la ligne correspondante au débitmètre sélectionné. Par la suite, on remplace le fichier de réglages par le fichier temporaire. On efface aussi le débitmètre de la liste.

- *private :System::Void button2_Click(System::Object * sender, System::EventArgs * e).* Ceci représente le récepteur des événements pour le bouton *button2*. On ajoute dans le fichier de réglages une nouvelle ligne correspondante aux paramètres spécifiés dans les boîtes de texte.

- *private : System::Void f1(System::Object * sender, System::EventArgs * e).* Cette fonction est un récepteur d'évènements pour le changement du texte dans les champs de saisie qui contiennent des nombres réels. Elle est similaire à la même fonction de la classe *Form1* et elle a le même comportement.

- *private : System::Void Debitmetre_Load (System::Object * sender, System:: EventArgs * e).* Cette fonction est exécutée à l'ouverture de la fenêtre. Elle ajoute dans la liste, les débitmètres qui se trouvent dans le fichier de réglages.

- *private :System::Void button1_Click(System::Object * sender, System::EventArgs * e).* Cette fonction représente le récepteur des événements pour le bouton

button1. Elle ferme la fenêtre si, dans la liste et dans le fichier de réglages se trouvent au moins les données d'un seul débitmètre.

4.4 La classe « Name.h »

Cette classe hérite la classe *Form* et représente donc aussi une fenêtre. On utilise cette fenêtre pour établir le code de l'expérience. Elle ne contient qu'un champ de saisie (*nom*) et un bouton (*button1*). Les fonctions de la classe sont les suivantes :

- *private :System::Void button1_Click(System::Object * sender, System::EventArgs * e).* Cette fonction représente le récepteur des événements pour le bouton. Si la chaîne de caractères qui se trouve dans le champ de saisie respecte les conditions pour être un nom d'expérience, il remplace l'ancien nom dans le fichier de réglages et la fenêtre est fermée.

- *void replaceOrAddValue(String*name,String *value).* Cette fonction attribue dans le fichier de réglages au paramètre spécifié par *name* la valeur spécifiée par *value*. Le changement se produit par l'intermédiaire d'un fichier temporaire. On copie dans le fichier temporaire le contenu du fichier de réglages sauf la ligne correspondante à la valeur qui doit être changée. Cette ligne est produite et écrite dans le fichier temporaire, mais ayant la valeur changée par le deuxième paramètre de la fonction. Après cela, on remplace le fichier de réglages par le fichier temporaire.

- *String* readItemValue(String *name,bool stopApplicationIfNotFound).* Cette fonction représente une copie de la fonction qui a le même nom dans la classe *Form1*.

- *private : System::Void Name_Load(System::Object* sender, System::EventArgs * e).* Cette fonction s'exécute à l'ouverture de la fenêtre. Elle lit dans le fichier de réglages le nom actuel de l'expérience et le met dans le champ de saisie *nom.*

4.5 La classe « Wait.h »

Cette classe représente une extension de la classe *Form*; elle représente donc une fenêtre. On l'utilise pendant le démarrage du système, l'arrêt du système, quand on sauvegarde les données sur le disque et quand on efface la mémoire tampon. Les variables contenues dans cette fenêtre sont seulement les éléments de l'interface :

- Deux pointeurs, un vers l'étiquette *label1* qui représente un message général et un autre vers *label2* qui va donner une information sur l'étape dans laquelle se trouve le système;
- Un pointeur vers un élément de la classe *ProgressBar* qui présente une représentation approximative du temps d'exécution.

Pour pouvoir accéder à ces variables à partir d'autres fils d'exécution, on les a déclarées comme *static.* On a expliqué plus tôt la raison de ce choix. La classe ne contient pas de fonctions sauf celles générées automatiquement par le « *builder* » pour une fenêtre donnée.

4.6 La classe « SetupDecimal.h »

Cette classe hérite aussi de la classe *Form.* Elle est utilisée pour établir quels seront les paramètres mesurés dans la boucle qui vont être sauvegardés en format texte après avoir mis en mémoire toutes les données en format binaire. Cette fenêtre contient une série de pointers ou éléments d'interface suivants :

- Un pointer vers un élément de type *CheckedListBox* nommé *list.* Les éléments affichés dans ce type de liste ont chacun sur leur côté une case à cocher. On va établir donc quels sont les canaux qui seront sauvegardés en cochant les cases correspondantes;
- Le pointer vers un élément de type Button nommé *button2,* qui représente le bouton qui sauvegarde dans le fichier de réglages les changements et qui ferme la fenêtre;
- Le pointer vers un élément de type Button nommé *button3,* qui représente le bouton utilisé pour cocher les boîtes entre l'index spécifié par les variables *mini* et *maxi*;
- Le pointer vers un élément de type Button nommé *button4*, qui représente le bouton utilisé pour décocher les boîtes entre l'index spécifié par les variables *mini* et *maxi*;
- Le pointer vers un élément de type Button nommé *buttonCheckAll,* qui représente le bouton utilisé pour cocher toutes les cases à cocher;
- Le pointer vers un élément de type ComboBox nommé *mini,* qui représente une boîte combinée qui spécifie le premier index pour les actions des boutons *button3* et *button4*;
- Le pointer vers un élément de type ComboBox nommé *maxi,* qui représente une boîte combinée qui spécifie le dernier index pour les actions des boutons *button3* et *button4.*

D'autre part les fonctions appartenant à cette classe sont :

- *void setCheckedStatusInList().* Cette fonction lit le statut de chaque canal à partir du fichier de réglages et établit de cette manière l'état de chaque case à cocher.

- *static void replaceOrAddValue(String*name,String *value).* Cette fonction est une copie de la fonction qui a le même nom et qui est située dans la classe *Form1.*

- *static String* readItemValue(String *name,bool stopApplicationIfNotFound).* Cette fonction est une copie de la fonction qui a le même nom et qui est située dans la classe *Form1*.

- *private : System::Void button2_Click(System::Object* sender, System::EventArgs * e).* Cette fonction est le récepteur des événements pour le bouton *button2*. On replace dans le fichier de réglages, pour chaque canal, les valeurs qui spécifient si le canal sera sauvegardé ou non dans le fichier texte. Par la suite, la fenêtre est fermée.

- *private : System::Void button3_Click(System::Object* sender, System::EventArgs * e).* Cette fonction est le récepteur des événements pour le bouton *button3*. Elle coche toutes les cases à cocher entre les valeurs choisies du *mini* et *maxi*.

- *private :System::Void button4_Click(System::Object * sender, System::EventArgs * e).* Cette fonction est le récepteur des événements pour le bouton *button4*. Elle décoche toutes les cases à cocher entre les valeurs choisies du *mini* et *maxi*.

- *private : System::Void buttonCheckAll_Click(System::Object * sender, System:: EventArgs * e).* Cette fonction est le récepteur des événements pour le bouton *buttonCheckAll*. Elle coche toutes les cases à cocher pour tous les canaux.

- *private : System::Void mini_SelectedIndexChanged(System::Object * sender, System::EventArgs * e).* Cette fonction est le récepteur des événements pour la boîte combinée *mini* quand on change l'élément sélectionné. Si la valeur de la boîte combinée *maxi* est plus petite que la valeur du *mini,* on change convenablement la valeur du *maxi*.

- *private : System::Void maxi_SelectedIndexChanged(System::Object * sender, System::EventArgs * e).* Cette fonction est le récepteur des événements pour la boîte combinée *maxi* quand on change l'élément sélectionné. Si la valeur de la boîte combinée *maxi* est plus petite que la valeur du *mini,* on change convenablement la valeur du *mini.*

- *static void readOffsetAndKThcouple(double __nogc *offset,double&kthcouple).* Cette fonction représente une copie de la fonction qui a le même nom dans la classe *Form1.*

- *private : System::Void Form1_Load(System::Object * sender, System::EventArgs * e).* Cette fonction s'exécute à l'ouverture de la fenêtre. Elle coche les canaux de la liste en fonction du contenu du fichier de réglages.

4.7 Les tests de fonctionnement et de validation

Le but de ce travail a été de développer un logiciel d'acquisition de données qui sera plus robuste, plus stable et plus flexible, pour être adapté à d'autres systèmes d'exploitation comme Windows XP ou Vista. De plus le nouveau logiciel devait être compilé en 32 bits. Le tableau 4.1 contient les paramètres vérifies pendant la validation du logiciel SADLT version 1.0.

En concordance avec les objectifs fixés pour ce projet des tests ont été effectuées. Les tests de fonctionnement et de validation du logiciel d'acquisition des données se sont déroulés dans la période entre 23.06.2008 et 26.01.2009 dans la présence des professeurs Alberto Teyssedou et Andrei Olekhnovitch et des étudiants François-Xavier de Cordoüe-Hecquard et Altan Muftuoglu. L'application à été utilisée en alternance avec le programme initial d'acquisition des données. Chaque option du logiciel a été testée et les résultats et comportements observes ont été compares avec les résultats obtenus par

l'application de référence. Le tableau 4.1 présente le sommaire des déterminations et calculs qui ont été vérifiées pour des fins de validation.

Tableau 4.1 Les opérations vérifiées pour des fins de validation.

Opération	Logiciel de référence (VB)	Le nouveau logiciel (C++)
Lecture des températures	✓	✓
Lecture de la pression d'entrée	✓	✓
Lecture de la pression de sortie	✓	✓
Lecture de la température d'entrée	✓	✓
Lecture de la température de sortie	✓	✓
Lecture du débit	✓	✓
Lecture de l'intensité du courant électrique	✓	✓
Lecture de la tension électrique	✓	✓
Détermination de la température maximale	✓	✓
Élimination des erreurs systématiques introduites par les chaînes de mesure	✓	✓
Application des corrections	✓	✓
Calcul de la puissance	✓	✓
Fonctionnement de l'alarme	X	✓
Compatibilité système d'exploitation 32 bits	X	✓
Fréquence d'interruption du système	0.125 fois / heure	0.004 fois / heure

On conclut que le logiciel réponde à la demande initiale du personnel du Laboratoire Thermo-hydraulique de l'IGN.

Chapitre 5

Le système de contrôle pour la boucle au Fréon

Pour la boucle au Fréon, on utilise pour la partie « *hardware* » un ensemble des équipements nommé *I/O95* qui est réalisé par la compagnie MTL Inc. Pour la partie « *software* » on utilise une application spécialisée en interface et contrôle des équipements industriels nommée « Nematron Paragon ».

Ces systèmes permettent de réaliser le contrôle de la boucle et le suivi des certains données d'opération. On peut aussi effectuer une acquisition de données mais à une vitesse relativement faible (i.e., un maximum de 100 lectures par seconde). Le système I/O 95 fournit une interface intelligente entre l'ordinateur et les détecteurs et déclencheurs montés pour réaliser une expérience (i.e., pression, sous refroidissement, débit, etc.). Il est formé par des modules digitaux d'entrée et de sortie, des modules analogiques d'entrée et de sortie, des blocs d'alimentation, des bases d'arrêt, des interfaces et des logiciels (pilote) pour pouvoir communiquer avec l'ordinateur. Dans la figure 5.1 un exemple de configuration est présenté. Dans les sections suivantes, chaque élément qui compose le système de contrôle sera présenté, en partant des dispositifs matériels.

5.1 Interface : « Intelligent Input/Output Processor (IIOP) »

L'IIOP est une carte qui assure l'interface pour l'ordinateur avec le système I/O95 (voir la figure 5.1). Elle assure la communication entre le pilote et le réseau d'I/O 95.

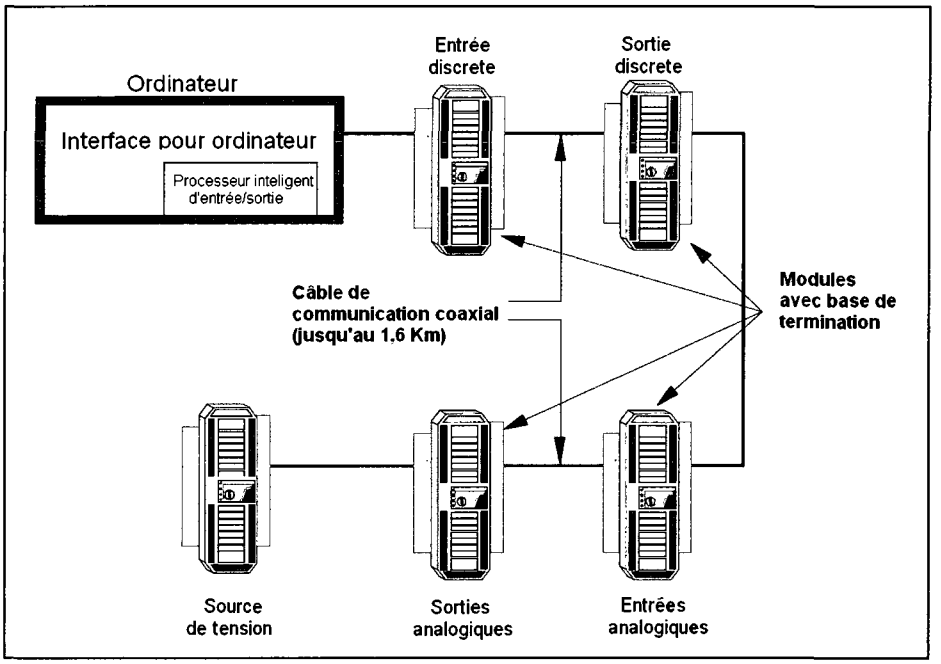

Figure 5.1 : Exemple de modules couplés par un réseau de communication TransNet à une interface IIOP

Pour pouvoir effectuer ces opérations, l'interface utilise les équipements suivants :

- 64 Kb de mémoire RAM;
- Le processeur de gestion de mémoire;
- Le processeur pour la gestion des communications;
- Une mémoire pour l'échange des données.

Les fonctions les plus importantes d'IIOP sont résumées comme suit :

- La communication avec le réseau TransNet;
- La sauvegarde dans la mémoire des données qui sont reçues de la part des modules d'entrée et qui donne accès à celles transmises à l'ordinateur;
- L'envoi des données obtenues à partir de l'ordinateur vers les modules de sortie;
- L'interprétation des commandes qui proviennent de l'ordinateur.

Il faut mentionner que l'IIOP a deux états de fonctionnement : mode émetteur et mode commande/réponse.

Le premier état correspond à l'état normal de fonctionnement, quand l'IIOP assure la communication à grande vitesse pour le contrôle en temps réel. Dans cet état, le système effectue automatiquement la collection, le dépôt et l'écriture de données d'entrée/sortie pour tous les modules. Dans le deuxième état, l'IIOP effectue la configuration des modules. Cet état est utilisé en intercalant le mode émetteur pour pouvoir avoir accès à certaines caractéristiques des modules. Dans cet état, le système effectue automatiquement la collection, le dépôt et l'écriture de données d'entrée/sortie pour seulement une partie des modules.

