UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

ARCHITECTURE LOGICIELLE DISTRIBUÉE POUR UN SYSTÈME DE SENSEURS PHOTONIQUES BASÉS SUR LE PRINCIPE D'INTERFÉROMÉTRIE EN LUMIÈRE BLANCHE

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

FATIHA DJEBBAR

MAI 2003

UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS
Département d'informatique et d'ingénierie

Ce mémoire intitulé :

ARCHITECTURE LOGICIELLE DISTRIBUÉE POUR UN SYSTÈME DE SENSEURS PHOTONIQUES BASÉS SUR LE PRINCIPE D'INTERFÉROMÉTRIE EN LUMIÈRE BLANCHE

présenté par
Fatiha Djebbar

pour l'obtention du grade de maître ès sciences (M.Sc.)

a été évalué par un jury composé des personnes suivantes :

Wojtek J. Bock…….directeur de recherche

Kamel Adi……codirecteur de recherche

Karim El Guemhioui…….président du jury

Michal Iglewski……....membre du jury

Mémoire accepté le : 14 mai 2003

À mes parents.

À mes frères et sœurs et tout particulièrement
à mon frère Aref et ma sœur Nadia.

Avant-propos

Je tiens à exprimer ma profonde gratitude à M. Wojtek J. Bock, mon directeur de recherche, et à M. Kamel Adi, mon codirecteur, pour leur patience, leur confiance, pour leur encadrement, le temps qu'ils m'ont consacré et pour les conseils judicieux qu'ils m'ont généreusement donnés.

Un spécial remerciement à M. Karim El Guemhioui pour son aide très précieuse, son professionnalisme et son sens des responsabilités.

Mes remerciements vont également à Mme Ilham Benhayia pour ses conseils précieux.

Je remercie aussi mes amies Nawel Chefai, Samar Dakdouki et Linda Gamaz pour leur soutien moral et leur précieuse amitié.

Ce mémoire représente l'aboutissement d'un travail qu'il m'aurait été sans doute impossible de réaliser sans l'aide, l'amour et le soutien moral et inconditionnel de mes parents, mes frères et sœurs. J'espère qu'ils trouveront ici l'expression de mon éternelle reconnaissance.

Résumé

Il est bien connu que depuis quelques années, un intérêt primordial est accordé à l'amélioration des procédés d'acquisition et de traitement de données. La dernière décennie a vu une forte émergence des systèmes d'acquisition de données dans la plupart des domaines de l'industrie. Ils sont devenus une partie intégrante du processus de surveillance locale et à distance des systèmes.

En dépit de la profusion de ce type de systèmes sur le marché et les solutions proposées, quelques lacunes persistent. En effet, l'aspect temps réel de l'architecture n'ont pas été entièrement explorés.

Dans cette perspective, nous proposons une nouvelle architecture logicielle à plate-forme indépendante pour un système d'acquisition de données distribué, temps réel basé sur les senseurs photoniques. L'architecture en question est validée par la production d'un logiciel, qui permet aux utilisateurs d'accéder simultanément à des instruments localisés dans des sites différents pour l'acquisition, le traitement et l'affichage de données. Les résultats obtenus montrent que nous avons pu développer un système performant, distribué, portable et contenant de robustes mécanismes de sécurité pour l'acquisition et le traitement des données.

Abstract

In the last few years, a strong resurgence of interest in improving procedures and processing techniques used in remote data acquisition systems has been observed. This has led to new applications of such systems emerging in many industrial or environmental control areas of technology. Data acquisition systems have become an integral part in monitoring local and distributed systems.

Despite the profusion of commercially available data acquisition systems, there are still persisting limitations. The real time data collection was not fully explored nor the generic aspects of the specific applications were carefully considered.

In this thesis, we propose a new distributed software architecture for real-time, platform-independent data acquisition system specifically designed for photonic sensors. This architecture is employed to develop a software, which allows many users to simultaneously access instruments physically located in different sites. The obtained results show, that we have developed, built and initially tested a distributed, real-time, portable data acquisition system. The proposed system embeds robust security mechanisms for user identification and encrypted transactions.

Table des matières

Liste des figures

Chapitre 1

Introduction

Depuis la dernière décennie, les recherches relatives au domaine de l'acquisition de données à travers le réseau Internet ont suscité un intérêt primordial et se sont révélées être d'une importance considérable dans le domaine industriel. En effet, les systèmes d'acquisition de données sont à la base des outils informatiques qui permettent la surveillance locale et à distance des systèmes utilisant des senseurs.

L'expansion rapide de l'Internet et sa capacité de fournir une interconnexion entre des réseaux séparés géographiquement, ont influencé son utilisation pour le déploiement des systèmes distribués d'acquisition de données. De plus, Internet favorise une interaction entre ces systèmes permettant ainsi une collaboration étroite pour l'accès, le traitement et la diffusion en temps réel des données (Figure 1.1). Les systèmes d'acquisition de données peuvent être d'une utilité cruciale pour différents domaines d'applications, parmi lesquels nous pouvons citer: la

surveillance environnementale (surveillance de la qualité de l'eau, la métrologie, les changements climatiques, etc.), les infrastructures de génie civil, de transport, etc. Un exemple d'application considère la problématique de la sécurité d'une structure civile (pont). Dans le but d'assurer une surveillance à long terme et permanente du pont, un système basé sur des senseurs à fibres optiques, noyés dans la structure (Figure 1.2), doit être installé pour pouvoir mesurer les déformations. Les mesures effectuées constituent des informations d'une importance capitale aussi bien pour un gouvernement qui doit assurer la sécurité des utilisateurs du pont, que pour le laboratoire de recherche qui veut améliorer la qualité des produits utilisés pour la construction de ces structures. L'accès à distance à cette information ainsi que la possibilité d'une collaboration entre ces deux parties peuvent se réaliser par l'utilisation de l'Internet : cette technologie qui a la possibilité de fournir l'accès entre des réseaux séparés géographiquement.

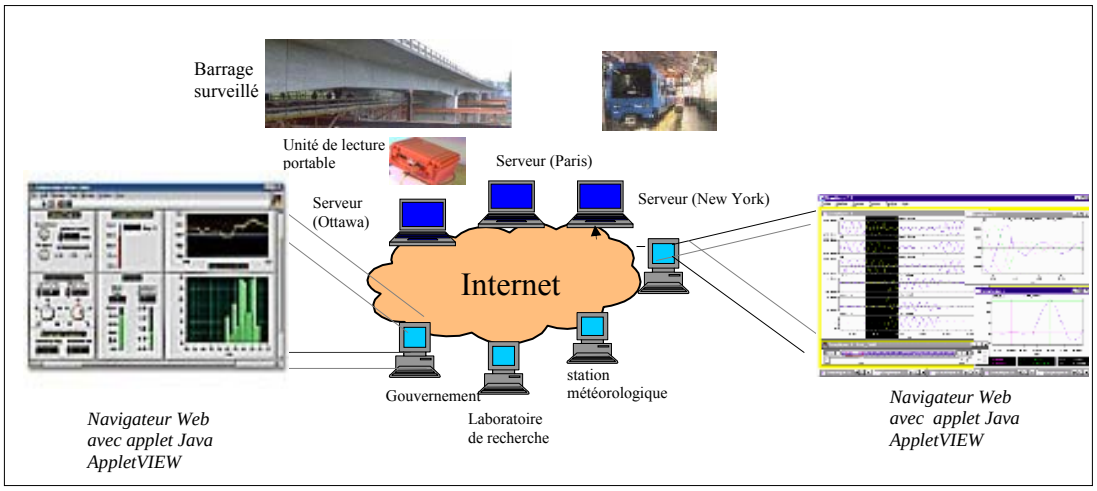

Figure 1.1 : Acquisition de données via Internet

Différents clients (gouvernement, laboratoire de recherche, etc.) ont un accès direct par Internet aux systèmes d'acquisition localisés dans différentes parties du monde. Les clients utilisent un navigateur Web.

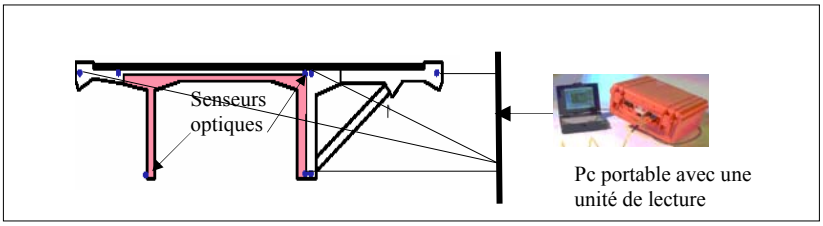

Figure 1.2 : Disposition des senseurs dans le barrage

Les dernières recherches sur les systèmes d'acquisition de données se sont révélées très fructueuses [1, 7, 12, 15, 16], elles ont oeuvré à l'amélioration du processus de surveillance à distance. Toutefois, ces travaux présentent quelques lacunes auxquelles il convient de remédier. Ainsi, une partie de ces travaux ne considère que la visualisation graphique des données déjà collectées à travers le Web et la génération dynamique des rapports HTML [16, 7], de plus, ces solutions sont adaptées à un domaine précis. L'autre partie bien qu'elle propose des solutions plus complètes, soit par la présentation d'une architecture logicielle [1, 15], soit par la définition d'un modèle conceptuel pour un système d'acquisition de données générique [12], ne sont malheureusement pas portables ou en temps réel. Nous pensons que cette propriété est essentielle pour assurer la surveillance des systèmes à distance.

Ce mémoire nous permet de pallier aux lacunes des systèmes d'acquisition de données existants. Ainsi, nous présentons une architecture logicielle distribuée pour un système d'acquisition de données qui est à la fois :

- **Adapté à la génération des senseurs photoniques** : En effet, jusqu'à présent, il n y a pas de système d'acquisition de données adapté à la génération des senseurs photoniques.

- **Distribué** : L'évolution rapide de l'Internet et la mobilité des clients ont fait que l'acquisition et le traitement des données à distance sont devenus à la fois possibles et indispensables.