5.2 Le réseau de communication TransNet

Ce réseau assure la communication entre les modules d'entrée-sortie et l'interface. Il consiste en un câble coaxial et un protocole de communication TransNet. La longueur maximale du câble peut aller jusqu'à 1.5 km. La fréquence de communication est de 375 Kbaud avec une modulation par déplacement de fréquence entre 1.5 et 2 MHz. Le protocole de communication est basé sur une distribution du temps de dialogue entre les modules, qui est synchronisé par un signal qui part de l'interface. Le temps alloué pour chaque module est établi au moment où s'effectue la configuration du réseau. Pour vérifier l'exactitude des données, le réseau utilise un contrôle de redondance cyclique (CRC).

La topologie du réseau est une configuration en guirlande dont chaque module est directement connecté à la ligne principale. Les câbles du réseau TransNet sont connectés à une base de branchement de modules pour l'opération des entrées-sorties. Les bases de

branchement sont désignées pour offrir un chemin de communication continu. Le réseau ne supporte pas directement des configurations en anneau, arbre ou étoile.

5.3 Les bases de branchement

Ces bases sont conçues pour pouvoir brancher les modules des entrées-sorties. Elles disposent entre autres des principales caractéristiques suivantes :

- Des connecteurs pour les capteurs et les déclencheurs;
- Des fusibles pour les canaux de sortie, l'alimentation et la communication;
- Des connecteurs pour monter les modules d'entrées-sorties.

Le seul composant électronique actif dans les bases est l'EEPROM (*Electrically-Erasable Programmable Read-Only Memory* ou mémoire effaçable électriquement et programmable), actif quand le module est branché, qui sauvegarde les paramètres de réglages de celui-ci.

5.4 Les modules d'entrées-sorties

Les modules sont produits pour plusieurs valeurs de signaux et de types pour pouvoir s'adapter aux demandes de la plupart des applications. Il y a des modules qui permettent le branchement de 8, 16 et 32 canaux en mode commun ou différentiel. Les opérations effectuées par les modules peuvent être résumées comme suit :

- Effectuer la lecture et écriture des canaux d'entrées-sorties. Cette opération est effectuée continûment et de manière asynchrone pendant la communication avec les autres modules;
- Processeur des données d'entrées et de sorties. Ce processus est responsable du filtrage et du codage des données d'entrée, la linéarisation et la graduation

des données analogiques. Pour les modules de sortie, cette opération est exécutée continûment et de manière asynchrone pendant la communication avec les autres modules, et le suivi des données de sorties;

- Établir la communication en base de temps alloué ou en système commande – réponse. Quand l'interface située dans l'ordinateur transmet une commande, les modules synchronisent leurs horloges qui vont servir par la suite à la distribution de la communication;
- Teste de diagnose. Après chaque opération d'entrée-sortie le processeur, situé dans le module, roule une séquence de testes pour vérifier l'intégrité du software et hardware.

Voici la liste des modules d'entrée/sortie qui sont utilisées dans la boucle au Fréon de l'IGN :

a) *Les modules d'entrées*

- Les modules d'entrées digitales : convertissent les signaux (potentiels électriques) produits par les détecteurs tels que les boutons, les commutateurs de limite, les fermetures du contact, les commutateurs de pression, etc., dans des données utilisables pour l'ordinateur. Cette conversion demande que le voltage d'entrée soit plus grand que le voltage de seuil pour un certain module et qu'il soit appliqué pendant une période plus longue que la période de filtrage déclarée au moment de la configuration du module. On utilise une période de filtrage pour éliminer les bruits. Les modules digitaux peuvent fonctionner en état de verrouilleur. On suppose que si le module est entré en état de déclenchement il ne va sortir de cet état qu'au moment où il reçoit une commande explicite de la part de l'ordinateur.
- Les modules d'entrée analogique : font la conversion des signaux reçus de la part des détecteurs en des signaux digitaux représentés sur 2 octets. Les signaux acceptés comme entrée sont dans divers domaines en fonction de la

version du module et des 13 différents types de thermocouples, thermomètres à résistance, capteurs électriques actifs de déplacements linéaires et jauges de déformation.

b) Les modules de sorties

- Les modules de sorties discrètes : sont responsables d'effectuer la conversion des données digitales reçues de l'ordinateur par l'intermédiaire de l'interface en niveaux de voltage utilisables par les appareils. Elles sont produites seulement avec 16 canaux et les canaux peuvent produire jusqu'à 2 A chacun. L'intensité maximale du courant total par module est de 20 A. En plus, chaque canal de sortie est protégé contre les courts circuits et contre les variations de voltage.
- Les modules de sorties analogiques : effectuent la conversion d'un nombre représenté sur 12 bits (mais envoyé comme un nombre positif sur 15 bits de 0 à 32760) dans un signal analogique pour pouvoir être utilisé par les dispositifs situés sur le champ. Les signaux de sortie sont respectivement situés entre 0 et 20 mA et entre 4 et 20 mA.

5.5 Les blocs d'alimentation

Ceux unités assurent l'alimentation électrique des modules d'entrées-sorties. L'alimentation s'effectue directement par l'intermédiaire du câble coaxial du réseau TransNet. Les sources de tension doivent produire une tension comprise entre 16 et 35 V CC. Dans notre cas, on utilise une source PSC-24 produisant 170 W. En plus, la source inclut des filtres et des composants pour assurer la diminution du bruit électrique.

5.6 Les fonctions du pilote informatique

Pour pouvoir communiquer avec le réseau à partir de l'ordinateur, on utilise un pilote fourni par la compagnie MTL I/O. En plus, ce pilote offre une application pour la configuration des modules pour effectuer des essais préliminaires. L'application de configuration doit être exécutée immédiatement après l'installation du pilote, avant de commencer à utiliser le réseau TransNet. Elle alloue des adresses pour les modules et permet de configurer chaque entrée ou sortie pour chaque module. Les possibilités de configuration de chaque canal et la fréquence des déterminations dépendent du type de module installé sur place. Ce système permet de gérer jusqu'à 120 modules simultanément. Pour les canaux digitaux on peut aussi établir un filtre en fonction du temps, qui permet de recevoir ou d'envoyer un signal au moins pendant la période établie du temps. Tous les réglages sont par la suite sauvegardés dans un fichier de configuration. L'application d'essais permet la visualisation rapide des déterminations reçues par le réseau.

Le « pilote » offre aussi la possibilité de communiquer avec le réseau à partir de différentes langages de programmation. Pour le langage C, le « pilote » offre des fichiers bibliothèques de commande qui sont appelés à partir des fichiers « headers ». Les fonctions écrites dans le langage C++ peuvent être appelées si les headers sont inclus dans le projet de travail. On va seulement présenter à titre d'exemple (dans le présent projet on a utilisé une autre modalité pour communiquer avec le réseau TransNet) quelques fonctions; les plus importantes sont les suivantes :

- *short system_start(short board, char *file, short mode).* Cette fonction initialise le dialogue avec la carte IIOP. Le premier paramètre représente l'identificateur de la carte d'interface dans l'ordinateur (dans notre cas il devrait toujours être égal à 1). Le deuxième paramètre est le nom du fichier dans lequel on désire écrire l'historique du

branchement de la carte. Le troisième paramètre est une constante qui spécifie l'état de la carte IIOP par la suite.

- *short mod_stat_get(short board, short module).* Cette fonction retourne une valeur qui représente l'état du module. Elle a comme paramètres l'adresse de la carte et l'adresse du module. Les deux adresses correspondent à la configuration effectuée initialement.

- *short read_a_chan_float(short board, short module, short canal, float *data).* Cette fonction lit la valeur d'un canal. Elle a comme paramètres l'adresse de la carte IIOP, l'adresse du module, le numéro du canal et un pointer vers une variable réelle dans laquelle on va recevoir la réponse.

- *void system_free(short board).* Cette fonction arrête le dialogue avec la carte IIOP.

Chapitre 6

L'application Paragon

Pour définir le système d'acquisition de données et de contrôle de la boucle thermique au Fréon, on a utilisé l'application Paragon produite par la compagnie Nematron (référence [13]). Cette application est spécialisée dans l'automatisation des processus industriels et permet de bâtir des ensembles composés de l'interface utilisateur, des applications de contrôle et des acquisitions de données (« Supervisory Control And Data Acquisition », SCADA) ayant des capacités permettant le contrôle des systèmes distribués. Ce système est utilisé pour une panoplie d'applications industrielles et de recherche ayant jusqu'à mille points d'entrées-sorties.

Les applications qui sont créées avec Paragon, ont à la base un partage des tâches entre clients et serveurs comme présente dans la figure 6.1. Elles peuvent offrir des multiples possibilités de connections vers des technologies diverses.

Dans la première étape, le dialogue entre les components crées avec l'application Paragon est assuré par une zone commune d'accès aux ressources (« Common Resource Access Layer », CRA). Si nécessaire, les données peuvent être utilisées par d'autres clients non créés avec Paragon, par l'intermédiaire de l'interface avec les clients extérieurs (« Client Objects Interface »).

Voici les serveurs de l'application Paragon :

1) Le gérant de l'application (« Application Manager », AM)
   L'AM est le noyau du Paragon, utilisé pour accéder à tous les sous-ensembles et pour démarrer les fonctions du moteur d'exécution. Il fournit également

l'accès de sécurité, la diagnose des sous-systèmes et assure le branchement de l'utilisateur à l'application.

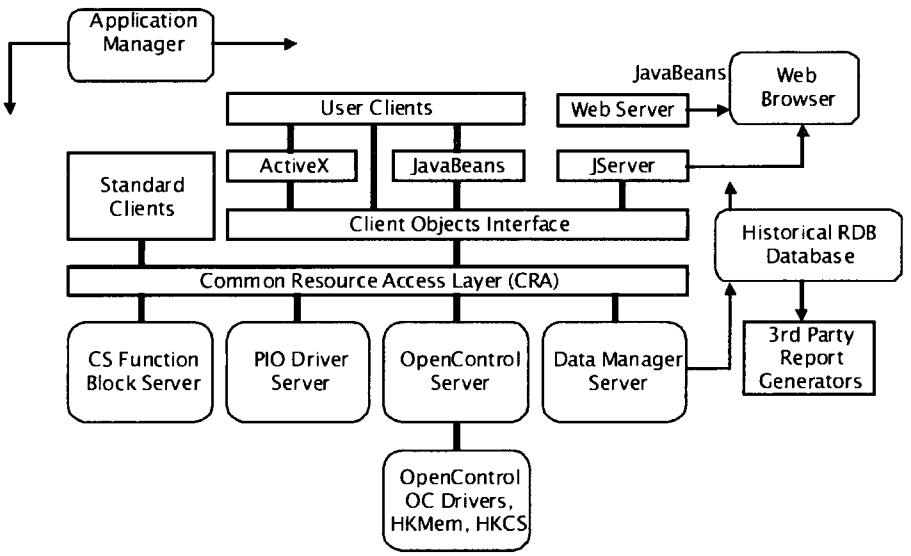

Figure 6.1 : Relations entre les composantes d'une installation Paragon (référence [12])

2) Le processus d'entrées-sorties (« Process Input Output », PIO)

Le PIO contrôle les « pilotes » pour acquérir des données à partir de l'envoi des changements envers les dispositifs externes. Ces dispositifs incluent les automates programmables industriels (« Programmable Logic Controller », PLC), les contrôleurs avec cycle unique (« Single Loop Controller », SLC), les entrées-sorties simples, etc. Il fournit également la conversion du signal en unités de mesure technologique et permet la linéarisation, de placer ou exécuter les alarmes et de rapporter une erreur. Le PIO peut travailler avec des pilotes multiples qui communiquent par les portes séries, les cartes de branchement, le protocole TCP/IP, etc. Des données sont montrées par des clients, typiquement sans traitement additionnel, et peuvent être consultées par réseau.

3) Le gérant des données (« Data Manager », DM)

Le DM est déclenché par des événements et gère les listes d'alarmes, les listes des événements, les journaux, les tendances et les données historiques. Il contient aussi des fonctions périodiques et des fonctions de compression utilisées pour bâtir de grandes collections de données. Les tendances, les données, les alarmes et les informations sur les événements peuvent être transportées automatiquement dans des bases de données relationnelles en ajoutant une instruction SQL (« Structured Query Language ») à la collection. Le transport peut être périodique ou peut être déclenché par un événement.

4) La stratégie continue (« Continuous Strategy », CS)

La CS est une application basée sur le système des blocs fonctionnels. Il utilise un moteur de calcul pour effectuer des fonctions mathématiques et logiques. Les applications de la CS sont établies graphiquement en se reliant au PIO pour les données d'entrées et de sorties. Le CS a un ensemble complet de fonctions comprenant : le calcul analogique, le calcul discret, la chaîne de caractère, et la conversion entre les différents types de données.

5) Contrôle Ouvert (« Open Control », OC)

OC est un moteur de calcul séquentiel à grande vitesse avec des « pilotes » très rapides. L'OC et ses pilotes roulent sur « Imagination Systems Hyperkernel ». De plus, l'OC est programmé en utilisant un organigramme interactif. Les variables sont localement disponibles pour les clients du Paragon et dans un réseau utilisant des conventions adéquates. Les pilotes du OC représentent une alternative pour réaliser l'interface avec les processus d'entrée-sortie avec les composantes clients et serveurs de Paragon.

De plus, l'application Paragon fournit des différents types de clients. Voici les types de clients que nous avons utilisés pour réaliser le système de contrôle de la boucle au Fréon :

1) Interface opérateur (« Operator Interface », OI)

   L'OI permet la création et le formatage des panels de commande sur ordinateur sans effectuer de programmation classique. Une interface intelligente permet de choisir les différents objets disponibles et de connecter leurs attributs à partir des serveurs. L'OI peut aussi effectuer des opérations sur le serveur comme régler des valeurs ou imposer des valeurs qui doivent être attendues par un contrôleur.

2) Interface pour ingénieur (« Engineering Interface », EI)

   Cette interface permet de faire le suivi de données fournies par les serveurs, sans construire d'autres fenêtres. On peut suivre toutes les valeurs des fonctions du moteur d'exécution ou d'un trend spécifié.

6. 1 Fichiers d'une application Paragon typique

L'élaboration d'un logiciel de contrôle qui utilise la technologie Paragon nécessite la gestion d'une série de fichiers. On peut à priori classer ces fichiers dans les catégories suivantes :

- Deux fichiers qui comprennent la description des entrées et de sorties générées par PIO. Les processus et les fonctions sont définis dans la section 6.2.1. Ces fichiers sont : le fichier bibliothèque (extension *pid*) qui contient tous les processus, ainsi que les fonctions d'entrée-sortie et le fichier de configuration (extension *pic*) qui est utilisé comme fichier de mise en marche pour le sous-ensemble courant. Il définit quels sont les processus exécutés quand on démarre l'application.

- Deux fichiers d'interface avec l'opérateur créés par le « *builder* » pour l'interface opérateur. Un fichier non compilé d'interface avec l'opérateur (extension *oil*) qui

est utilisé pour créer et modifier l'application et un fichier compilé de l'interface avec l'opérateur (extension *oid*) est employé pour exécuter l'application.

- Pour pouvoir configurer des fonctions spécifiques avec le gérant des données il est nécessaire de créer deux fichiers en utilisant le « *builder* » du DM : un fichier bibliothèque de commandes qui contient les processus (extension *dmd*) et un fichier de configuration (extension *dmc*).

- Trois fichiers représentant la stratégie continue créés par le « *builder* » de CS : la page graphique (extension *csb*) qui contient des graphiques utilisés pour créer la stratégie, un fichier de configuration (extension *csc*), un fichier bibliothèque de commandes (extension *csd*). De plus, on peut créer un fichier avec l'extension *csl* qui représente un format portable entre Windows et OS/2.

- Un fichier pour gérer les composantes de l'application peut être créé avec le gérant d'application. Il contient les réglages et les spécifications sur l'application courante.

6.2 Serveurs et clients de Paragon

Pour pouvoir accéder aux différents niveaux de l'arbre des éléments de l'application, Paragon utilise une convention qui peut s'écrire comme suit :

Sous-système. Processus. Fonction. Élément,

où :

- Le sous-système représente en général un serveur ou un client (i.e., PIO);
- Le processus est un groupe de fonctions, ainsi que des interconnexions de Paragon dans un sous-ensemble donné. Le type du processus est spécifique au pilot, utilisé pour communiquer avec le système de contrôle;

- La fonction représente une opération de base comme : calcul mathématique, algorithme de commande, collecte des données, communications avec un dispositif spécifique du système de contrôle et surveillance d'alarme. Des connexions sont établies entre les fonctions dans le même sous-système ou dans des sous-ensembles différents pour fournir l'interaction entre différentes parties de l'application et pour assurer la communication du flux des données;
- L'élément représente une caractéristique de la fonction (i.e., la valeur de sortie).

Il y a des éléments qui comprennent plusieurs valeurs (une structure similaire à un vecteur). Dans ce cas, l'indice de la valeur demandée sera spécifié entre accolades. Par exemple, pour effectuer la lecture d'un détecteur branché sur la deuxième position dans un module d'entrée analogique, les étapes à suivre dans PIO seront : choisir un processus spécifique au pilot utilisé, choisir la fonction correspondante au module d'entrée analogique. Dans la CS, l'élément sera la lecture du détecteur (**readIn**) et l'indice correspondant au branchement effectué sur le module sera 1 (parce que l'indice de la première position est toujours 0).

Parmi les serveurs, on trouve le processus d'entrée-sortie (PIO), le gérant des données (DM) et la stratégie continue (CS) et parmi les clients, on considère le constructeur de l'interface operateur.

6.2.1 Le processus d'entrée-sortie (« Process Input Output », PIO)

Celui-ci comprend deux fichiers de configuration qui ont des extensions différentes :

- Le fichier de configuration (*.PIC) du processus et la bibliothèque des fonctions. Ceci comprend une liste de différents processus. Il faut noter qu'un processus est caractérisé par son nom (qui peut avoir une longueur de 12

caractères), le nom de l'ordinateur situé en réseau où le processus évolue, le nom du fichier qui contient la bibliothèque des commandes. Pour chaque processus on doit spécifier une série de propriétés comme : *IIOP1_CFG* qui identifie le fichier qui contient les réglages du système de contrôle (ce fichier est produit au préalable par l'application de configuration du pilot de IIOP) et *ScanPeriods* qui représente les intervalles de temps possibles entre deux interactions successives avec les équipements du système. Pour le deuxième paramètre, on peut choisir jusqu'à 6 valeurs pour pouvoir par la suite, (pour chaque paramètre), spécifier un de ces intervalles entre deux opérations successives.

- La bibliothèque des fonctions (*.PID). Dans ce fichier, pour chaque processus doit être spécifiés le type de processus et les fonctions d'entrée-sortie que celui-ci regroupe. Le type du processus est spécifique pour chaque pilot (dans notre cas MTLProcess). Une fonction d'entrée-sortie représente un groupe d'appareils du même type qui sont branchés sur le même module. Les différents types de fonctions possibles sont : entrée digitale, entrée analogique, sortie digitale, sortie analogique, entrée digitale pulsation. Les paramètres de la fonction dépendent du type de celle-ci. On différencie quand même des paramètres qui sont les mêmes pour tous les types de fonctions. Cela est présenté dans le tableau 6.1. Les fonctions d'entrée digitale, ainsi que les fonctions d'entrée et de sortie analogique partagent les mêmes types de paramètres. Le tableau 6.2 contient la liste de ces paramètres.

  Pour les fonctions de sortie digitale, les paramètres possibles sont moins nombreux (*AlrmPriority, AlrmInhibit, ScanPeriod*, etc.).

Tableau 6.1 Les paramètres communs pour les fonctions de PIO.

Nom du paramètre	Description du paramètre
NumChan	Est le numéro des canaux dans la séquence des canaux suivis.
PortAddr	Est un numéro entier qui représente le numéro de la carte dans l'application de configuration. Dans notre cas ce numéro est 1.
ModuleAddr	Représente le numéro du module qui est branché dans le réseau TransNet. Cette caractéristique est aussi établie par l'application de configuration de la carte.
StartChan	Représente le numéro de branchement pour le premier canal de la séquence des instruments de mesure ou contrôle, sur le module de branchement.
DataType	Le type de données reçues ou envoyées.

Tableau 6.2 Les paramètres communs pour les fonctions d'entrée digitale et les fonctions d'entrée/sortie analogique de PIO

Nom du paramètre	Description du paramètre
Description	Une chaîne de caractères d'une longueur maximale de 28 qui contient la description de la fonction. Cette description est envoyée dans les messages d'alarme.
LoRange	Ceci représente la valeur minimale du domaine de mesure qui est envoyée aux autres sous-systèmes par PIO.
HiRange	Ceci représente la valeur maximale du domaine de mesure qui est envoyée aux autres sous-systèmes par PIO.
LoRow	Ceci représente la valeur minimale envoyée par PIO à l'équipement quand il doit envoyer l'équivalent du minimum du domaine.

Nom du paramètre	Description du paramètre
HiRow	Ceci représente la valeur maximale envoyée par PIO à l'équipement quand il doit envoyer l'équivalent du maximum du domaine.
LoAlrmVal	Ceci représente la valeur inférieure pour laquelle le système doit envoyer le signal d'alarme.
HiAlrmVal	Ceci représente la valeur supérieure pour laquelle le système doit envoyer le signal d'alarme.
LoLoAlrmVal	Ceci représente le minimum du domaine d'alarme inférieur.
HiHiAlrmVal	Ceci représente le maximum du domaine d'alarme supérieur.
AlrmPriority	Ceci représente la priorité de l'alarme. La valeur peut varier de 0 à 99 (la plus élevée est 99).
AlrmInhibit	Ceci établit si l'alarme est activée ou non.
ScanPeriod	Ceci représente l'intervalle de temps entre deux lectures successives.

6.2.2 Le gérant des données (« Data Manager », DM)

Les spécifications des processus qui sont chargées de la sauvegarde des données, se trouve dans deux fichiers :

- Le fichier de configuration (*.DMC). La structure est similaire à la structure du fichier de configuration du processus d'entrée-sortie. Ce fichier comprend une liste de processus. Un processus DM, comme un processus PIO, est caractérisé en partie par : son nom, le nom de l'ordinateur et le nom du fichier qui contient la bibliothèque des commandes. De plus, d'autres caractéristiques sont disponibles : *StartOnLoad* – établit si le processus

commence automatiquement à l'initialisation ou non, *ScanPeriod1-ScanPeriod6* – sont les périodes entre deux actions successives.

- La bibliothèque des fonctions (*.dmd). Dans ce fichier on va spécifier pour chaque processus ses fonctions de sortie dépendant du type du processus. Le type est donné par le format du message qu'on veut envoyer vers l'extérieur et l'action possible à exécuter par la suite. Le tableau 6.3 contient quelques exemples de types de fonctions. Les quatre premières fonctions présentées dans le tableau 6.3 (*Trend, History, MultiHistory* et *DbandHistory*) ont une série de paramètres similaires parce que pour toutes le domaine d'utilisation est la manipulation des données mesurées. Ces paramètres sont présentés dans le tableau 6.4.

Tableau 6.3 Exemples de types de fonctions DM.

Nom du paramètre	Description du paramètre
Trend	Ce type de fonction permet de retenir toutes les valeurs d'un paramètre pendant une session de travail du DM. Les données ne sont pas stockées sur le disque dur.
History	Ce type de fonction permet de retenir les valeurs d'un paramètre. La différence entre ce processus et ***Trend*** est que les données peuvent être sauvegardées dans un fichier contenu sur le disque dur.
MultiHistory	Est une généralisation pour le type ***History*** qui permet de retenir plusieurs paramètres qui par la suite sont sauvegardés dans un fichier.
DbandHistory	Permet de retenir des valeurs pour un seul paramètre qui seront par la suite affichées dans un élément de l'interface de type ***Plot***. Les données peuvent aussi être sauvegardées dans un fichier ou exportées vers une base de données.

Nom du paramètre	Description du paramètre
Alarm	Lorsqu'un événement qui produit l'alarme a lieu, l'application envoie un message vers une partie de l'application chargée de cela, qui maintient à jour un fichier des événements de ce type. En plus, dans l'interface on a un type d'élément qui peut afficher les alarmes au moment de l'apparition.
MsgOut	Ceci est une fonction qui permet d'envoyer aux différentes fonctions du DM ou CS des messages périodiques.

Tableau 6.4 Paramètres des fonctions *Trend, History, MultiHistory* et *DbandHistory* de DM

Nom du paramètre	Description du paramètre
Description	Ceci est une chaîne de caractères pouvant aller jusqu'à 128 caractères et qui comprend une description de la fonction.
SrcConnName	Contient le nom de l'élément suivi. Une caractéristique de Paragon est la convention arborescente de la nomination des différents systèmes.
ScanPeriod	Ceci représente la période de temps entre deux déterminations ou commandes successives.
SmplSize	Ceci représente le nombre de déterminations effectuées avant l'arrêt automatique de l'exécution de la fonction.
DfltCmd	Ceci représente la commande qui est mise en exécution quand la fonction reçoit le message « Execute ».

En ce qui concerne le paramètre *DfltCmd* du tableau 6.4, il spécifie la modalité de sauvegarde des données. Sa structure est :

<commande> [<option+parameters>] SQL:"<instruction SQL>"

<commande> peut avoir une de les trois valeurs suivantes :

- *Copy* : cette commande effectue la copie des données retenues par la fonction vers le disque dur (dans un fichier) ou vers une base de données. Ses options sont : *from* (copie l'information à partir du moment indiqué), *to* (copie l'information jusqu'au moment indiqué), *includesFrom* (copie l'information à partir du moment indiqué plus le premier set donné avant) et *includesTo* (copie l'information jusqu'au moment indiqué plus la donnée suivante).
- *CopySince* : copie la nouvelle information acquise par la fonction de DM à partir de la dernière exécution du CopySince. Un marqueur est placé dans le fichier pour identifier la dernière donnée envoyée. Quand un autre CopySince est exécuté, il envoie des données commençant par la donnée après le marqueur. Parce que l'action de marquage est effectuée dans le fichier, elle suit le même principe même si DM est redémarré. Les options qui peuvent être utilisées sont *to* et *includesTo*.
- *MoveSince* : La commande de MoveSince déplace toutes les données à partir des plus anciennes disponibles. Elle est similaire à la commande de CopySince sauf qu'après, l'information choisie ait été déplacée vers le fichier, les données sont effacées de la fonction du DM. On commence par les plus anciennes pour s'assurer qu'il n'y aura aucun trou dans la collecte de données. Les options qui peuvent être utilisées sont *to* et *includesTo*.

Le langage utilisé pour établir la modalité selon laquelle s'effectue l'export des données est une adaptation simple du standard SQL. La seule commande utilisée est *Insert*. La structure de la commande est la suivante :

INSERT INTO <nom_fichier> VALUES
<succesion_noms_generiques_colonnes>

Le nom du fichier doit respecter toutes les règles exigées par le système d'exploitation. Les noms génériques des colonnes sont : « *:0* » pour le temps/date, « *:1* » pour un indice de qualité de la donnée qui précise si la valeur est correcte, « *:2* » pour la donnée elle-même et « *:all* » pour toutes les caractéristiques.

6.2.3 La stratégie continue (« Continuous Strategy », CS)

Cette partie du Paragon assure les transformations mathématiques, logiques et de contrôle du flux de données. Elle comprend un serveur chargé de fonctions mathématiques, booléennes, de commande directe, de surveillance et de manipulation de chaînes de caractères. Ces fonctions produisent de l'information à partir des valeurs obtenues, dans la plus-parte des cas, du PIO (i.e., des déterminations provenant des thermocouples branchées sur des modules) qui peuvent, après être manipulées, produire des données qui seront retournées vers le même serveur (i.e., valeur de contrôle pour commander l'ouverture d'une vanne). Un environnement graphique interactif est fourni pour permettre la construction des stratégies. Ca donne la possibilité de faire le suivi du flux des données qui peut être aussi distribué par plusieurs couches. Les couches sont initiées à l'intérieur des blocs spécialement prévues pour ça.

Les structures utilisées pour construire des stratégies sont des graphes orientés qui contiennent dans les nœuds des opérations (d'entrée ou de sortie des données ou des alarmes, opérations mathématiques ou booléennes, des commandes ou des manipulations de chaînes de caractères). Les blocs d'entrée ou de sortie sont nommés terminateurs parce que le flux de données commence ou finit à cet endroit. Les autres blocs sont nommés fonctions. Les paramètres pour les nœuds de type fonction sont les résultats des autres opérations de calcul (fonctions) ou d'entrée qui sont liées par le nœud courant en utilisant des arcs (nommés connexions).

Les connexions forment donc des trajets pour le flux de données à partir des fonctions d'entrée du PIO spécifié dans la couche initiale, en passant par des fonctions de la stratégie continue et arrivant vers un élément qui sera affiché ou sauvegardé ou, si on utilise un contrôleur, à une fonction de sortie du même PIO.