- **Temps réel** : L'environnement dans lequel les systèmes d'acquisition de données opèrent est souvent critique, un système temps réel est un atout considérable.

- **Interface utilisateur configurable** : Vu la profusion des systèmes d'acquisition de données, nous proposons une solution avec interface utilisateur configurable, adaptée à plusieurs types d'applications industrielles.

- **À Plate-forme indépendante** : Puisque, nous nous servirons du réseau Internet pour effectuer l'acquisition et le traitement de données, notre système devrait pouvoir fonctionner sur des plates-formes différentes.

L'architecture que nous proposons est une innovation dans le domaine des systèmes d'acquisition de données dans la mesure où ces caractéristiques lui permettent un vaste champ d'applications industrielles.

Le prototype logiciel développé dans le cadre de ce mémoire, est validé sur un système de senseurs interférométriques basés sur la lumière blanche et sur la transmission lumineuse par fibres optiques hautement biréfringentes. Ce système a été breveté à l'université de Québec en Outaouais (UQO) (Dr. W.Bock, W.Urbanczyk, US Patent No 5 517 022, obtenu le 14 mai 1996).

1.1 Structuration du mémoire

Le reste du mémoire est organisé comme suit :

Le chapitre 2 présente l'état de l'art de notre thème de recherche. Il est organisé en deux parties. La première présente une étude critique sur les systèmes existants d'acquisition de données à distance. La deuxième décrit les différentes technologies en lien avec les senseurs à fibres optiques.

Au chapitre 3, nous présentons l'architecture globale du système distribué d'acquisition de données. En premier, nous allons faire la modélisation statique du système, basée sur une analyse des besoins de l'utilisateur et de l'application. En second, nous présentons une étude détaillée sur les principaux modules du système.

Dans le chapitre 4, l'aspect dynamique du système d'acquisition de données est présenté. Nous allons faire la modélisation dynamique de l'application serveur, suivie par celle de l'application client. Dans les deux cas, l'accent est mis sur la nature des communications et les interactions possibles entres les différents objets constituant les deux applications.

Le chapitre 5 est consacré à notre méthodologie pour l'acquisition de données. Nous ferons la description du module responsable de cette fonction et nous discuterons des différentes étapes suivies pour son développement.

Dans le chapitre 6, nous présentons les mécanismes d'authentification sous-jacent à notre système d'acquisition de données. En premier, nous allons décrire le protocole SSL (Secure Socket Layer) utilisé pour l'encryptage des données transmises à travers le réseau. En second, nous présentons les mécanismes d'authentification et les algorithmes assurant la confidentialité des données, implantés dans notre système. Finalement, nous allons décrire en détail les sous modules responsables des communications sécurisées.

Dans le chapitre 7, nous présentons les protocoles de communication utilisés dans notre système distribué d'acquisition de données. Tout d'abord, nous allons faire la description du format des messages échangés, puis nous présentons l'ensemble des règles de communication entre les deux parties du système et enfin, le module responsable de la vérification et de l'interprétation des messages échangés dans notre système est présenté.

Au chapitre 8, l'approche utilisée pour la conception de l'interface utilisateur graphique est décrite. Nous expliquons le but principal d'une interface configurable et nous présentons en détail les fonctionnalités du module principal pour l'interface utilisateur.

Dans le chapitre 9, nous présentons l'aspect temps réel de notre système. Ainsi, nous explorons l'analyse de performance pour le système d'acquisition de données dans son environnement réel. Cette analyse sera réalisée grâce à l'application des

théories des algorithmes d'ordonnancement pour les systèmes temps réel distribués. L'analyse de performance nous permettra de vérifier si les dates limites des événements du système vont être respectées et par conséquent si une restructuration du système serait nécessaire.

Finalement, une brève conclusion présentera les principaux résultats et l'importance de nos travaux, ainsi qu'un aperçu des développements futurs possibles.

Chapitre 2

Etat de l’art

Dans ce chapitre nous présentons une revue de la littérature sur les concepts clés de notre mémoire. En premier, nous expliquons le but principal de l’acquisition de données, ensuite, nous présentons les standards et les systèmes d’acquisition de données les plus importants. Enfin, nous allons faire un survol des technologies sous-jacentes aux senseurs à fibres optiques.

2.1 Acquisition et traitement de données

Un système d’acquisition de données est un ensemble de ressources logicielles et matérielles qui fournissent les moyens pour accéder aux données d’un

système utilisant des senseurs. Par la suite, ces données seront interprétées, traitées et converties en une information utile qui sera distribuée aux utilisateurs.

Un système d'acquisition de données de base est constitué d'un système à senseurs, d'un convertisseur de données, d'un système de traitement de données et d'un système de visualisation et de sauvegarde (Figure2.1).

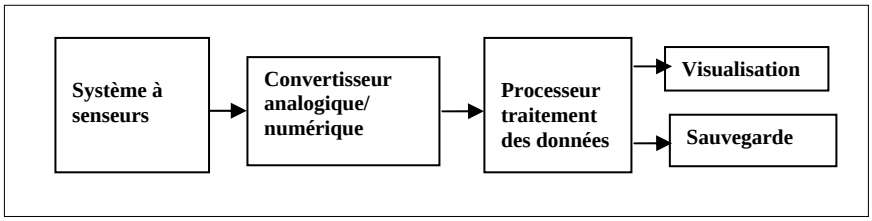

Figure 2.1 : Système typique d'acquisition de données

Un système d'acquisition de données plus complexe doit inclure certaines fonctions qui lui permettront d'acquérir et de traiter les données en temps réel, à distance, à partir d'environnements hétérogènes, etc. Ainsi, ces systèmes peuvent rencontrer les exigences des clients potentiels, qui sont en majorité mobiles et profiter de l'évolution du secteur des télécommunications.

2.2 Standards sur les systèmes d'acquisition de données

De nos jours, plusieurs standards sur les systèmes d'acquisition de données existent, parmi lesquels, nous pouvons distinguer :

- ***Interchangeable Virtual Instruments (IVI)***: C'est un nouveau standard fondé en 1997 par National Instruments [9] dans le but de permettre l'interchangeabilité des instruments entre différents fournisseurs. *IVI* incorpore des caractéristiques lui permettant d'améliorer les performances des systèmes et la réduction du temps de programmation et de maintenance des systèmes. De plus, il offre des spécifications aux clients possédant des systèmes standards permettant l'accès à des instruments tels que les oscilloscopes, les multimètres digitaux et les commutateurs.

- ***Object Linking and Embedding-for Process Control (OPC):*** Ce standard a été fondé en 1995 par l'organisation *OPC* [14]. Il a été défini pour standardiser les échanges de flux entre les serveurs et les clients, puisque la communication en environnement hétérogène posait des problèmes de maintenance et de pérennité. Dans ce standard, les serveurs collectent des données dans des équipements, généralement des automates, les traitent selon les indications des spécifications *OPC* et exposent des interfaces aux clients *OPC*. Ces interfaces permettent aux clients *OPC*, en fonction de la spécification concernée, d'accéder aux données en temps réel, aux données historiques, aux alarmes, etc. Ce standard permet aussi d'éviter la prolifération des protocoles, de remplacer ou d'ajouter des applications clientes sans toucher aux serveurs et sans altérer les autres clients. Il permet également de réaliser des supervisions plus performantes et pour un coût réduit, les applications peuvent être améliorées en fonction des nouveaux besoins des clients.

- ***Open Data Acquisition Standard (ODAS)***: Fondé en 1996 par l'Association d'acquisition de données ouverte [13] pour permettre l'interopérabilité des cartes d'acquisition de données entre différents fournisseurs. L'utilisation du standard *ODAS* procure l'avantage de réduire le temps de développement des applications et permet d'avoir des produits plus robustes puisqu'il se concentre sur l'ajout de fonctionnalités aux matériels et logiciels plutôt que sur le développement de nouveaux équipements.

2.3 Systèmes existants

Cette section à pour objectifs de présenter une étude critique des systèmes actuels d'acquisition de données. Cette étude nous permettra de mettre en lumière les faiblesses de ces systèmes motivant ainsi nos objectifs de recherche.

- Dans [1], Bertocco et Parvis proposent une architecture logicielle à plate-forme indépendante pour effectuer des mesures sur un système distribué. L'architecture proposée permet aux utilisateurs d'accéder aux instruments éloignés afin de collecter les données mesurées. Le logiciel inclut des mécanismes de sécurité pour l'identification des utilisateurs et le cryptage des données transmises à travers le réseau. Bien que cette solution présente plusieurs avantages, elle n'est malheureusement pas à temps réel, une caractéristique très importante pour les systèmes d'acquisition de données.

- Définir un modèle conceptuel pour les systèmes d'acquisition de données fut l'objet d'une étude réalisée dans [12] par Nieva et Wegmann, afin que

les développeurs puissent avoir un point de départ (modèles et techniques) pour la conception de systèmes d'acquisition de données spécifiques. Ce modèle pourrait être appliqué pendant la phase d'analyse du système, en vue de faciliter la compréhension de sa problématique, et permettre ainsi de réduire le temps et le prix de développement des systèmes d'acquisition de données. Cette solution est aussi incomplète, puisque le comportement dynamique (comportement du système dès l'exécution de certaines contraintes) du système n'a pas été pris en considération.

- Un système distribué d'acquisition de données a été conçu par J.Pareira et al [15]. Le système est basé sur une architecture à quatre couches : acquisition des données, stockage des données, analyse des données et prise de décision. Son utilisation est spécialement dédiée à des médecins pour la reconstruction, le traitement et l'analyse d'images rayons X des patients. Les informations véhiculées par ce système, étant très critiques et strictement confidentielles, des mécanismes de sécurité ont été implantés. Bien que cette solution est considérablement importante pour le domaine médical, elle n'est malheureusement pas portable. En effet, le système a été construit par plusieurs ensembles d'outils, ce qui rendra sa maintenance une tache ardue.