Obligatoirement, dans la couche initiale deux blocs seront spécifiés : un bloc du type stratégie continue (CS) relié à un bloc du type processus d'entrée-sortie (PIO). À l'intérieur du bloc de stratégie continue on peut implémenter la stratégie elle-même.

Chaque type de bloc est identifié par un nom (i.e., bloque d'entrée analogique – AIN – « Analog Input »). Chaque bloc possède une série de paramètres qui détermine entre autre le traitement appliqué aux données. Ainsi, on peut lire des paramètres de ce type de bloc à partir des systèmes d'application Paragon. Cela nous permet par exemple de lire les valeurs de données avec les éléments d'interface graphique ou d'établir des paramètres pour le bloc également à partir des éléments d'interface. En fonction de leur type, le bloc possède des *connexions vers* lui et des *connexions de* lui.

Les caractéristiques communes pour tous les blocs à part les blocs terminateurs sont les suivantes :

- Nom (« NAME ») : chaîne de caractères qui représente le nom générique du bloc;
- Description : chaîne de caractères qui donne une brève description du bloc ;
- Vitesse d'exécution (« Scan Rate ») : l'usager peut choisir dans une liste de valeurs qui a été établie au préalable.

Voici les principaux types de blocs qu'on trouve dans une application :

1. Bloc terminateur : spécifie une liaison de ou vers une fonction du processus PIO;

2. Bloc d'entrée ou de sortie : est utilisé pour effectuer les réglages sur le flux de données qui provient d'un bloc terminateur. En fonction des types de données traités, les blocs sont soit analogiques, soit digitales.
   L'identification des blocs analogiques se fait par leurs noms : AIN (« Analog Input » - entée analogique) et AOUT (« Analog Output » - sortie analogique). Ce type de bloc permet d'effectuer la mise en échelle des valeurs d'entrée selon les trois possibilités suivantes linéaire, racine carrée ou division par 10, d'appliquer des alarmes et des filtres digitaux. Le bloc de sortie permet même de comparer la valeur de sortie avec des valeurs et de déclencher une alarme s'il y a des différences dépassant un certain seuil;

   Les blocs digitales sont identifiées comme : DIN (« Digital Input » - entée digitale) et DOUT (« Digital Output » - sortie digitale).

   De plus on doit indiquer le type de connexion. *Connexions vers le bloc* : In (le flux des données à l'entrée), HiAlrmVal (valeur de l'alarme supérieure), LoAlrmVal (valeur de l'alarme inférieure), HiLimit (limite supérieure utilisée pour la dilatation des données), LoLimit (limite inférieure également utilisée pour la dilatation des données), AlrmInhibit (contrôle d'activation de l'alarme), etc. *Connexions de bloc* : Out (le résultat de l'application des opérations spécifiées dans le bloc), HiAlrmStat (l'état de l'alarme supérieure - valeur digitale), LoAlrmStat (état de l'alarme inférieure – valeur digitale), etc.

3. Le bloc de calcul (« EXPR » - expression) : permet d'effectuer des opérations mathématiques pouvant aller jusqu'à quatre données en entrée. Le bloc est d'un format limité et ne permet que d'effectuer des opérations ayant la forme suivante :

$$OUT = f1\ (I1)\ \&1\ f2\ (I2)\ \&2\ f3\ (I3)\ \&3\ f4\ (I4)\ \ldots, \qquad (6.1)$$

où :

I1-4	flux des données d'entrée (connections)
&1-4	un de les opérateurs : +, -, *, **, /
f1-4	une des fonctions suivantes : sinh, cosh, tanh, exp, ln, log, abs, sqrt et d (qui est la dérivée d'une valeur, par exemple la vitesse de changement en unités/seconde).

Il faut noter que les arguments des fonctions trigonométriques doivent être exprimés en radians (et non en degrés).

À cause du fait que dans les calculs ne puissent pas utiliser directement des valeurs numériques dans l'expression, le bloc de calcul permet à l'usager de spécifier jusqu'à 4 valeurs qui seront identifiées par K1-4. Ces constantes peuvent être utilisées dans l'expression. Les caractéristiques pour ce type de bloc sont :

- Maximum et minimum du domaine (High & Low scale); ceci représente des valeurs maximales et minimales dans les unités de mesure spécifiées qui peuvent être produites par le bloc (valeurs permises de -10000000 à 10000000). Ces valeurs ne peuvent pas être établies à partir d'un flux externe;
- Limites maximale et minimale (High & Low limit) : ceci sont les limites pour le résultat du calcul. Les valeurs des ces limites doivent être à l'intérieur du domaine. Si le résultat du calcul sort du domaine, le bloc va retourner la valeur maximale ou minimale;
- Numéros d'entrées : ceci est le numéro des flux de données qui participeront au calcul. Les valeurs permises sont des entiers compris entre 1 et 4;

- Première entrée (Input1) : ceci peut prendre la valeur I1 s'il est un flux externe ou une valeur fixe (comprise entre -10000000 et 10000000);
- Deuxième entrée (Input2) : ceci respecte les mêmes conditions qu'Input1.
- Troisième et quatrième entrée (Input3 et Input4) : doivent obligatoirement être des flux de données externes;
- Résultat (OUT) : est un champ de saisie où les expressions de calcul seront introduites en respectant le format présenté au début.
- K1-4 : les valeurs des constantes;

Connexions vers le bloc : In1-In4 (les flux de données d'entrée), HiLimit (limite supérieure utilisée pour la dilatation des données), LoLimit (limite inférieure utilisée aussi pour la mise en échelle des données), etc.
Connexions de bloc : Out (le résultat de l'application des opérations spécifiées dans le bloc), ManualStart (mode de départ manuel ou automatique – contrôle digital), etc.

4. Bloc de contrôle proportionnel, intégral et dérivatif (PID) : constitue le système de commande le plus complexe dans Paragon. Il obéit à l'équation suivante :

$$\begin{aligned} OUT = {} & (100 / PROPORTIONAL) \cdot (ERROR - MEAS \cdot Ds) \cdot (1 + 1/Is) \cdot \\ & \cdot (OUT\ RANGE / MEAS\ RANGE) + (BIAS \cdot KBIAS), \end{aligned} \tag{6.2}$$

où :

- MEAS : valeur mesurée du paramètre à contrôler
- SETP : valeur cible (« Set Point »)
- ERROR : différence entre la valeur actuelle et la valeur cible (SETP – MEAS)
- MEAS RANGE : intervalle de mesure du bloc ascendant, fournissant la mesure

- OUT RANGE : longueur du domaine de sortie du bloc PID
- 100 / PROPORTIONAL : gain proportionnel
- D : valeur du terme dérivée en minutes
- I : valeur de la constante de temps intégrale en minutes par répétition
- S : operateur de Laplace
- BIAS : signal externe qui peut intervenir dans le calcul
- KBIAS : facteur de multiplication du signal externe

Les caractéristiques de ce type de bloc sont :

- Maximum et minimum du domaine (High & Low scale) : représente des valeurs maximales et minimales dans les unités de mesure spécifiées qui peuvent être produites par le bloc;
- Limites maximale et minimale (High & Low limit) : sont les limites pour le résultat du calcul;
- Les valeurs des constantes : Proportionnel, Intégral, Dérivative, Bias et Kbias;
- Les valeurs d'alarme, etc.

Connexions vers le bloc : In (le flux des données à l'entrée), Setp (la valeur cible du contrôleur), HiAlrmVal (la valeur de l'alarme supérieure), LoAlrmVal (la valeur de l'alarme inférieure), HiLimit (la limite supérieure utilisée pour la dilatation des données), LoLimit (la limite inférieure également utilisée pour la dilatation des données), AlrmInhibit (le contrôle de l'activation de l'alarme), PBand (la valeur du paramètre du gain proportionnel, Derivative (la valeur du terme dérivée en minutes), Integral (la constante de temps intégrale en minutes par répétition), etc.
Connexions de bloc : Out (le résultat de l'application des opérations spécifiées dans le bloc), HiAlrmStat (l'état de l'alarme supérieure - valeur digitale),

LoAlrmStat (état de l'alarme inférieure – valeur digitale), ManualStart (mode de départ manuel ou automatique – contrôle digital), etc.

5. Bloc d'alarme : est un bloc qui produit un résultat digital si le signal d'entrée sort du domaine établi. Les caractéristiques que nous pouvons établir pour ce type de bloc sont les suivantes :

   - Valeur d'alarme supérieure (High alarm);
   - Valeur d'alarme inférieure (Low alarm);
   - Bande morte pour l'alarme (Alarm Deadband) qui représente la différence entre la valeur pour laquelle le système entre en état d'alarme et la valeur pour laquelle le système revient en état normale;
   - Valeur d'alarme pour le changement rapide (Rate alarm) : est une valeur minimale pour la différence entre deux données successives pour laquelle le bloc entre en état d'alarme pour ce type d'événement;
   - Valeur cible (Setpoint) : est une valeur qui sera utilisée comme paramètre pour l'alarme de déviation;
   - Valeur d'alarme de déviation (Deviation Alarm) : est la valeur minimale pour la différence entre la valeur cible et la valeur mesurée pour laquelle le bloc entre en état d'alarme pour ce type;

   Connexions vers le bloc : In (le flux des données d'entrée), Setp (la valeur cible du contrôleur), HiAlrmVal (valeur de l'alarme supérieure), LoAlrmVal (valeur de l'alarme inférieure), AlrmInhibit (contrôle d'activation de l'alarme) Setp (valeur cible), etc.
   Connexions de bloc : Out (le résultat de l'application des opérations spécifiées dans le bloc), HiAlrmStat (l'état de l'alarme supérieure - valeur digitale), LoAlrmStat (état de l'alarme inférieure – valeur digitale), DevnAlrmStat (état de l'alarme de déviation), RateAlrmStat (état de l'alarme pour le changement

rapide), ManualStart (mode de départ manuel ou automatique – contrôle digital), etc.

6. Bloc commutateur (SWCH) : est une fonction qui permet de sélectionner parmi les deux flux de données d'entrée lequel va passer comme flux de sortie. Les caractéristiques qu'on peut établir pour ce type de bloc sont les suivantes :

   - Maximum et minimum du domaine (High & Low scale) : représente des valeurs maximales et minimales dans les unités de mesure spécifiées qui peuvent être produites par le bloc;
   - Limites maximale et minimale (High & Low limit) : sont les limites pour le résultat du calcul. Les valeurs de ces limites doivent être à l'intérieur du domaine. Si le résultat du calcul sort du domaine, le bloc va retourner la valeur maximale ou minimale;
   - Premier flux d'entrée (Input1) : il peut être un flux de données extérieur ou on peut fixer une valeur;
   - Premier flux d'entrée (Input1) : il peut aussi être un flux des données extérieur ou on peut fixer une valeur;
   - Commutateur (Switch) : représente le flux de données sélectionné.

   Connexions vers le bloc : In1-2 (les flux de données d'entrée), HiLimit (limite supérieure utilisée pour la dilatation des données), LoLimit (limite inférieure également utilisée pour la dilatation des données), Switch (l'index du flux sélectionné), etc.
   Connexions de bloc : Out (le résultat de l'application des opérations spécifiées dans le bloc), Switch (l'index du flux de données sélectionné), etc.

7. Bloc mémoire pour la chaîne de caractères (String Keyboard – SKBD) : contient une chaîne de caractères.
   En plus des caractéristiques habituelles il a :

   - La chaîne de caractères (Out);
   - La longueur maximale de la chaîne de caractères (String Length).

   Connexions vers le bloc : Feedback (les flux de données d'entrée).
   Connexions de bloc : StrOut (le résultat de l'application des opérations spécifiées dans le bloc), NewStr (bit qui indique si une nouvelle chaîne a été générée, etc.).

8. Bloc générateur pour des valeurs digitales (Digital Keyboard – DKBD). Caractéristiques :

   - Valeur initiale (Initial Output);
   - Initialisation (Initialise) : Réalise l'initialisation du flux avec la valeur d'entrée;

   Connexions de bloc : Out (la valeur la plus récente du flux des données de sortie), Inverse (la valeur inversée la plus récente du flux de données de sortie), etc.
9. Bloc constructeur de chaînes de caractères (String Builder – SBLD) : est un bloc qui permet d'apposer des chaînes de caractères. Caractéristiques :

   - Nombre d'entrées (Number of inputs) : spécifie combien de connexions de données d'entrée sont acceptées. La valeur maximale est de 8;

- Entrées actives (Active inputs) : Spécifie quelle combinaison de changements parvenus dans les flux des données d'entrée va déclencher l'exécution du bloc;
- Le type actif (Active Type) : cette option est reliée à l'option antérieure et spécifie si le changement dans un seul flux détermine l'exécution du bloc ou si le changement de tous les flux sélectionnés va le faire;
- Déclenchement externe (External trigger) : spécifie si l'exécution du bloc va être décidée par une connexion digitale ou de manière automatique en respectant les options antérieures;
- Format de sortie : est la méthode de construction du flux de sortie. Les différents flux d'entrée sont identifiés de S1 jusqu'à S8;
- La longueur maximale de la chaîne de caractère de sortie (String Length) : valeur maximale 512;

Connexions vers le bloc : In01-08 (les flux de données d'entrée), Trigger (le déclencheur), etc.
Connexions de bloc : StrOut (le résultat de l'application des opérations spécifiées dans le bloc), NewStr (Valeur digitale qui spécifie si une nouvelle chaîne a été produite ou non), etc.

6.2.4 Le constructeur d'interface d'operateur (OI)

L'interface opérateur permet de :

1. Afficher des graphiques de processus animés;
2. Afficher des données provenant des serveurs de Paragon;
3. Saisir des données dans des boîtes de texte;
4. Afficher des graphiques de tendances;
5. Afficher les journaux d'alarme;

6. Effectuer des commandes par l'intermédiaire des éléments comme : boîte de liste, bouton radio, boutons et glisseurs;
7. Afficher des images;
8. Intégrer et afficher d'applications de tiers et multimédia.

Les objets peuvent être sélectionnés à partir d'une palette ou directement depuis le menu. Les objets d'interface disponibles sont :

- Arc (arc de cercle);
- Bitmap (image);
- Check Box (case à cocher);
- Data Display (étiquette pour afficher des données);
- Data Entry (champ de saisie des données);
- Ellipse;
- Event (événement);
- List Box (liste);
- Plot (diagramme pour suivre l'évolution de différentes données);
- Polygon (polygone);
- Polyline (ligne multiple);
- Push Button (bouton);
- Radio Button (bouton radio);
- Slider (barre de déroulage);
- Text (étiquette statique);
- Window (fenêtre).

On va seulement présenter les éléments qu'on a utilisés dans le système de contrôle de la boucle au Fréon :

1. Fenêtre

Les caractéristiques des fenêtres sont :

- La position et les dimensions. Pour permettre un arrangement relativement facile, l'unité de mesure utilisée est la hauteur de l'écran;
- L'existence ou non de certaines caractéristiques graphiques ou de comportement comme : la barre de titre, le bouton de maximisation, savoir si la fenêtre est redimensionnable, dans quel cas on pourra changer la position, l'existence du menu de contexte de système, si la fenêtre est affichée en plein écran ou non, si la fenêtre va être au début maximisée ou non, etc.;
- Le titre de la fenêtre;
- La couleur de fond;
- Période entre deux mises à jour successives;

Les fenêtres supportent aussi une série d'événements. Pour implémenter un événement les étapes à suivre sont : établir le type d'événement et par la suite la succession d'actions à effectuer. On peut choisir parmi une des actions suivantes : fermer la fenêtre, fermer le sous-système OI, ouvrir une fenêtre spécialisée dans l'affichage des évolutions de données, ouvrir une autre fenêtre qui couvre la fenêtre actuelle, ouvrir une autre fenêtre qui ne couvre pas la fenêtre actuelle, rouvrir la dernière fenêtre fermée, remplacer toutes les fenêtres par une nouvelle fenêtre, remplacer la fenêtre actuelle par une nouvelle fenêtre, exécuter un programme, établir une valeur pour un paramètre appartenant au sous-système de Paragon, etc. Les types d'événements pour la fenêtre sont classés en trois catégories :

a) Relatives au clavier :
   - Appui ou dégagement d'une touche.

b) Événements du cycle de vie de la fenêtre :
   - Ouverture ou fermeture de la fenêtre.

c) Actions relatives aux changements sur les flux des données :
   - Tout changement;
   - Changement d'un flux de données réel (si la valeur devient plus grande ou plus petite qu'une valeur de référence);
   - Changement d'un flux des données digital (si la valeur devient égale à une valeur de référence);
   - Changement d'un flux de données de type chaîne de caractères (si la valeur devient égale à une valeur de référence).

Pour tous les objets suivants d'interface, la position et les dimensions, on peut seulement les établir par l'intermédiaire du constructeur graphique. Donc on ne peut pas les spécifier par des valeurs numériques.

2. *Boîte de saisie de données*

   Caractéristiques :

   - Le format de la donnée affichée (longueur et éventuellement nombre des décimales);
   - La connexion : l'attribut d'un objet d'un sous-système relié à la valeur affichée. C'est la caractéristique la plus importante;
   - Le flux digital ou l'attribut qui établit si l'objet de l'interface est accessible ou non;
   - Les limites pour les données introduites;
   - Les limites pour la dilatation des données. Cela permet d'afficher des données transformées à partir du flux connecté. Les données introduites vont subir l'opération inverse;
   - Les couleurs et la police à utiliser.

3. *Bouton radio*

Les boutons radio doivent être groupés en utilisant la commande Make Composite et en choisissant par la suite le bouton qui sera initialisé en état sélectionné.
Caractéristiques :

- Étiquette;
- La connexion : l'attribut d'un objet d'un sous-système relié à la valeur affichée. C'est la caractéristique la plus importante;
- Le flux digital ou l'attribut qui établit si l'objet d'interface est accessible ou non;
- Les couleurs et la police à utiliser;
- Les actions, quand le bouton est coché. La liste des actions est la même pour la fenêtre;
- Les actions quand un autre bouton du groupe est coché. On a toujours la même liste d'actions.

4. *Bouton*

Les caractéristiques des boutons sont :

- Étiquette;
- La connexion : l'attribut d'un objet d'un sous-système relié à la valeur affichée. C'est la caractéristique la plus importante;
- Le flux digital ou l'attribut qui établit si l'objet d'interface est accessible ou non;
- Les couleurs et la police à utiliser;
- Les actions quand le bouton est appuyé. La liste des actions est la même pour la fenêtre.

5. *Diagramme*

On peut visualiser jusqu'à 8 flux de données dans le même diagramme si son type est un diagramme d'évolution par rapport au temps ou jusqu'à 4 fonctions si on représente des diagrammes X-Y (flux de données en fonction d'un autre flux). On ne peut représenter que des fonctions du DM. La plus utilisée est la fonction du type *Trend*. Les caractéristiques des diagrammes sont :

- Type de diagramme;
- Description;
- Police;
- Couleurs;
- Les réglages de l'aire de représentation comme : l'intervalle de temps de représentation des données, les réglages pour la grille, les couleurs pour la surface et pour la grille etc.;
- Les réglages pour les traceurs comme: la fonction du DM qui est représentée, les couleurs utilisées, les limites de la surface de représentation, etc.

Chapitre 7

L'application pour contrôler la boucle au Fréon

7.1 L'installation et la configuration des programmes préalables

Après l'installation du pilote informatique, une étape importante est de vérifier et de valider les connexions des capteurs avec les modules. Cela est réalisé en utilisant l'application fournie avec le pilote informatique nommée « Setup ». On peut attribuer, pour mieux reconnaître les connexions, des noms génériques aux modules et aux capteurs branchés. On peut aussi faire varier le domaine de mesure. Comme on peut observer dans la figure 7.1, il y a trois modules : un module digital de sortie, un module analogique de sortie (au maximum 8 commandes de dispositifs à contrôler) et un module analogique d'entrée (au maximum 16 capteurs à brancher).

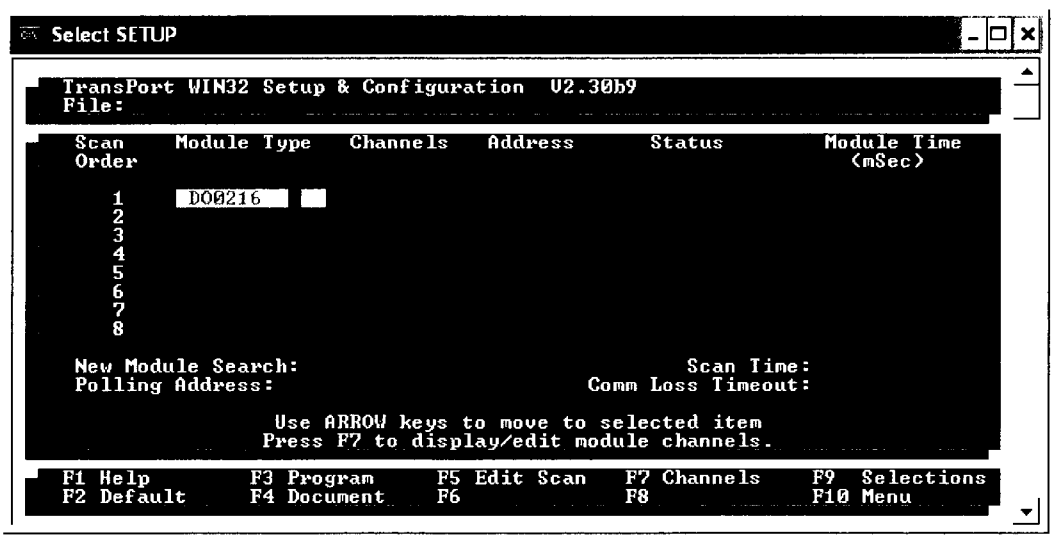

Figure 7.1 : Interface de configuration des modules de branchement du réseau TransNet.

Pour le cas du premier module, on a branché une seule sortie qui représente l'alarme qui arrête le chauffage de la section d'essais; ceci est illustré dans la figure 7.2.

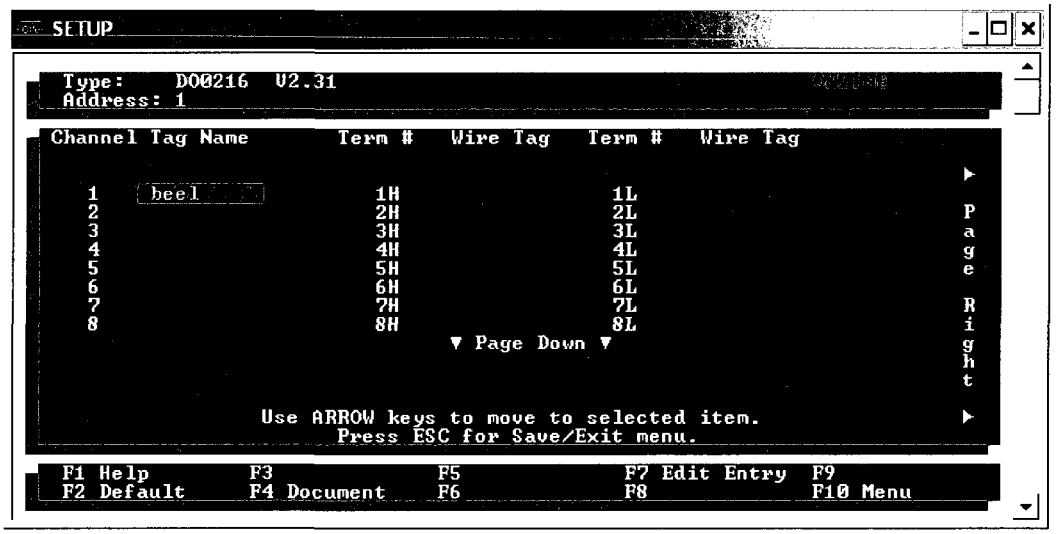

Figure 7.2 : La configuration du module de sortie digitale.

Le tableau 7.1 résume les différents capteurs qui sont branchés au module d'entrée analogique. Un imprimé-écran de l'application de l'application « *Setup* » du pilot informatique de l'IIOP est montré dans la figure 7.3.

Tableau 7.1 Les capteurs branchés sur le module d'entrée analogique.

Entrée no.	Type de capteur
1.	RTD, température à l'entrée du condenseur
2.	RTD, température à la sortie de l'échangeur de chaleur
3.	RTD, température de l'écoulement à la sortie du préchauffeur
4.	Débitmètre à vortex
5.	Capteur de pression, pression dans le condenseur
6.	Capteur de pression différentielle, différence de niveau dans le ballon condenseur
7.	RTD, température dans le condenseur
8.	RTD, température ambiante
9.	RTD, température à l'entrée de la pompe

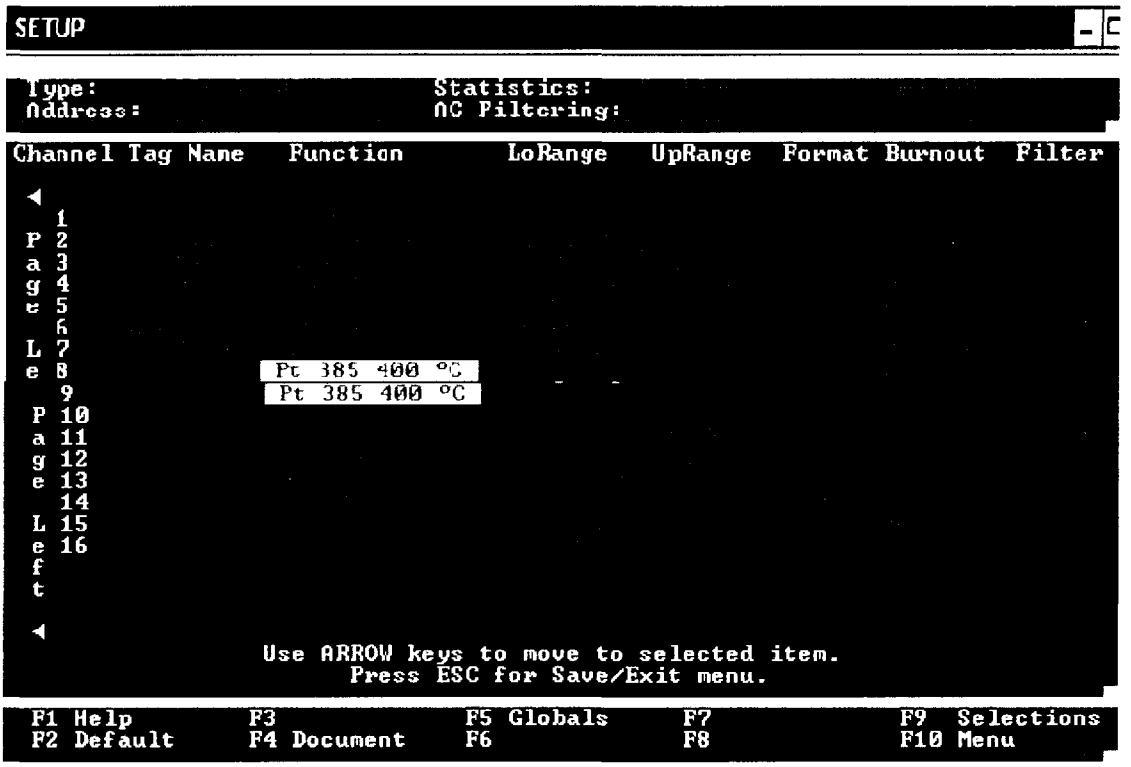

Figure 7.3 : La configuration du module d'entrée analogique.

Le tableau 7.2 résume les dispositifs analogiques contrôlés qui sont branchés au module sortie analogique. Un imprimé-écran de l'application de l'application *Setup* du pilot informatique de l'IIOP est montré dans la figure 7.4.

Tableau 7.2 Les capteurs branchés sur le module de sortie analogique.

Entrée no.	Type d'instrument
1.	Vanne de mélange, contrôle la température de la douche dans le condenseur
2.	Vanne de mélange, contrôle la température du mélange entre le liquide provenant de la sortie de l'échangeur de chaleur et de la sortie de la pompe

Entrée no.	Type d'instrument
3.	Niveau de puissance dans le préchauffeur, contrôle la température à l'entrée de la section d'essais
4.	Vanne, contrôle le débit par la boucle
5.	Vanne, contrôle le débit dans le gicleur situé le ballon condenseur pour établir la pression à la sortie de la section d'essais

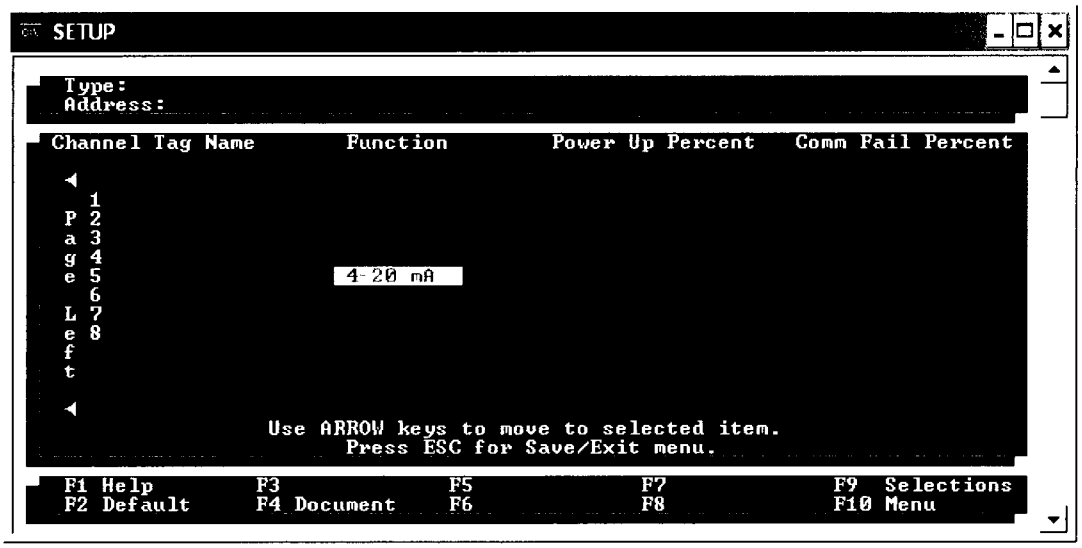

Figure 7.4 : La configuration du module de sortie analogique.