- Dans [16], Toran et al, ont développé un système distribué pour la gestion et le contrôle de la qualité de l'eau dont le but est de fournir aux utilisateurs des services tels que : la sauvegarde automatique des données, la visualisation des résultats de mesures à travers le Web et la génération dynamique des rapports HTML. Cette solution présente des

lacunes, puisque l'accès aux données est limité à celles déjà collectées antérieurement (même lacune est trouvée dans [7]), aucun mécanisme de sécurité n'a été prévu lors de leurs transmission et aucune interaction réelle entre les clients et le système n'est directement établie. De plus, ce système est spécifique au contrôle de la qualité de l'eau et ne peut être employé dans d'autres types d'applications.

2.3.1 Discussion

Malgré la diversité des solutions proposées pour développer des systèmes d'acquisition de données (proposition d'architecture, conception de logiciel et définition de spécifications), quelques lacunes sont toujours présentes. Certaines solutions ne traitent que l'accès aux données déjà mesurées. Étant spécifiques, leurs champs d'application sont limités. Les architectures proposées ne tiennent compte que du comportement statique du système et font abstraction du comportement dynamique.

Notre mémoire nous permet de pallier aux lacunes des systèmes d'acquisition de données existants en présentant une architecture logicielle distribuée à la fois :

- **Adaptée aux senseurs photoniques** : En effet, jusqu'à présent, il n y a pas de système d'acquisition de données adapté aux senseurs photoniques. Le prototype logiciel que nous nous proposons de développer, sera validé sur un système de senseurs interférométriques, basés sur la lumière blanche et sur la transmission lumineuse par fibres optiques hautement biréfringentes.

- **Distribuée** : La conception d'une architecture logicielle pour un système d'acquisition de données à distance effectuant plusieurs activités, nous a conduit à nous tourner vers une architecture distribuée de type client/serveur. Le niveau client sera composé d'interfaces graphiques permettant l'accès des utilisateurs au système d'acquisition, ainsi que quelques procédures permettant la réception et l'interprétation des résultats. Par contre, Le niveau serveur contiendra les procédures nécessaires pour gérer la logique de traitement. Le système d'acquisition est composé de deux parties. Une partie matérielle représentée par les senseurs à fibres optiques responsable de la collecte de données et une partie logicielle dépendante de cette dernière.

- **Temps réel** : L'environnement dans lequel les systèmes d'acquisition de données temps réel opèrent est un facteur important à considérer lors de leurs conceptions. L'exactitude d'un tel système ne dépend pas seulement de ses résultats logiques, mais aussi du temps mis pour les produire. Un système temps réel doit accomplir des tâches supplémentaires pour être classé comme tel. Par exemple, le système doit gérer un module d'estimation du temps de traitement et de réponse pour une requête émanant d'un usager, de plus il doit implanter un ordonnanceur de tâche qui permet d'attribuer dynamiquement des priorités aux requêtes. Également, le système doit garantir le respect des événements sélectionnés lors de situations de crises et doit parfois décider de ne livrer qu'une réponse approchée à une requête. Il est alors important de souligner qu'une conception mal réalisée peut engendrer la défaillance du système. Aussi, des conséquences néfastes peuvent se produire si les contraintes de temps ne sont pas respectées.

- **Interface utilisateur configurable** : Dans le but d'offrir un logiciel destiné à un vaste champ d'applications, nous avons opté pour un système avec interface utilisateur configurable. Dans un tel système, l'utilisateur peut reconfigurer l'interface client, selon ses besoins spécifiques. Par exemple, il peut reconfigurer l'interface client pour pouvoir surveiller un senseur de température sur un site et un senseur de pression sur un autre site.

- **À Plate-forme indépendante** : Le prototype que nous allons développer sera basé sur la technologie Java. Java est principalement un langage de programmation et une plate-forme d'exécution. Il se compose de concepts/mécanismes de multi-tâches, de gestion d'exceptions, et de sécurité, qui font de lui le langage de programmation approprié pour des développements robustes, stables et sûrs. L'environnement d'exécution Java se compose d'une machine virtuelle *(JVM)* responsable de charger et d'interpréter le byte code préalablement généré par le compilateur. De par son indépendance à l'infrastructure matérielle et logicielle sous-jacente, le byte code Java peut-être interprété sur toute plate-forme informatique pour laquelle une machine virtuelle Java est disponible. Cette indépendance est donc en soi un gage de portabilité des applications Java. Java comporte également des APIs (*Application Programming Interfaces*) qui fournissent aux développeurs de logiciels de nombreuses librairies de classes facilement utilisables.

L'architecture que nous proposons sera une innovation dans le domaine des systèmes d'acquisition de données dans la mesure où ses caractéristiques lui permettront un vaste champ d'applications industrielles.
Ce qui suit, est une présentation d'un survol sur les technologies sous-jacentes aux senseurs à fibres optiques.

2.4 Senseurs à fibres optiques

Un senseur à fibres optiques est un système capable de convertir une quantité physique variable en un signal optique modulé et de le décoder en signal électrique normalisé [2]. Pour des fins de mesures, le signal de sortie doit être calibré contre un étalon spécifique, pour devenir ainsi un senseur de mesure pour la même quantité physique. Les senseurs à fibres optiques sont issus de la rencontre de deux technologies : les fibres optiques (fibre sensible et fibre de connexion) et l'optoélectronique (laser à semi-conducteur et le photo détecteur optique). L'une et l'autre ont connu un développement extraordinaire ces deux dernières décennies, citons : l'amélioration des performances, la diminution des coûts et la miniaturisation. L'utilisation des senseurs à fibres optiques dans le domaine de l'ingénierie est maintenant très acceptée, ce qui a permis l'émergence de ces senseurs dans des champs d'applications principalement réservés aux systèmes électroniques.

2.4.1 Exemples de senseurs à fibres optiques

Il existe plusieurs types de senseurs à fibres optiques, parmi lesquels nous pouvons distinguer:

- ***Senseurs d'intensité*** : Utilise la modulation de l'intensité comme mécanisme de transmission de l'information mesurée sur la lumière guidée

dans la fibre. Ces senseurs sont très attractifs par leur simplicité et leur faible coût. Ils peuvent être développés par n'importe quel type de fibre optique et une source non cohérente. Par contre, quelques inconvénients sont à signaler concernant l'instabilité de l'intensité, les variations de la longueur d'onde de la source de lumière ainsi que les pertes engendrées par l'environnement de la fibre optique, les coupleurs et les connecteurs. La grandeur à mesurer (pression, température, etc.) module directement l'intensité de la lumière traversant la fibre optique. Cette modulation est la plus simple à réaliser. La lumière ainsi modulée est mesurée par un photo détecteur linéaire. Une méthode consiste à mesurer la quantité de lumière qui sort d'une fibre et rentre dans une autre. La mesure peut être faite par transmission, réflexion ou diffusion par le senseur mobile. On peut ainsi détecter la position (avec une précision du nanomètre), le déplacement, la vitesse et la pression. Dans d'autres cas, la source est à un bout de la fibre, le récepteur à l'autre bout et le transducteur au milieu. Dans de pareille situation, le phénomène physique permettant la mesure peut être d'origine mécanique (pertes par micro courbures), électro-optique, piezo-optique ou bien exploiter la magnéto-absorption [5].

- ***Senseurs interférométriques*** : La modulation de phase due à différentes causes (élévation de la température, pression, effet électro-optique, etc.) est mesurée indirectement. En effet, les fréquences optiques ne permettent pas de mesurer directement la phase d'une onde. Néanmoins par interférométrie, il est possible d'avoir accès à la phase et cela d'une manière très précise. Plusieurs interféromètres peuvent être utilisés : Mach-Zender, Michelson, Fabry-Perot ou Sagnac. Ces procédés sont utilisés dans des

gyroscopes, des jauges optiques pour mesurer les contraintes sur des ailes d'avions (impacts), la métrologie d'objets microscopiques (industries électroniques et nanotechnologies), séismologies, études acoustiques, analyse de croissance de dépôts, hydrophone, magnétomètre, ampèremètre, etc. Ce type de senseurs nécessite souvent des procédés de traitement des signaux sophistiqués. C'est pour cela qu'ils sont réservés aux besoins nécessitant une précision élevée [5].

- ***Senseurs polarimétriques*** : Les senseurs polarimétriques peuvent fonctionner suivant deux modes: propagations monomodes ou multimodes. La modulation de polarisation apparaît lors de la propagation d'une onde dans un milieu biréfringent. La biréfringence dans une fibre optique peut être attribuée à plusieurs facteurs : ellipticité de cœur, contrainte latérale interne, effort latéral externe, torsion, courbure, température et champ magnétique. Les effets physiques mis en oeuvre peuvent aussi être eléctro-optiques (effet Pockels) ou magnéto-optiques (effet Kerr). Par exemple, les effets couplés (Pockels et thermiques) ont permis de mesurer, avec une précision du dixième de degré Celsius, la température dans des réservoirs géothermiques situés à 2 km de profondeur [5]. Ces dispositifs sont simples à mettre en œuvre.

2.4.2 Cas d'application : Senseur interférométrique à fibre optique basé sur la lumière blanche (SILB)

La technologie des senseurs à fibres optiques a introduit le principe de la lumière blanche interférométrique (*LBI*) dans les années 1980 [3], mais les recherches dans ce domaine ne sont devenues notables que dans les années 1990. Le prix élevé des

équipements de décodage, la déficience des interféromètres de haute cohérence et spécialement leur incapacité de travailler avec les mesures de valeurs absolues ont permis de rediriger les efforts pour le développement des senseurs *LBI*. Bien que cette technologie soit relativement jeune, grâce à son faible coût et sa praticabilité elle est devenue la plus prometteuse parmi les autres technologies de senseurs à fibres optiques.