Tous les réglages sont sauvegardés automatiquement par l'application de configuration dans le fichier TSCS.cfg situé dans le répertoire correspondant au pilote informatique. Pour la configuration de l'application Paragon, elle s'est faite d'une manière standard, la seule chose spécifique étant le choix des pilotes informatiques pour notre carte du type « MTL_IO driver » réalisée dans le « wizard » de configuration.

7.2 La configuration des comptes d'usager nécessaires pour accéder au Paragon

Une chose importante est aussi le choix du nom d'usager et du mot de passe. Paragon offre des possibilités de créer des groupes d'usagers ayant des droits ou non d'accéder aux ressources de l'application. Parce qu'on a considéré que la sécurité n'est pas une priorité (l'ordinateur sur lequel on fait rouler l'application est protégée par son emplacement dans le laboratoire de Thermo-hydraulique), on a simplement utilisé le nom d'usager offert par défaut (i.e., « guest ») avec tous les privilèges. Le mot de passe est vide.

7.3 La déclaration et l'implémentation des processus et des fonctions qui gèrent les données d'entrée et de sortie (PIO)

Dans le serveur PIO le nom des fichiers correspondants est « *pio* ». On a déclaré un seul processus nommé « pioprocess ». Le type (la classe) du processus est MTLProcess, correspondant au réseau TransNet. Dans le fichier de configuration, le paramètre IIOP1_CFG a la valeur « C:\TSCS_W32\TSCS.cfg » qui représente le fichier de configuration résultant de l'application de configuration du pilote informatique.

Même si on a trois modules : un module d'entrée analogique, un module de sortie analogique et un module de sortie digitale, dans la bibliothèque des fonctions (*.pid) sept fonctions ont été définies : quatre fonctions pour le module d'entrée analogique, deux pour le module de sortie analogique et une pour le module digital. La figure 7.5 contient un imprimé-écran du PIO au moment de la définition des fonctions.

Pour le module d'entrée analogique les capteurs sont des thermocouples de type k, des RTD, des capteurs de pression et un débitmètre. Les plages de mesure sont des caractéristiques qui proviennent des instruments utilisés.

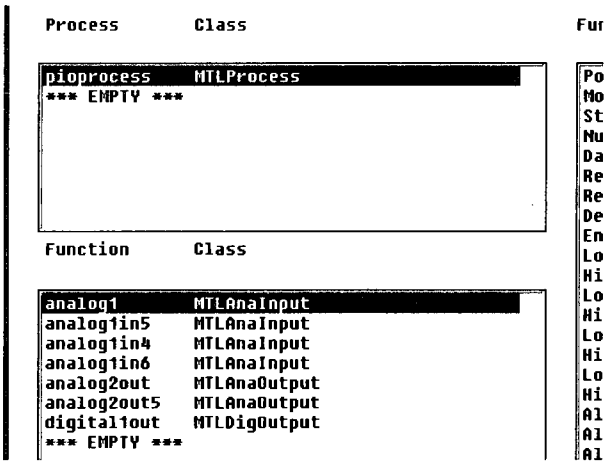

Figure 7.5 : La configuration du processus pioprocess.

Ces fonctions sont les suivantes :

- *analog1* de type MTLAnaInput. Cette fonction effectue la lecture des thermocouples. L'adresse du module est 2 (module d'entrée analogique);
- *analog1in4* de type MTLAnaInput. Cette fonction effectue la lecture du débitmètre. L'adresse du module est toujours 2 (module d'entrée analogique);
- *analog1in5* du type MTLAnaInput. Cette fonction effectue la lecture de la pression dans le condenseur. L'adresse du module est aussi 2 (module d'entrée analogique);
- *analog1in6* du type MTLAnaInput. Cette fonction effectue la lecture de la différence de pression entre la base et le haut du condenseur. Dans la stratégie continue, en partant de cette mesure on détermine le niveau de liquide dans le ballon condenseur.
- *analog2aut* du type MTLAnaOutput. Cette fonction envoie des commandes vers des vannes. L'adresse du module est 3 (module de sortie analogique);
- *analog2aut* du type MTLAnaOutput. Cette fonction envoie des commandes vers la vanne qui contrôle la pression dans le condenseur. L'adresse du module est 3 (module de sortie analogique);

- *digitallout* du type MTLDigOutput. Cette fonction envoie des commandes digitales qui seront utilisées pour arrêter le chauffage de la section d'essais. L'adresse du module est 1 (module de sortie digitale).

C'est important de remarquer que pour toutes les fonctions, l'intervalle de temps entre deux lectures successives est de 0.25s.

7.4 La construction de la stratégie continue pour le traitement des données (CS)

On doit mentionner dès au début que, à l'exception de la configuration de l'acquisition de données, les configurations effectuées dans la CS proviennent de l'ancienne application. Comme première étape, on a dû déclarer dans la couche principale une couche pour la stratégie continue nommée *strategus* et une couche reliée au processus du PIO utilisée dans notre cas *pioprocess*. Entre les deux on a défini un élément de type conduit, une sorte de liaison entre les deux serveurs. Le fichier dans lequel on travaille a été nommé *strategy.csb*. Selon l'action désirée, les données reçues ont été traitées de façon individuelle. Voici les données et leur traitement :

1. Le réglage pour la sauvegarde des données. Dans la couche *StringLayer* on a implémenté l'instruction SQL qui est utilisée dans le gérant des données pour sauvegarder les données reçues dans un fichier de sortie. Le nom du fichier est spécifié dans un élément de type boîte de saisie qui se trouve sur la fenêtre principale de l'application. Pour effectuer cette action on utilise quatre blocs qui sont représentés dans la figure 7.6. Dans le tableau 7.3 sont présentes les blocs utilisés dans la sauvegarde de données.

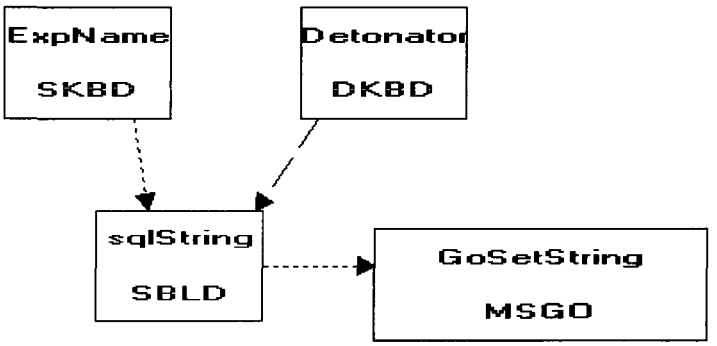

Figure 7.6 : La configuration de la sauvegarde des données en CS.

Tableau 7.3 Les blocs utilisés dans la sauvegarde des données.

Nom du bloc	Rôle
ExpName	Représente le nom du fichier.
Detonator	Provoque l'action de formation de la chaîne de caractères et l'envoi du résultat vers DM.
sqlString	Est la fonction qui produit l'instruction SQL. On intercale dans cette instruction le nom du fichier. Voici la structure de la commande SQL : 'MoveSince to:[',\39,'12-11-20 0:10:00',\39,'] SQL:',\39,'INSERT INTO ',S1,'.TXT VALUES (:all)',\39. Le caractère \39 est le (') et *S1* est le nom pour le fichier.
GoSetString	Est le bloc qui envoie la commande vers le bloc DM correspondant (*DM.DMprocess.inputGen.DfltCmd*).

2. Température à l'entrée de la douche. Le parcours des données est présenté dans la figure 7.7.

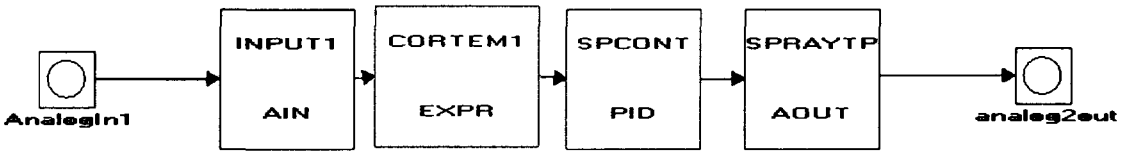

Figure 7.7 : La description du parcours des données pour contrôler le débit du spray en CS.

Dans *INPUT1* on lit la température. Par la suite, on effectue des corrections sur la lecture (-2,5 °C). Le résultat est l'entrée pour le contrôleur PID *SPCONT* (coefficients P=50, I=0.1, D=0). La sortie du contrôleur est envoyée vers la vanne de mélange à partir du bloc *SPRAYTP*.

3. Température à la sortie de l'échangeur de chaleur. Le parcours des données est présenté dans la figure 7.8.

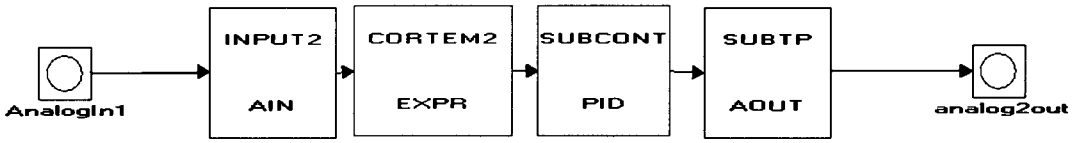

Figure 7.8 : La description du parcours des données pour contrôler la vanne de mélange à la sortie de l'échangeur de chaleur CS.

Dans *INPUT2* on lit la température. Par la suite, on effectue des corrections sur la lecture (-2,5 °C). Le résultat est l'entrée pour le contrôleur PID *SUBCONT* (coefficients P=50, I=0.1, D=0). La sortie du contrôleur est envoyée vers la vanne de mélange à partir du bloc *SUBTP*.

4. Température à la sortie du préchauffeur. Le parcours des données est présenté dans la figure 7.9.

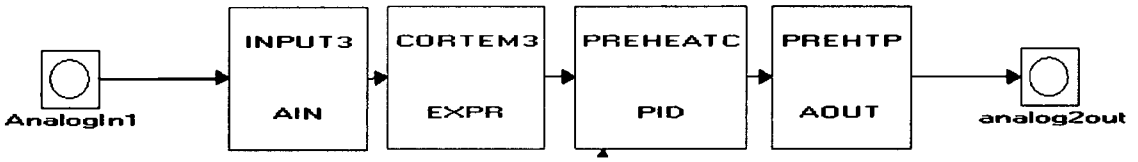

Figure 7.9 : La description du parcours des données pour contrôler la puissance électrique dans le préchauffeur en CS.

Dans *INPUT3* on lit la température. Par la suite, on effectue des corrections sur la lecture (-2,5 °C) dans le bloc *CORTEM3*. Le résultat est l'entrée pour le contrôleur PID *PREHEATC* (coefficients P=10, I=0.1, D=0). La limite supérieure

des valeurs de sortie est contrôlée à partir de la lecture du débitmètre. Si le débit a une valeur plus petite que 0.25 kg/s, la limite supérieure du contrôleur devient 0% et donc on arrête le chauffage de l'écoulement. La sortie du contrôleur est envoyée vers le préchauffeur à partir du bloc *PREHTP*.

5. Débit massique. On utilise un débitmètre à vortex pour la lecture de ce débit. Le parcours des données est présenté dans la figure 7.10.

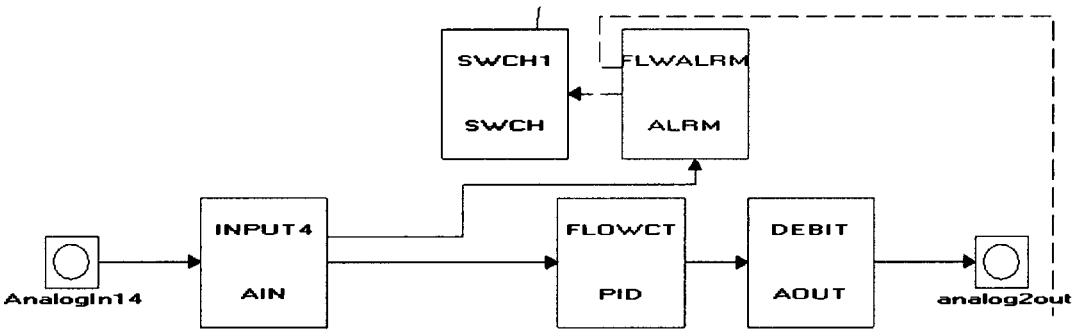

Figure 7.10 : La description du parcours des données pour contrôler le débit massique en CS.

Dans *INPUT4* on lit le débit actuel. Le débit est utilisé dans deux buts : pour envoyer un message d'alarme vers le préchauffeur (comme présenté dans le trajet des données précédent) ou pour contrôler une vanne. Dans le deuxième cas, le résultat est l'entrée pour le contrôleur PID *FLOWCT* (coefficients P=150, I=0.1, D=0). La sortie du contrôleur est envoyée vers la vanne de contrôle à partir du bloc *DEBIT*.

6. Pression dans le condenseur. Le parcours des données est présenté dans la figure 7.11.

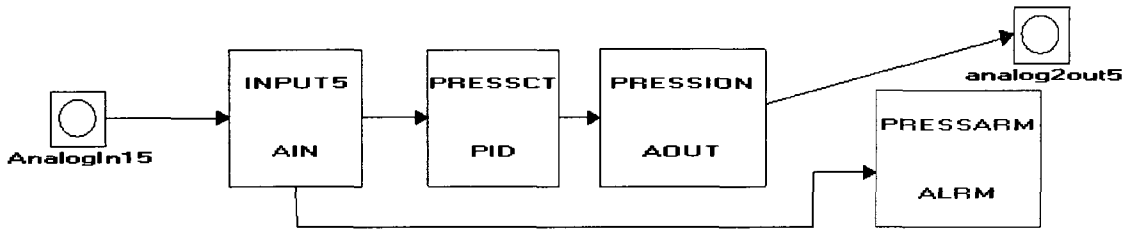

Figure 7.11 : La description du parcours des données pour contrôler la pression dans le condenseur en CS.