2.4.2.1 SILB basé sur les fibres hautement biréfringence (HB)

Il s'agit de mettre en application un nouveau type de senseur : le senseur interférométrique. Ce dernier est basé sur la lumière blanche et sur la transmission lumineuse par les fibres optiques hautement biréfringentes. Il a été breveté à l'UQO (Dr. W.Bock, W.Urbanczyk, US Patent No 5 517 022, obtenu le 14 mai 1996).

Le principe du SILB est illustré dans la Figure 2.2. La source de la lumière est une SLD avec une longueur d'onde centrale λ_0= 820nm et une longueur de cohérence de 15 μm [2]. Les fibres principales en entrée et en sortie sont raccordées à la tête du senseur selon des axes de polarisation alignés à 45°. La tête du senseur est elle même composée de deux longueurs égales de fibre HB (fibre de compensation et fibre sensible), raccordées avec les axes de polarisation à 90°. Cette opération permettra de donner à la sortie du système une valeur de déphasage proche de zéro. Les axes de polarisation à la sortie du senseur sont alignés à 45° par rapport aux axes de polarisation de la lentille L, l'analyseur A ainsi que la lentille CL sont aussi alignés à 45° par rapport à l'axe de polarisation du prisme WP.

Pour pouvoir délimiter le faisceau lumineux provenant de la source SLD, le signal de sortie, acheminé par la fibre principale est projeté sur la lentille L puis sur le prisme le Wollaston (WP). Les deux modes polarisés (selon x et y) du senseur sont spatialement séparés à travers le prisme WP par un angle α=2°. Après leur passage à travers l'anayseur A et la lentille cylindrique, les deux modes s'interfèrent.

Finalement le faisceau laser sera dirigé par la lentille CL sur la camera CCD, ce qui permettra d'enregistrer l'image interférométrique du signal de sortie (Figure 2.3) avec une résolution proche de 1024 pixels. Pour augmenter l'efficacité de notre système, nous utilisons un ensemble de quatre senseurs interforémetriques disposés en parallèle [3]. Chaque senseurs est composé de deux autre senseurs (senseur de pression et senseur de température). Le nombre de caméras CDD sera également augmenté pour pouvoir enregistrer les deux images interférométriques du signal de sortie.

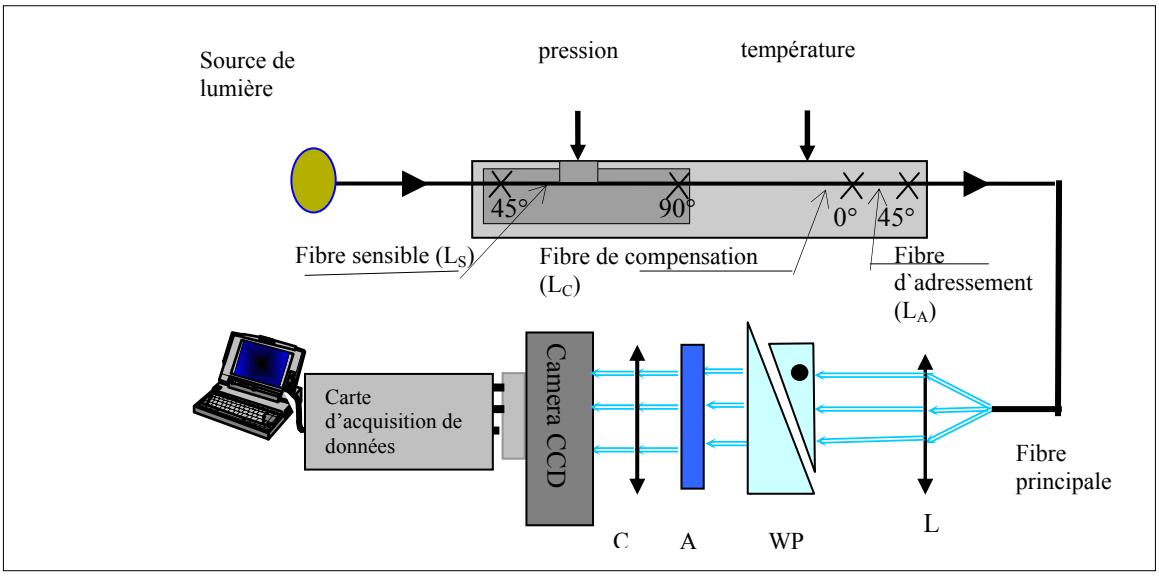

Figure 2.2 : Configuration du Système d'acquisition et du SILB basé sur les fibres HB

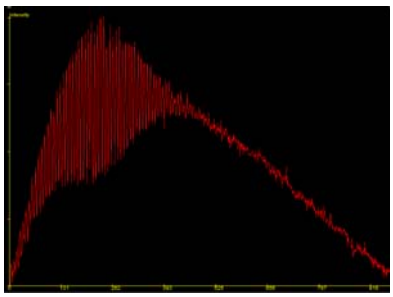

Figure 2.3 : Représentation de l'image interférométrique du signal de sortie

2.5 Conclusion

Dans ce chapitre, nous avons présenté l'état de l'art sur les systèmes d'acquisition de données ainsi que sur les technologies de senseurs à fibres optiques. En premier, Nous avons décrit les différents standards sur les systèmes d'acquisition de données. Ensuite, nous avons présenté et critiqué les systèmes actuels d'acquisition de données; leurs études nous a permis de mettre en lumière leurs faiblesses motivant ainsi nos objectifs de recherche. Enfin, nous avons fait un survol des technologies sous-jacentes aux senseurs à fibres optiques.

Chapitre 3

Vue d’ensemble et modélisation du système

Dans ce chapitre, nous présentons l’architecture globale du système distribué d’acquisition de données. En premier, nous allons faire la modélisation statique du système basée sur une analyse des besoins de l’utilisateur et de l’application. En second, nous présentons une étude détaillée sur les principaux modules du système.

3.1 Introduction

Pour le développement de l’architecture de notre système, nous nous sommes basés sur un modèle de type client/serveur [10, 4]. Ce choix est justifié par la nature même du système d’acquisition de données qui consiste principalement en une partie qui demande un service et une partie qui offre le service. Pour le développement et la réalisation de cette architecture nous allons nous inspirer de la méthode COMET (*Concurrent Object Modeling and architectural design*

mEThod) [8]. En effet, cette méthode de conception logicielle est très proche de la réalité de notre système, puisqu'elle est conçue principalement pour les systèmes distribués et temps réel. Par conséquent, avant de procéder à la décomposition du système, nous avons établi un modèle des besoins pour notre système. Le but derrière cette décomposition est de mieux cerner les limites du système, ces interaction avec le monde réel et ces besoins fonctionnels. Cette étude est présentée dans la section suivante.

3.2 Modélisation statique du système d'acquisition de données

La modélisation statique selon la méthode COMET doit comprendre les diagrammes des classes et les diagrammes des cas d'utilisation. Il s'agit de présenter la structure statique du système d'acquisition de données afin de lui associer une solution de conception dans le monde réel. Nous présentons en premier les diagrammes des cas d'utilisation les plus pertinents pour notre application. Nous présentons ensuite les diagrammes de classes pour l'application client et l'application serveur. Finalement, nous présentons une étude détaillée des principales composantes de ces deux applications.

3.2.1 Modèle des besoins

L'analyse des cas d'utilisation représente un outil inestimable pour la validation des attentes visées par la conception de notre système d'acquisition de données. Par cette étude, nous voulons nous concentrer sur les fonctionnalités essentielles du système, et présenter seulement les diagrammes de cas d'utilisation les plus

pertinents. D'une part, nous allons modéliser le diagramme de cas d'utilisation de l'application client « ClientAD » (Figure 3.1) [11], en mettant l'accent sur les opérations effectuées afin de pouvoir visualiser les résultats des acquisitions sur l'interface graphique utilisateur. D'autre part, nous présentons le diagramme de cas d'utilisation de l'application serveur « ServeurA » (Figure 3.2) pour le traitement d'une commande d'acquisition de données.

Pour le ClientAD, seulement trois acteurs sont en interaction avec le système: l'utilisateur, le ServeurAD et une horloge d'interface graphique.

- L'utilisateur peut principalement effectuer les opérations suivantes :
  - Établir une connexion avec le serveur.
    - Mettre en œuvre les mécanismes de sécurité : Initialiser le contexte SSL, choisir le protocole SSL utilisé et le mot de passe pour ouvrir les fichiers contenant la clé privée et la clé publique.
  - Exécuter les protocoles de communication.
  - Fermer une connexion.
  - Configurer l'interface graphique selon les besoins de l'expérience.

- Le serveur peut :
    - Envoyer les résultats.
    - Accepter les connexions

- L'horloge est en mesure de réactiver l'interface graphique pour chercher les nouvelles données et mettre à jour les résultats affichés.

Le serveurAD est en interaction avec trois acteurs : l'administrateur, le ClientAD et une horloge (timer). Afin d'assurer le bon fonctionnement de l'application, les cas d'utilisation suivants doivent être exécutés :

- Démarrer le serveur.
- Mettre en œuvre les mécanismes de sécurité.
- Exécuter les protocoles de l'application (réception des commandes 'client').
- Valider les commandes.
- Interpréter les commandes.
- Ordonnancer les commandes.
- Réaliser l'acquisition des données.
- Lire et écrire les données collectées.
- Envoyer les résultats au client approprié.
- Traiter les alarmes si nécessaires.

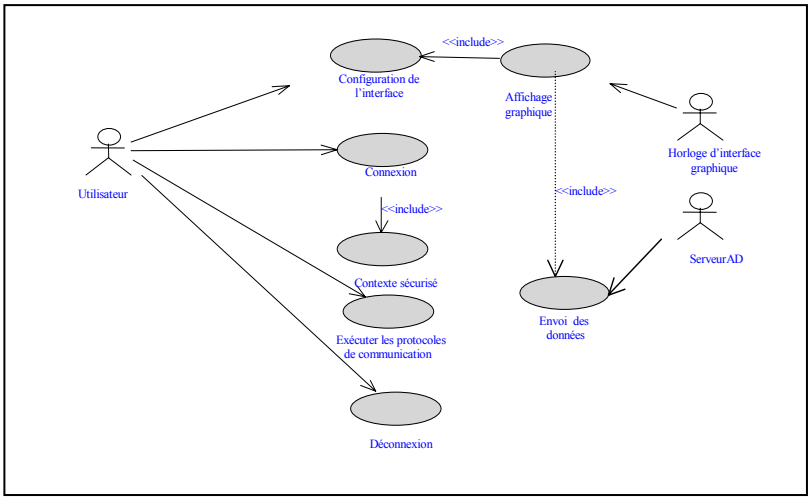

Figure 3.1 : Diagramme de cas d'utilisation du ClientAD

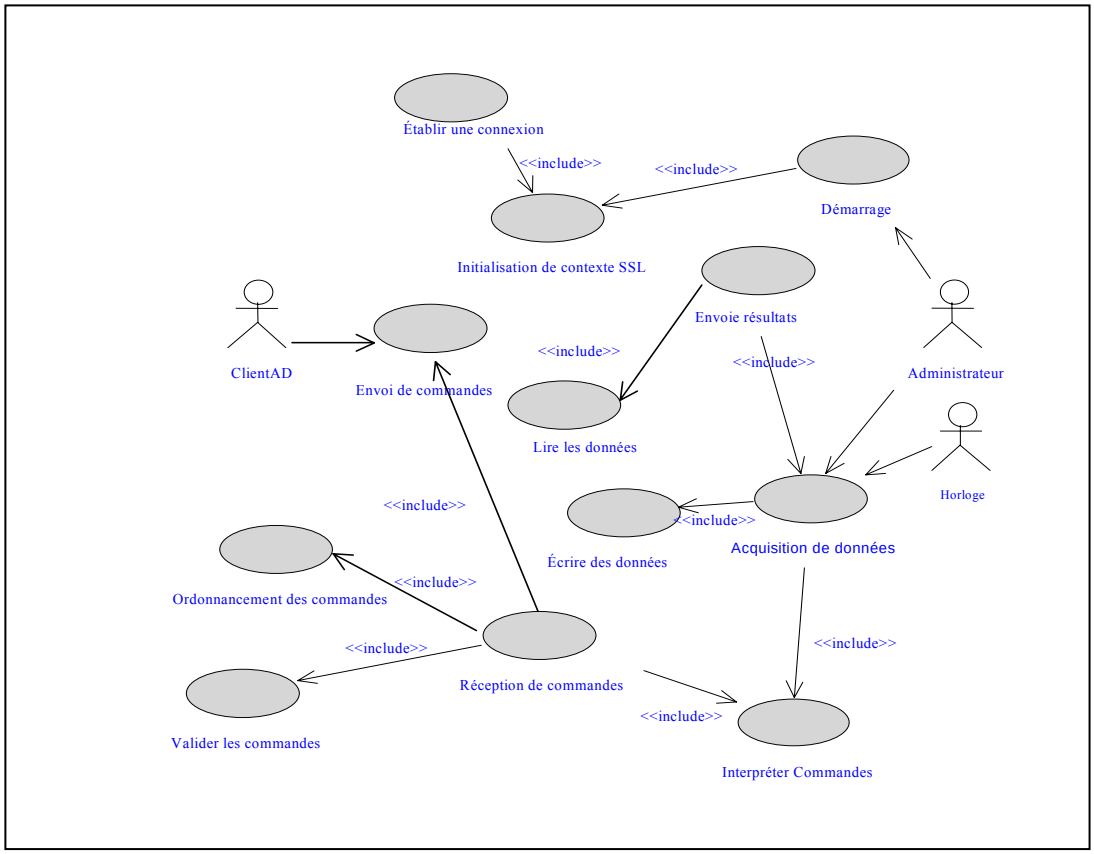

Figure 3.2 : Diagramme de cas d'utilisation du ServeurAD

Dans la section suivante, nous présentons la suite de notre modélisation. Il s'agit de la structure statique du système d'acquisition de données. Cette présentation décrit une solution de conception du système dans le monde réel et sera rehaussée par une description détaillée de l'architecture globale du système distribué d'acquisition de données.

3.2.2 Modèle statique

Bien que la structure statique du système défini en premier les classes d'objets dans un système et les relations entre ces classes, nous avons jugé qu'il serait préférable de décrire les modules principaux du système avant de présenter les classes constituant notre système. Cette façon de faire, nous permettra de présenter l'idée générale du fonctionnement interne du système d'acquisition de données et d'illustrer les interactions possibles entre les objets.

3.2.2.1 Architecture du serveur

L'application serveur doit principalement assurer un contexte de communication sécurisé pour les commandes 'client'. Elle doit être en mesure de collaborer avec plusieurs clients en même temps en leur assurant un service fiable et performant. Par conséquent, elle doit livrer la réponse appropriée dans un temps ne dépassant pas la date limite de la commande. Pour cela, nous avons décomposé la structure interne de l'application serveur en un ensemble de modules qui sont en interaction étroite les uns avec les autres (Figure 3.3). Cette structure est définie comme suit :

- Le gestionnaire d'information (*GI*) : Il gère les informations échangées entre les composantes du serveur. Le *GI* agit comme une boîte postale, utilisée pour la réception et l'envoi des messages de données ou de commandes des autres modules. Ce module est principalement composé du sous module *ServConnectionMng* qui est responsable de la gestion d'une connexion avec un client. Il est au cœur même du serveur. Chaque instanciation de cette classe résultera en la création d'un thread qui prendra

en charge la connexion avec le client. De plus, tout transit et tout dépôt d'informations doivent passer par le *ServConnectionMng*. En effet, la réception des commandes 'client', leur validation et leur interprétation devront tous être acheminés par ce sous module.

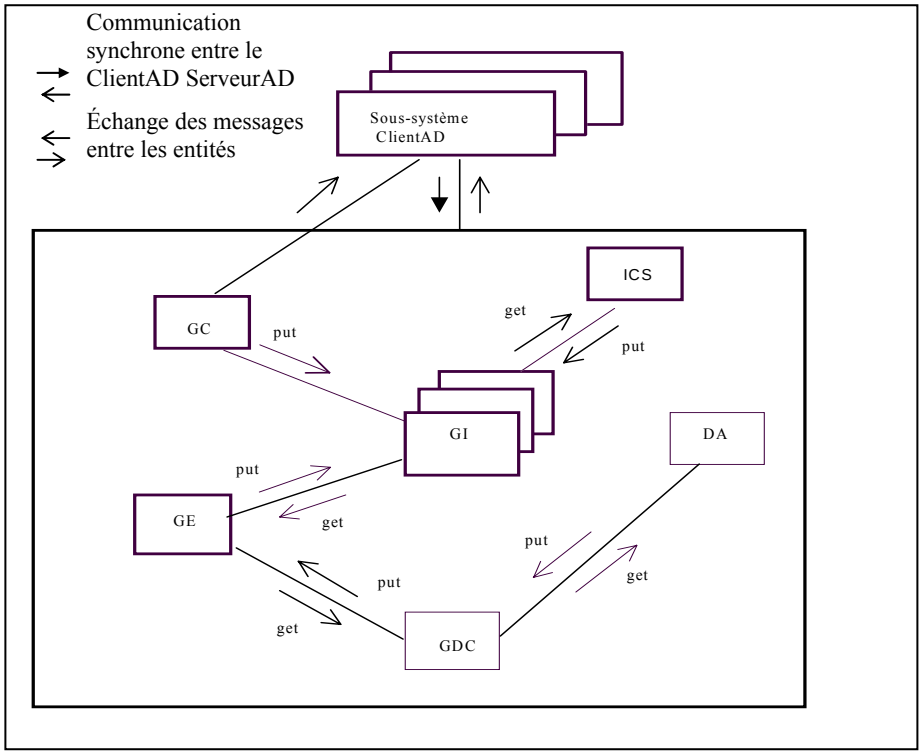

Figure 3.3 : Décomposition modulaire du ServeurAD.

- Le gestionnaire de connexions (*GC*) : Il est responsable de la gestion des connexions clients. Une fois q'une connexion est établi avec un client, il délègue sa gestion au sous module ServConnectionMng. Le *GC* est aussi

responsable de l'établissement des communications sécurisées avec les applications clients. Ce module englobe deux sous modules :

- ServeurSSL : Offre toutes les fonctionnalités SSL du serveur. Ce sous module est responsable de l'établissement de la relation de confiance avec les clients et de la création d'un contexte de communication sécurisée en SSL (voir chapitre 6).
- IdentificateurConnexion : Sert à identifier les connexions clients en enregistrant leurs paramètres « adresse IP et numéro de port ». Cette façon de faire permettra d'acheminer les résultats des requêtes au client approprié.

- L'interpréteur des commandes du serveur (*ICS*) : Chargé de l'interprétation et de la présentation des commandes sous un format standard. Le *ICS* extrait et traduit les paramètres de la commande afin de répondre correctement aux requêtes 'client'. Pour le développement des composantes de ce module, nous avons utilisé JavaCC : une nouvelle technologie développée par Sun Microsystems. Dans ce qui suit, nous présentons une brève description des sous modules du *ICS* (une étude détaillée sera présentée dans le chapitre 7) :

  - ServerParser : Permet d'établir un format standard pour l'ensemble des commandes reçues et vérifie qu'elles satisfont une certaine syntaxe.

- CommandsInterpreter : Est responsable du décodage des commandes 'client' une fois reçues. Ce sous module permet d'interpréter la commande et d'extraire les paramètres pertinents à son exécution.

- Le gestionnaire des exécutions (*GE*) : Ce module maintient une file d'attente pour les requêtes 'client'. Il leur attribue des priorités et gère leur ordre d'exécution. Il se compose principalement d'une file d'attente pour les commandes 'client' et d'un ordonnanceur qui s'occupe de la planification de la prochaine commande à exécuter. L'ordonnanceur utilise une file d'attente par priorité pour les commandes 'client'. De plus, il est doté d'une horloge qui sert à réveiller la prochaine requête à exécuter.