Dans *INPUT5* on lit la pression. Le résultat est l'entrée pour le contrôleur PID *PRESSCT* (coefficients P=150, I=0.1, D=0, Kbias=1). La sortie du contrôleur est envoyée vers la vanne de mélange à partir du bloc *PRESSION*. La valeur de la pression est aussi passée vers un bloc d'alarme qui se déclenche pour des pressions plus grandes que 20 bars.

7. Niveau du liquide dans le ballon condenseur. Ce niveau est déterminé à partir de la lecture d'un capteur de pression différentielle. Le parcours des données est présenté dans la figure 7.12. Dans *INPUT6* on lit la différence de pression entre le bas et le haut du condenseur. Dans *INPUT7* on lit la température dans le condenseur. Dans *INPUT8* on lit la température ambiante. On calcule dans les blocs de calcul des densités pour le Fréon liquide et vapeur à la température du condenseur et pour le Fréon liquide à la température extérieure. Dans le bloc *LEVELCOR* on calcule le niveau du liquide dans le condenseur selon la formule suivante :

$$(K4*((K1*K3*(I2-I3))-(K2*I1)))/(K1*(I4-I3)) \qquad (7.1)$$

Les paramètres de l'équation précédente sont présentés dans le tableau 7.4.

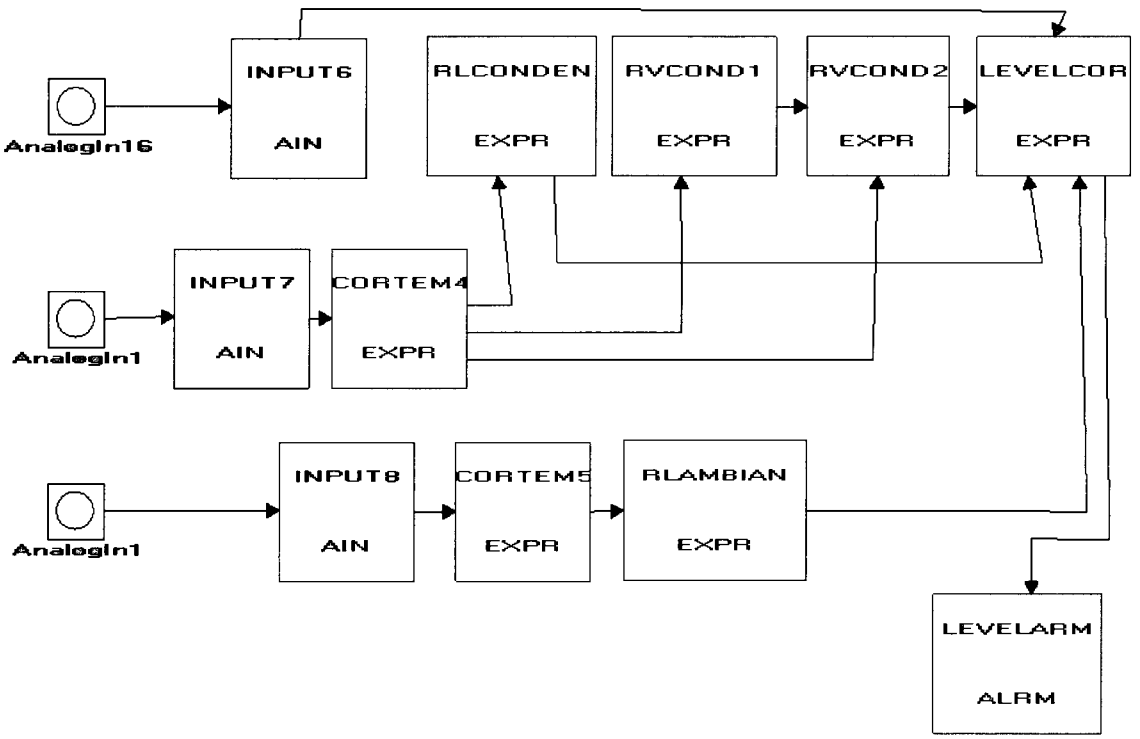

Figure 7.12 : La description du parcours des données déterminer le niveau du liquide dans le ballon condenseur en CS.

Tableau 7.4 Les paramètres de la formule utilisée dans le bloc LEVELCOR (CS).

Paramètre	Rôle
I1	La différence de pression mesurée;
I2	La densité du Fréon liquide à la température extérieure;
I3	La densité des vapeurs du Fréon à la température mesurée dans le condenseur;
I3	La densité du Fréon liquide à la température mesurée dans le condenseur;
K1	L'accélération gravitationnelle (g=9.81);
K2	Le facteur de proportionnalité pour la mesure du capteur de pression différentielle (248.65);
K3	La différence de niveau entre les points de mesure;
K4	Le facteur de proportionnalité (100).

Dans le bloc *LEVELALARM* on applique une alarme si le niveau est plus élevé que 45 ou plus petit que 25 cm.

8. La température à l'entrée de la pompe. Le parcours des données est présenté dans la figure 7.13.

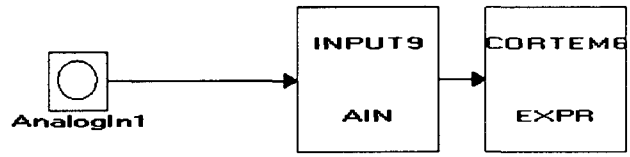

Figure 7.13 : La description du parcours des données pour déterminer la température à l'entrée de la pompe en CS.

Dans *INPUT9* on lit la température. Par la suite, on effectue des corrections sur la lecture (-2.5 °C) dans le bloc *CORTEM6*.

Pour pouvoir spécifier la liaison entre les terminateurs du CS et les fonctions du PIO (qui nous assure la liaison jusqu'aux capteurs et instruments de contrôle) l'application offre la possibilité de faire des associations entre des objets appartenant aux différentes couches (External connections). Les associations effectuées sont présentées dans la figure 7.14.

CS.strategus		PIO.pioprocess
analog2out(PREHTP.Out)	-->	analog2out.WriteOut{2}
analog2out(SUBTP.Out)	-->	analog2out.WriteOut{1}
analog2out(DEBIT.Out)	-->	analog2out.WriteOut{3}
analog2out(SPRAYTP.Out)	-->	analog2out.WriteOut{0}
analog2out5(PRESSION.Out)	-->	analog2out5.WriteOut{4}
AnalogIn1(INPUT7.In)	<--	analog1.ReadIn{6}
AnalogIn1(INPUT1.In)	<--	analog1.ReadIn{0}
AnalogIn1(INPUT9.In)	<--	analog1.ReadIn{8}
AnalogIn1(INPUT2.In)	<--	analog1.ReadIn{1}
AnalogIn1(INPUT3.In)	<--	analog1.ReadIn{2}
AnalogIn1(INPUT8.In)	<--	analog1.ReadIn{7}
AnalogIn14(INPUT4.In)	<--	analog1in4.ReadIn{3}
AnalogIn15(INPUT5.In)	<--	analog1in5.ReadIn{4}
AnalogIn16(INPUT6.In)	<--	analog1in6.ReadIn{5}
digital1out(BEEL.Out)	-->	digital1out.WriteOut{0}

Figure 7.14 : Les liaisons externes des blocks de CS.

Les indices dans les fonctions du processus d'entrée-sortie commencent à partir de la valeur 0.
L'imprimé-écran de la stratégie continue au complet est présenté dans la figure 7.15.

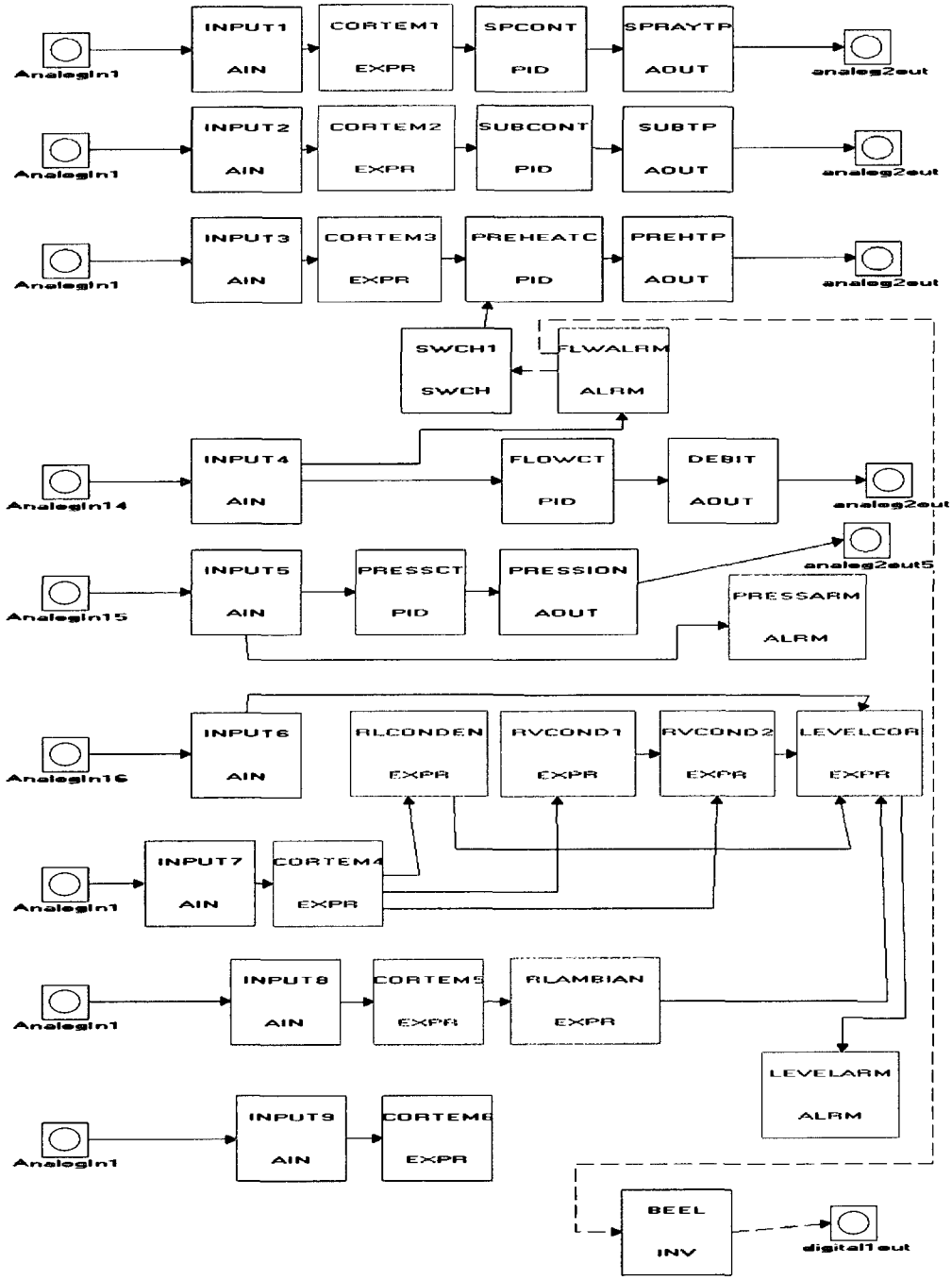

Figure 7.15 : Schéma de la stratégie continue.

7.5 La déclaration et l'implémentation des processus et des fonctions du gérant de données

On utilise le gérant des données pour sauvegarder les déterminations ou pour les montrer dans certains éléments de l'interface. Les fichiers écrits sont nommés « dmlib ». On a défini un seul processus nommé *DMprocess* dans lequel on a implémenté deux fonctions :

- *inputGen* du type MultyHistory qui permet de sauvegarder des données. Dans notre cas, les données proviennent du serveur de la stratégie continue et sont toutes des éléments *INPUT*. Donc, pour les éléments SrcConn0X on a établi les valeurs CS.strategus.INPUT*X*.Out (*X*=1,...8). La commande qui fait la description des paramètres de sauvegarde est aussi établie à partir de la stratégie continue comme on l'a déjà spécifiée. Le déclenchement de la sauvegarde consiste à attribuer la valeur 'Enabled' au champ SmplSvcPort (on va faire cela à partir de l'interface). Un imprimé-écran des paramètres de cette fonction est présenté dans la figure 7.16.

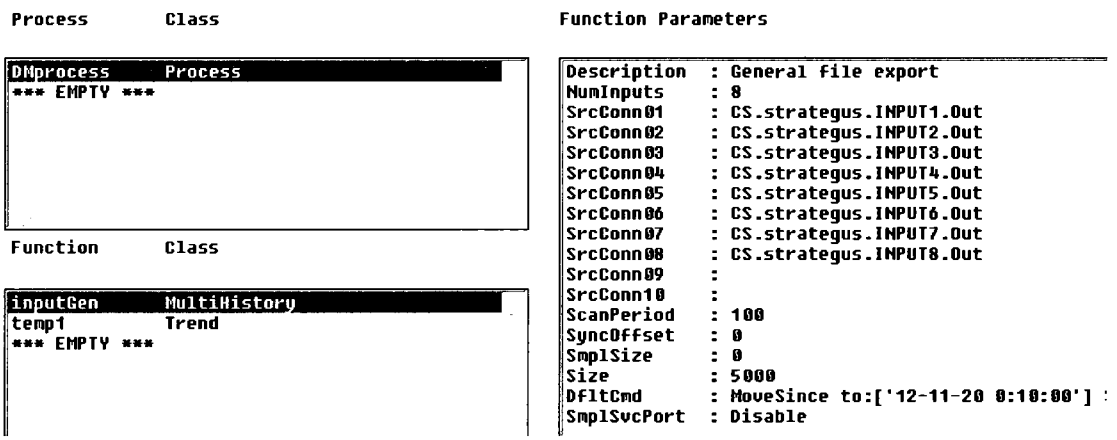

Figure 7.16 : La configuration de la fonction *inputGen* du processus *DMprocess* en DM.

- *temp1* est du type Trend. Elle est utilisée pour garder des données qui seront visualisées dans un élément d'interface sur la superficie de la fenêtre principale. On sauvegarde seulement la valeur de la température dans la chambre où se trouve la boucle. Pour spécifier cela, on a attribué à la propriété SrcConnName la valeur CS.strategus.Cortem5.Out qui correspond à cette température. Un imprimé-écran des paramètres de cette fonction est présenté dans la figure 7.17.