- Le module d'acquisition de données (*DA*) : Ce module permet la communication entre la partie matérielle (la carte d'acquisition de données et les senseurs à fibres optiques) et la partie logicielle (système d'acquisition de données). Il prend en charge les procédures de lecture et d'écriture des données collectées (voir chapitre 5). Il se compose principalement du sous module *SensorDataCollection,* responsable direct du processus permettant le déclenchement des différentes acquisitions de données. Il est configuré de façon à permettre au client d'avoir un contrôle à distance sur les senseurs et l'instrument de mesure. Ce sous module constitue la frontière entre la partie logicielle et la partie matérielle de notre système. Il est capable d'interroger l'instrument de mesure et de retourner les lectures une fois réalisées.

- Le gestionnaire des données collectées (*GDC*) : Ce module assure l'entreposage des données collectées à partir des senseurs. Après que le module *DA* ait fini de collecter les données, le *GDC* procède à la lecture, à l'échantillonnage et finalement à la sauvegarde des données. Il est principalement composé du sous module *SyncCharedMemory,* responsable principal de la synchronisation des accès aux données. En effet, le risque de conflit, lors de l'accès simultané à la ressource partagée (sous module *SensorDataCollection*), est maximal quand les threads réalisent des écritures. Pour éviter cette situation nous avons posé un verrou du début à la fin de l'acquisition sur l'objet qui réalise les lectures. Toute autre opération simultanée est alors interdite. Une fois l'acquisition complétée, touts les threads en attente seront notifiés.

Dans ce qui suit, nous présentons le diagramme des classes de l'application serveur (voir la figure 3.4). Par ce diagramme, nous voulons essentiellement montrer les relations entres les différentes classes. L'aspect interaction entre les objets et le comportement dynamique des classes sera approfondi dans le prochain chapitre.

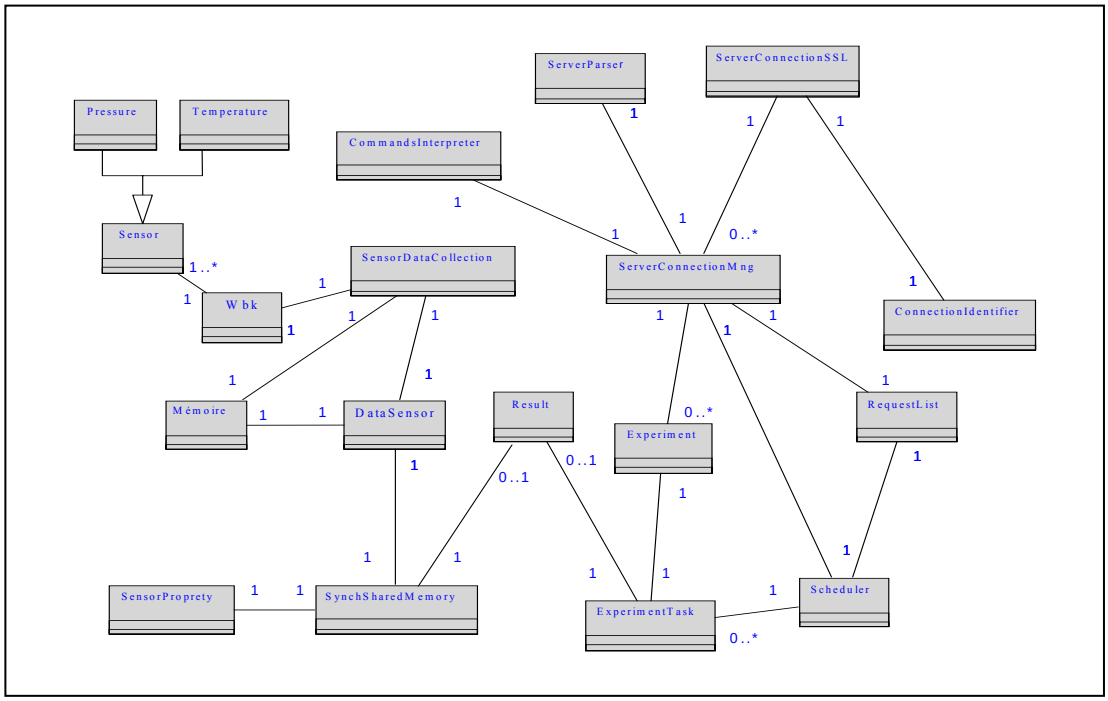

Figure 3.4 : Diagramme des classes pour le ServeurAD

3.2.2.2 Architecture client

L'application client doit être en mesure d'établir une communication sécurisée a travers le réseau. Elle doit aussi être capable de fournir à l'utilisateur un panneau virtuel qui lui permet le contrôle du système de mesure à distance. En effet, l'utilisateur sera en mesure de configurer le senseur à tester et d'exécuter des protocoles d'application pour communiquer avec le serveur (voir le chapitre 7). La Figure 3.5 présente les principaux modules de l'architecture client.

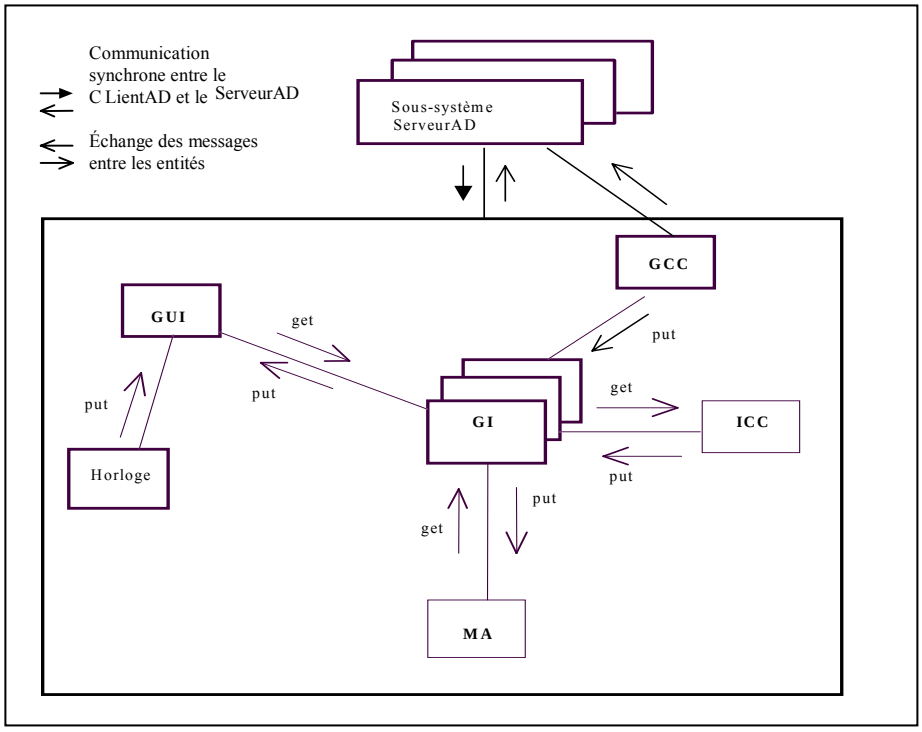

Figure 3.5 : Décomposition modulaire du ClientAD.

- Interface graphique de l'utilisateur (*GUI*) : Ce module regroupe toutes les composantes graphiques de l'application client. Une de ces fonctionnalités est de fournir au client un panneau virtuel qui permet le control à distance de l'instrument de mesure en plus de la configuration des paramètres des senseurs à surveiller. Ce module permet aussi l'affichage des résultats une fois reçus (voir chapitre 7 pour une étude plus approfondie).

- Gestionnaire des communications clients (*GCC*) : Ce module s'occupe des communications avec les serveurs. Il a la possibilité d'établir une

connexion avec un serveur et de s'assurer de son identité. Il s'occupe de la négociation des clés de session et de l'initialisation d'un contexte « SSL » pour une communication sécurisée avec le serveur.

- le gestionnaire d'information (*GI*) : Ce module gère les informations échangées entre les composantes de l'application client. Il récupère la commande formulée par le client, l'envoie au serveur et une fois les réponses prêtes, il les achemine vers le composant graphique approprié.

- L'interpréteur des commandes 'client' (*ICC*) : Il est responsable de l'interprétation et de la présentation des commandes envoyées. Il est composé de deux sous modules, un pour la génération du parseur et l'autre pour la vérification de la syntaxe de la commande à envoyer.

- L'analyseur mathématique (*MA*) : Permet l'interprétation et la présentation finale des données selon l'algorithme approprié. Le diagramme de classe suivant (Figure 3.6) montre les principales classes de notre modèle.