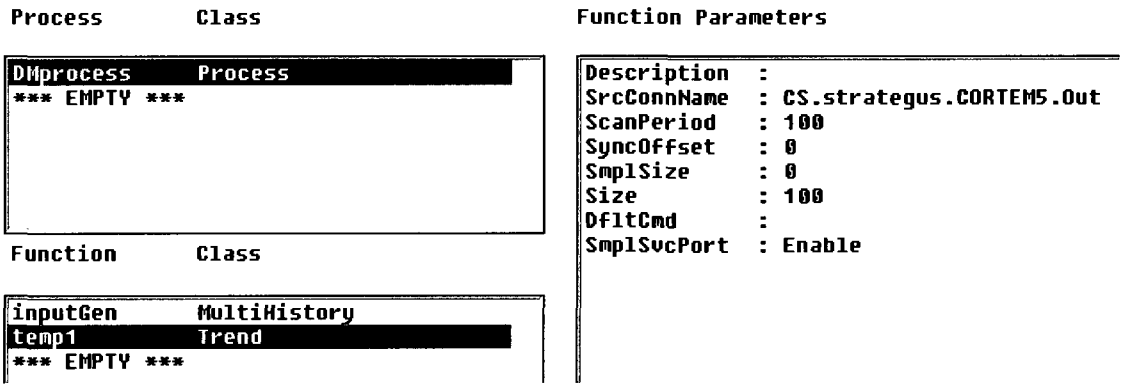

Figure 7.17 : La configuration de la fonction *temp1* du processus *DMprocess* en DM.

7.6 La construction de l'interface usager

L'interface comprend seulement la fenêtre principale de l'application à partir de laquelle on contrôle les dispositifs où on affiche les valeurs. Pour les contrôleurs on affiche le paramètre de contrôle (boîte de saisie de données), la valeur cible (boîte de saisie de données), le signal de contrôle du PID (boîte de saisie de données) et le mode de calcul pour le signal de contrôle (manuel ou automatique; des boutons radio). L'imprimé-écran de la fenêtre réalisée est présenté dans la figure 7.18.

Chaque contrôleur du système est suivi et manipulé par l'intermédiaire des éléments d'interface usager :

- Pour la température du liquide dans le gicleur ont été crées : une boîte de saisie pour la température actuelle qui est connectée à *CS.strategus.CORTEM1.Out*, une boîte de saisie pour la valeur cible qui est connectée à *CS.strategus.SPCONT.Setp*, une boîte de saisie pour le signal de contrôle du PID qui est connectée à *CS.strategus.SPCONT.Out* et deux boutons radio connectés à *CS.strategus.SPCONT.ManualStat* pour activer ou désactiver le contrôleur;

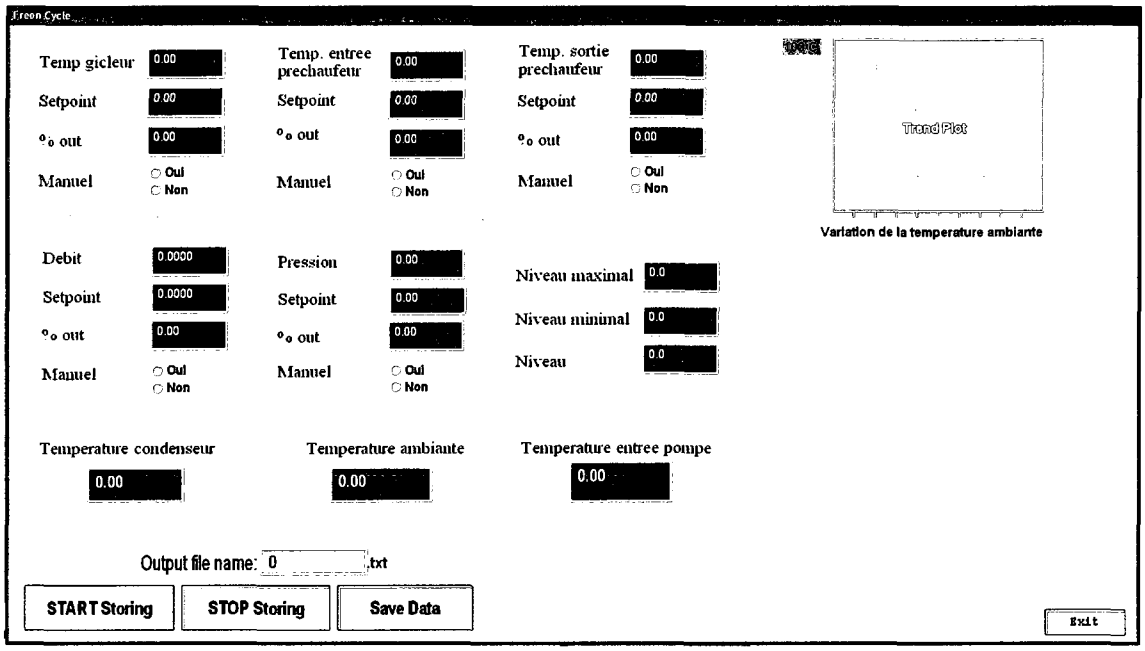

Figure 7.18 : La fenêtre principale de l'interface de l'application.

- Pour la température à l'entrée du préchauffeur (qui est obtenue à partir du mélange entre deux écoulements de températures différentes) ont été crées : une boîte de saisie pour la température actuelle qui est connectée à *CS.strategus.CORTEM2.Out*, une boîte de saisie pour la valeur cible qui est connectée à *CS.strategus.SUBCONT.Setp*,, une boîte de saisie pour le signal de contrôle du PID qui est connectée à *CS.strategus.SUBCONT.Out* et deux boutons radio connectés à *CS.strategus.SUBCONT.ManualStat* pour activer ou désactiver le contrôleur.

- Pour la température à la sortie du préchauffeur ont été crées : une boîte de saisie pour la température actuelle qui est connectée à *CS.strategus.CORTEM3.Out*, une boîte de saisie pour la valeur cible qui est connectée à *CS.strategus.PREHEATC.Setp*, une boîte de saisie pour la le signal de contrôle du PID qui est connectée à *CS.strategus.PREHEATC.Out* et deux boutons radio connectés à *CS.strategus.PREHEATC.ManualStat* pour activer ou désactiver le contrôleur.
- Pour le débit ont été crées : une boîte de saisie pour le débit actuel qui est connectée à *CS.strategus.INPUT4.Out*, une boîte de saisie pour la valeur cible qui est connectée à *CS.strategus.FLOWCT.Setp*, une boîte de saisie pour le signal de contrôle du PID qui est connectée à *CS.strategus.FLOWCT.Out* et deux boutons radio connectés à *CS.strategus.FLOWCT.ManualStat* pour activer ou désactiver le contrôleur.
- Pour le contrôleur de pression dans le condenseur ont été crées : une boîte de saisie pour la pression actuelle qui est connectée à *CS.strategus.INPUT5.Out*, une boîte de saisie pour la valeur cible qui est connectée à *CS.strategus.PRESSCT.Setp*, une boîte de saisie pour le signal de contrôle du PID qui est connectée à *CS.strategus.PRESSCT.Out*, et deux boutons radio connectés à *CS.strategus.PRESSCT.ManualStat* pour activer ou désactiver le contrôleur.

Les restants des éléments d'interface sont présentés dans le tableau 7.5.

Tableau 7.5 Éléments d'interface définis dans l'OI

No.	Élément d'interface usager
1	Une boîte de saisie pour le niveau maximal de liquide dans le ballon condenseur qui est connectée à *CS.strategus.LEVELARM.HiAlrmVal*;
2	Une boîte de saisie pour le niveau minimal de liquide dans le ballon condenseur qui est connectée à *CS.strategus.LEVELARM.LoAlrmVal*;
3	Une boîte de saisie pour le niveau actuel de liquide dans le ballon condenseur qui est connectée à *CS.strategus.LEVELARM.In*;
4	Une boîte de saisie pour la température dans le ballon condenseur qui est connectée à *CS.strategus.CORTEM4.Out*

No.	Élément d'interface usager
5	Une boîte de saisie pour la température dans la chambre de la boucle qui est connectée à *CS.strategus.CORTEM5.Out*
6	Une boîte de saisie pour la température de l'écoulement à l'entrée de la pompe qui est connectée à *CS.strategus.CORTEM6.Out*
7	Une boîte de saisie pour la température de l'écoulement à l'entrée de la pompe qui est connectée à *CS.strategus.CORTEM6.Out*
8	Une boîte de saisie pour le nom du ficher de sortie qui est connectée à *CS.strategus.ExpName.StrOut*
9	Un bouton qui déclenche le début de l'acquisition de données. L'action d'appui génère le changement du *DM.DMprocess.inputGen.SmplSvcPort* à la valeur *Enabled*;
10	Un bouton qui déclenche la fin de l'acquisition de données. L'action d'appui génère le changement du *DM.DMprocess.inputGen.SmplSvcPort* à la valeur *Disabled*;
11	Un bouton qui déclenche la sauvegarde des données génère le changement du *DM.DMprocess.inputGen.StrmSvcPort* à la valeur *Execute*;
12	Un diagramme pour représenter la variation de la température. On a défini le stylo numéro 1 à la valeur : *DM.DMprocess.temp1.KeyedDataOut {0} -> CS.strategus.CORTEM5.Out*

Conclusions

Pour l'application d'acquisition de données, on a effectué premièrement une étude des besoins réels du laboratoire. Pour que cette application soit utilisée pour les deux boucles, on a dû tenir compte les caractéristiques principales de deux boucles (i.e., la boucle eau-vapeur et la boucle au Fréon). Par la suite, on a programmé dans le langage spécifique de l'équipement Tempscan les séquences nécessaires pour chaque étape de fonctionnement (i.e., les réglages initiales du système, le transfère des données de la mémoire tampon du Tempscan vers le disc dur de l'ordinateur, etc.).

À partir du langage de programmation C++, en utilisant les fonctions du pilote informatique, on a testé chaque séquence à la fois. Les fonctions du pilote informatique finissent parfois leur exécution, dans le programme où elles sont appelées, plus tôt que l'équipement Tempscan. Donc, une question importante a été d'établir les temps d'attente nécessaires pour l'ensemble de l'application au moment d'exécution de chaque instruction. Le pilote informatique ne fournit aucune façon dynamique pour déterminer les périodes d'indisponibilité du système d'acquisition Tempscan.

Par la suite, on a programmé l'interface graphique en Visual C++.NET. Le défi important de cette étape a été de synchroniser le fonctionnement des éléments d'interface avec les réponses de l'équipement d'acquisition de données. Pour pouvoir faire cela, on a utilisé des fils d'exécution multiples.

On a essayé l'application initiale et l'application réalisée en parallèle pendant l'état d'arrêt de la boucle eau-vapeur. On a comparé les résultats dans les deux cas et on a constaté qu'ils sont identiques. Comme étape finale, on a essayé l'application dans des conditions réelles de travail de la boucle thermique à eau-vapeur. On a constaté que le logiciel est devenu beaucoup plus stable (la fréquence d'interruption est passée de 0.125

fois/heure à 0.004 fois/heure) et souple pour le changement de système d'exploitation. La conclusion générale des tests de validation effectués est que toutes les tâches ont été accomplies.

En ce qui concerne l'application pour le contrôle de la boucle au Fréon, premièrement on a étudié la nature des changements effectués. En fonction de cela, on a effectué une étude de l'application déjà existante pour pouvoir respecter, par la suite, ses spécifications. Comme étape suivante, on a effectué l'installation et la configuration des applications pré-requises. On a dû effectuer aussi des réglages pour que le « pilote informatique » soit capable de reconnaître correctement les caractéristiques des capteurs et des dispositifs de commande du réseau TransNet.

Avant de commencer la réalisation des applications en Paragon, on a dû projeter la structure logique de l'application et la distribution des composantes de l'application par serveurs et clients. Dans l'étape suivante, on a défini les processus et les fonctions correspondantes pour les serveurs. Une chose notable, dans la CS, une bonne partie du travail provient de l'application antérieure (la description des étapes de calcul pour les flux des données d'entrée vers les données de sortie). En continuant, on a réalisé la nouvelle interface graphique, en respectant en partie les spécifications de l'ancienne interface (position des panneaux des contrôleurs et affichage des températures). Comme étape finale, on a essayé les composantes utilisables de l'application.

On considère que la première application servira le Laboratoire Thermo-hydraulique de l'IGN dans le but d'effectuer l'acquisition de données pendant les expériences dans les boucles eau-vapeur et au Fréon. Par contre, la deuxième application devra subir des ajustements en fonction aussi de la configuration finale de la boucle au Fréon, pour pouvoir devenir entièrement fonctionnelle.

Bibliographie

[1] T. Budd, et C. Horstmann, La Bible C++, Éditions Micro Application, 2004

[2] A. Olekhnovitch, Étude du flux des chaleurs critiques à des pressions faibles, Thèse présentée en vue de l'obtention du diplôme de Philosophiæ Doctor (Ph. D.), 1997

[3] B. Stroustrup, The C++ Programming Language, Addison Wesley, Reading Mass. USA, 2000

[4] A. Teyssedou et A. Olekhnovitch, Détermination des valeurs de compensation (« offset ») de l'instrumentation – Communication interne

[5] IOtech Inc., IOTech Personal488 User's Manual For Windows® 95/98/Me/NT/2000/XP, p/n 495-0903 Rev. 3.0, 2001 – Document en format électronique

[6] IOtech Inc., IOTech Personal488 User's Guide For Windows® 95/98/Me/NT/2000/XP p/n 495-0940 Rev 1.0, 2003 – Document en format électronique

[7] IOtech Inc., TempScan & MultiScan Quick Start, 2008 – Document en format électronique

[8] IOtech Inc., TempScan / MultiScan User's Manual p/n 446-0901 rev 3.0, 2008 – Document en format électronique;

[9] Microsoft Corporation, Visual C++ Reference, http://msdn.microsoft.com/en-ca/library/ty9hx077.aspx

[10] MOST Incorporated, I/O95™ & TransPort™ Application Note – PCI Interface Installation and Initialization Ver. 2.01, Document en format électronique

[11] MTL Incorporated, MTL I/O 95 Révision 2.00, 1998 – Document en format électronique

[12] NemaSoft Inc., Paragon 5.3 Getting Started, 1998 – Document en format électronique

[13] NemaSoft Inc., Paragon 5.3 User's Guide, 1998 – Document en format électronique

[14] Nematron Corporation, Nematron Software 5.60 Paragon, OpenControl, HyperKernel – Release Notes, 2003 – Document en format électronique