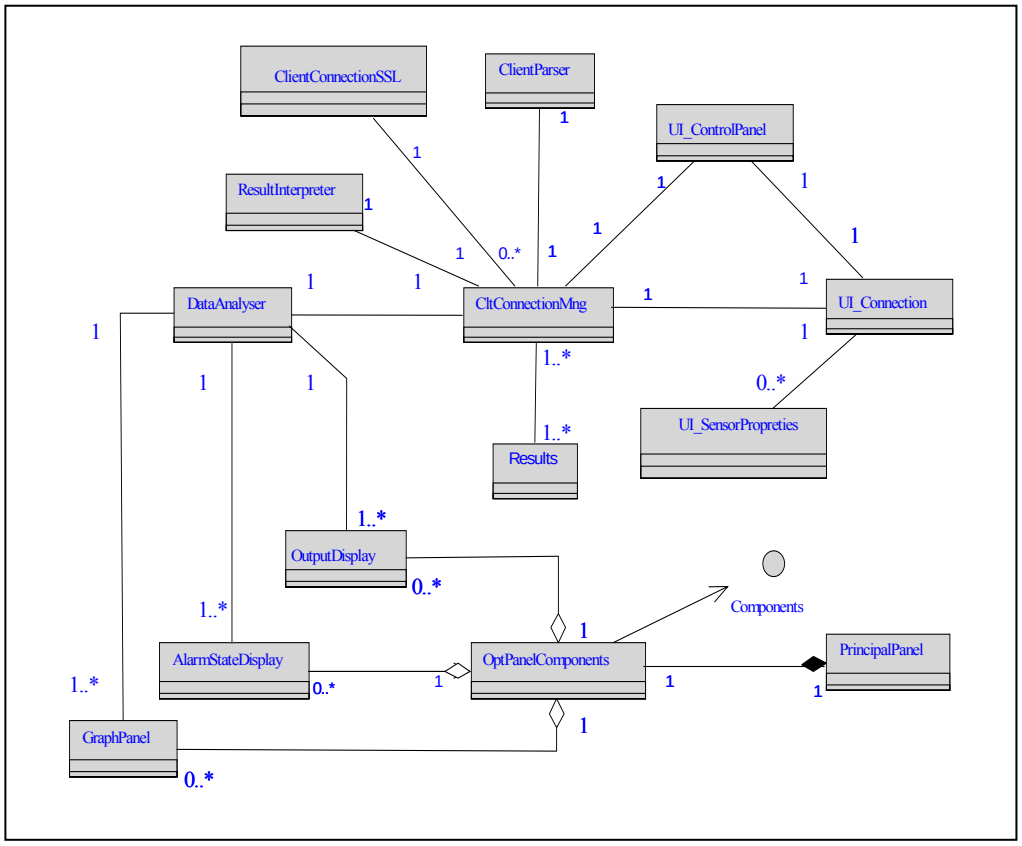

Figure 3.6 : Diagramme de classe pour le ClientAD

3.3 Conclusion

Dans ce chapitre, nous avons présenté les principaux modules de notre système. Nous avons décrit l'architecture logicielle globale du système d'acquisition de données à distance. Pour le moment, seulement la structure générale de cette architecture est présentée, dans le chapitre suivant, une étude plus approfondie sera présentée afin d'illustrer la nature des communications et les interactions

possibles entre les différents objets constituant le système d'acquisition de données.

Chapitre 4

Comportement dynamique du système

Dans ce chapitre, nous présentons l'aspect dynamique du système d'acquisition de données. Nous allons faire en premier, la modélisation dynamique de l'application serveur suivie par celle de l'application client. Dans les deux cas, l'accent est mis sur la nature des communications et les interactions possibles entres les différents objets constituant les deux applications.

4.1 Introduction

L'objectif principal de la modélisation dynamique du système d'acquisition de données est de définir les interactions possibles entre les différents objets ainsi que leur nature. En général, les diagrammes de collaboration sont assez représentatifs pour ce type de modélisation. En effet, dans la section suivante, nous présentons

les diagrammes de collaboration les plus pertinents, autant pour l'application client que pour l'application serveur.

4.2 Structuration du système

Puisque l'architecture du système d'acquisition de données est basée sur un modèle client/serveur, la décision de le structurer en un sous système client « ClientAD » et un sous système serveur « ServeurAD » est évidente (voir à la Figure 4.1). Le principe de fonctionnement du système d'acquisition de données repose essentiellement sur trois étapes majeures :

- Le client envoie une commande d'acquisition au serveur.
- Le serveur lit la commande, la dépose dans une file d'attente en lui accordant un ordre d'exécution selon sa priorité.
- Le serveur exécute la commande et envoie la réponse au client.

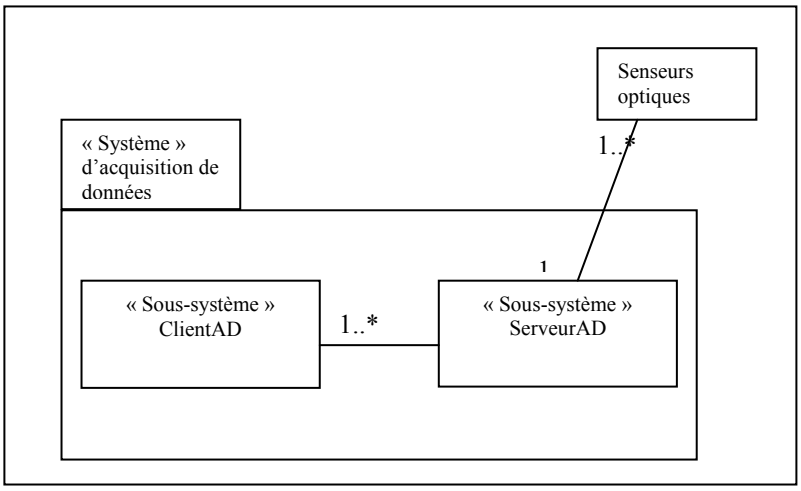

Figure 4.1 : Vue globale du système d'acquisition de données

Dans le cas de notre application, plusieurs clients peuvent interagir avec plusieurs serveurs et l'application serveur doit être en mesure de gérer plusieurs messages en provenance des différents clients (voir à la Figure 4.2).

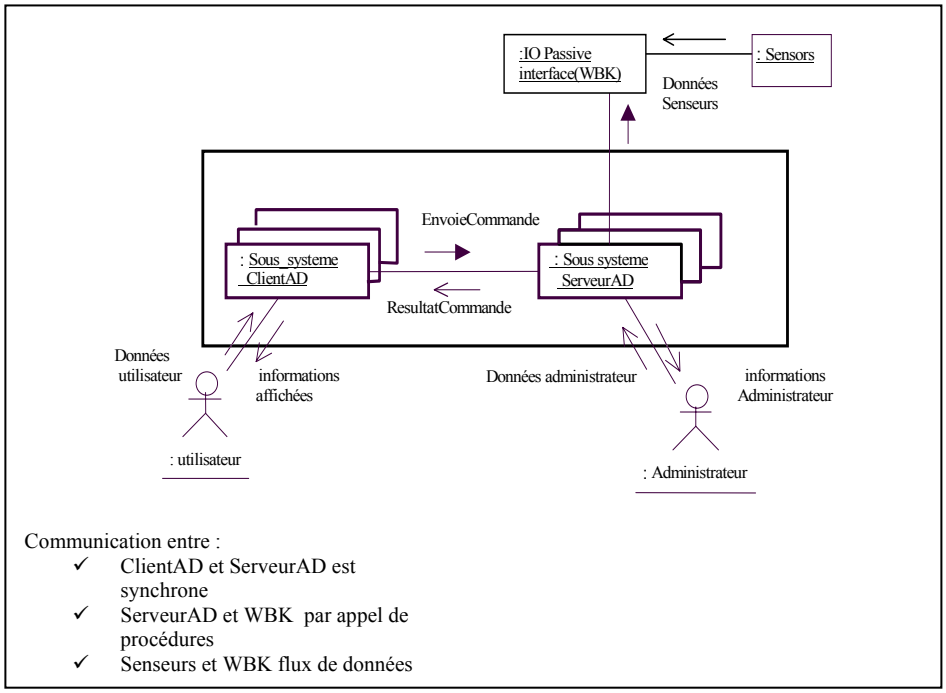

Figure 4.2 : Diagramme de collaboration du système d'acquisition de données

La communication entre les deux sous systèmes étant synchrone, le processus client responsable de la gestion de cette communication avec le serveur (session), reste en attente d'une réponse du système avant de continuer son traitement.

Du coté serveur, l'acquisition de données se fait par l'interrogation de l'interface d'entrée/sortie (voir chapitre 5). Cette interface représente la frontière entre la partie matérielle et la partie logicielle de notre système. Son invocation se fait sur

demande (horloge) et permet de contacter le senseur approprié, de collecter les données et de faire la sauvegarde.

4.2.1 Sous système ServeurAD

Dans cette section, nous présentons le comportement dynamique du ServeurAD (Figure 4.3). Pour réaliser cette étude nous nous sommes essentiellement basés sur le diagramme de cas d'utilisation présenté à la Figure (3.3). De plus, cette étude sera principalement concentrée sur les messages échangés et sur la nature des communications et les interactions possibles entre les différents objets.

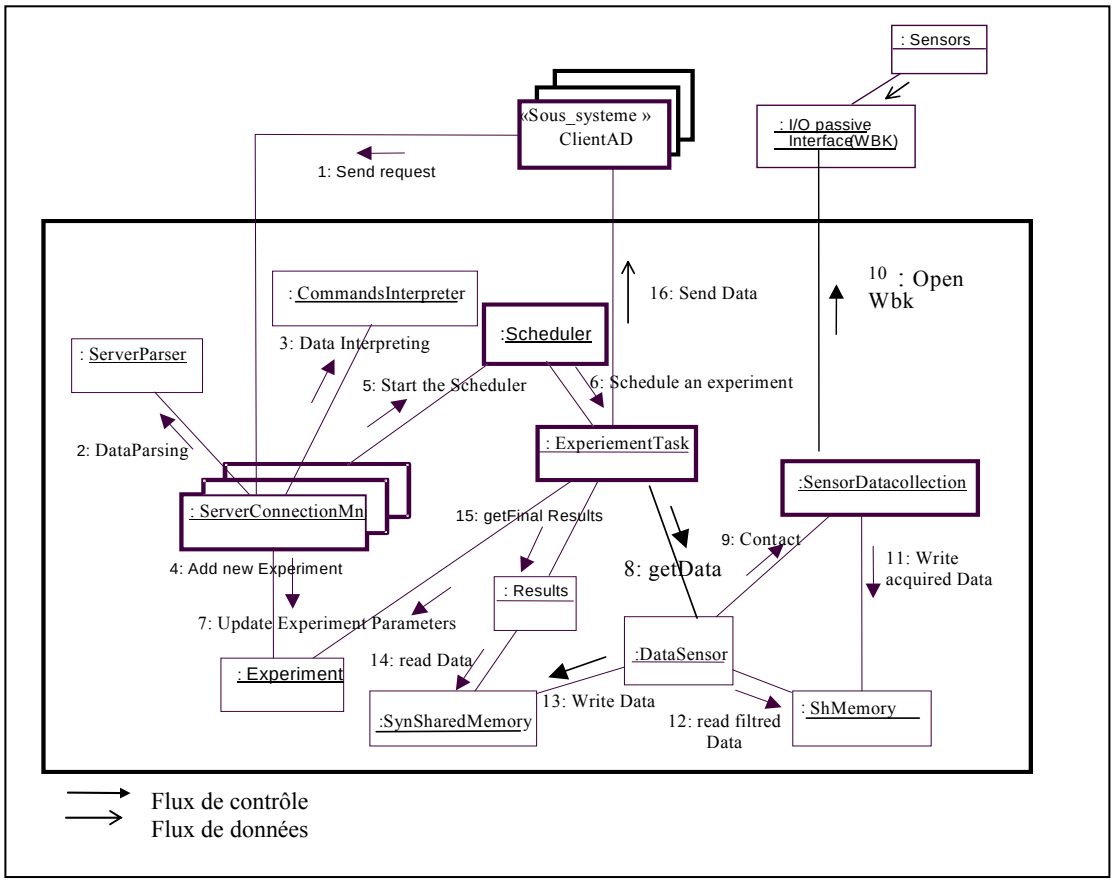

Figure 4.3 : Diagramme de collaboration pour la réception des messages

Le premier objet participant dans ce diagramme est le *ServerConnectionMng.* Cet objet s'occupe de la gestion des commandes 'client', en leurs attribuant les traitements nécessaires. Une fois la commande reçue, *ServerConnectionMng* fait appel :

- à l'objet parseur *ServerParser* pour la vérification de la syntaxe de la commande.
- à l'objet *CommandsInterpreter* pour l'extraction des paramètres de la commande
- à l'objet *Experiment* afin de l'initialiser avec les paramètres de la commande.

De plus, le *ServerConnectionMng* est responsable du démarrage de l'ordonnanceur *Scheduler,* ce dernier maintient une file d'attente et utilise un algorithme d'ordonnancement pour planifier l'ordre d'exécution des requêtes 'client'.

L'algorithme utilisé par le *Scheduler* accorde dynamiquement des priorités d'exécution selon la périodicité de chaque événement. Plus la période est courte, plus la priorité est élevée. Parmi les algorithmes de tri connus, nous avons choisi l'algorithme de tri par fusion. Dans la majorité des cas, le tri rapide est préféré au tri fusion. Cependant, le choix du pivot dans le tri rapide est totalement aléatoire, ce qui rend son temps d'exécution imprévisible. Avec le tri fusion, il n'y a aucun facteur aléatoire et c'est pour cette raison qu'il a été choisi comme algorithme d'ordonnancement pour le *Scheduler.* Pour chaque événement d'acquisition de données, les paramètres de la commande véhiculés par cet événement, seront affectés à l'objet *Experiment*. Ce dernier, sera associé à une horloge pour assurer

l'exactitude de son temps d'exécution. Cette association est encapsulée par l'objet *ExperimentTask.* Ce dernier, hérite de la classe *TimerTask* et doit être redémarré par le *Scheduler.* Cette opération va automatiquement démarrer la prochaine itération de l'objet *Experiment* au bon moment pour son exécution. Cette dernière étape va enchaîner une suite de traitements tels que : l'activation de l'interface passive *WBK*, l'acquisition des données, la récupération des données et leur envoi au client approprié. L'acquisition de données se fait par l'interrogation de l'interface passive *WBK* (périphérique d'entrée/sortie). Cette interface représente la frontière entre la partie matérielle et la partie logicielle de notre système. Son invocation se fait sur demande (horloge) et permet en premier de contacter le senseur approprié, en second, de collecter les données et finalement de faire la sauvegarde.

Pour éviter le problème de l'exclusion mutuelle, nous avons utilisé le concept de moniteur. En effet, *ShMemory* et *SynSharedMemory* sont deux objets du type « *information hiding object* » utilisés pour l'encapsulation des données collectées. Ils permettent l'accès synchronisé aux données grâce à des appels d'opérations pour les lectures et les écritures :

- *ShMemory :* Gère l'accès aux données brutes des senseurs. Les opérations d'écriture et de lecture sont respectivement appelées par deux objets : *SensorDataCollection* et *DataSensor.*

- *SynSharedMemory :* Gère l'accès aux données filtrées des senseurs. Les opérations d'écriture et de lecture sont respectivement appelées par deux objets : *DataSensor* et *Results.*

4.2.2 Sous système ClientAD

Pour l'application client, le scénario de réception, de traitement et d'interprétation des données est le scénario le plus significatif pour la modélisation dynamique du *ClientAD*. Effectivement, la réception des résultats des commandes, leurs transformations et leur affichage représente le cas le plus intéressant à modéliser puisqu'il comporte le plus grand nombre d'objets en interaction.

Le scénario établi par ce diagramme est assez simple (voir à la Figure 4.4). Le premier objet participant dans ce diagramme est le *ClientConnectionMng* qui, une fois reçu les résultats du serveur, les interprète selon la commande envoyée. Ensuite, chaque objet graphique récupérera les résultats transformés par l'objet *DataAnalyser* et les affichera. L'interface utilisateur procédera à un rafraîchissement toutes les secondes pour pouvoir récupérer les nouveaux résultats.

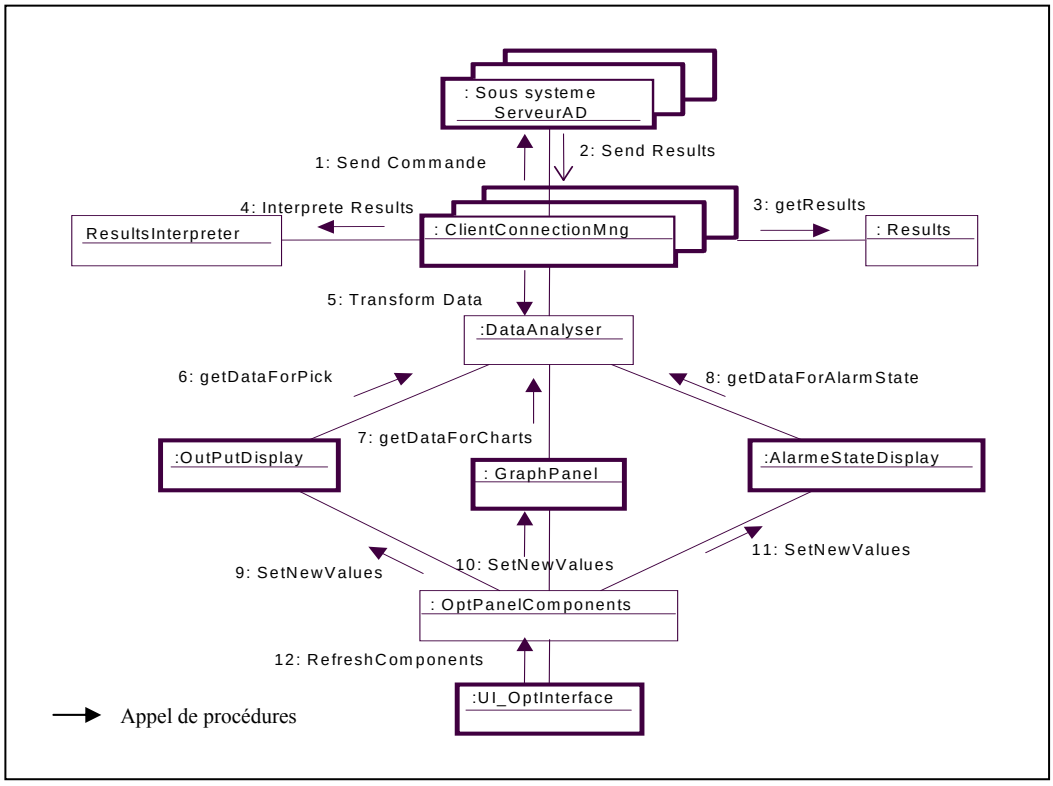

Figure 4.4 : diagramme de collaboration pour la réception des résultats

4.3 Conclusion

Dans ce chapitre, nous avons présenté la modélisation dynamique du système d'acquisition de données. Aussi, selon les scénarios d'utilisations les plus pertinents, nous avons décrit les comportements et les interactions des objets dans chaque sous-système. Cette modélisation a été soutenue par les concepts et mécanismes utilisés pour la conception des systèmes concurrents et temps réel.

Chapitre 5

Acquisition de données

Dans ce chapitre, nous présentons notre méthodologie pour l'acquisition de données. Nous allons faire la description du module responsable de cette fonction et nous discutons les différentes étapes suivies pour son développement.

5.1 Procédé d'acquisition de données

Dans notre système, le serveur n'effectue les acquisitions de données que sur demande. Toutefois, avant que les demandes client ne soient desservies, le serveur doit réaliser les étapes suivantes :

1. Extraction des paramètres des commandes : En effet, pour chaque opération d'acquisition, le serveur doit extraire un certain nombre de paramètres qui lui permettront d'interroger correctement le périphérique d'acquisition. Ces paramètres sont véhiculés par la

commande 'client' et fournissent des données telles que le nom du senseur, la fréquence des lectures, la périodicité des acquisitions, etc.

2. Temps d'activation : Chaque opération d'acquisition doit être associée à une horloge afin de préciser le temps de son exécution et de sa prochaine itération s'il y a lieu. Le temps d'exécution, ne doit pas dépasser la date limite définie pour effectuer l'acquisition des données.

3. Contacter le périphérique d'acquisition : Pour que la collecte des données se déroule correctement, il faut que le périphérique d'acquisition soit libre au moment de son invocation. Dans ce cas, les opérations suivantes seront entamées (voir Figure 5.1) :

    i. Ouverture du périphérique
    ii. Initialisation des paramètres d'acquisition
    iii. Collecte des données
    iv. Fermeture du périphérique
    v. Sauvegarde des données collectées

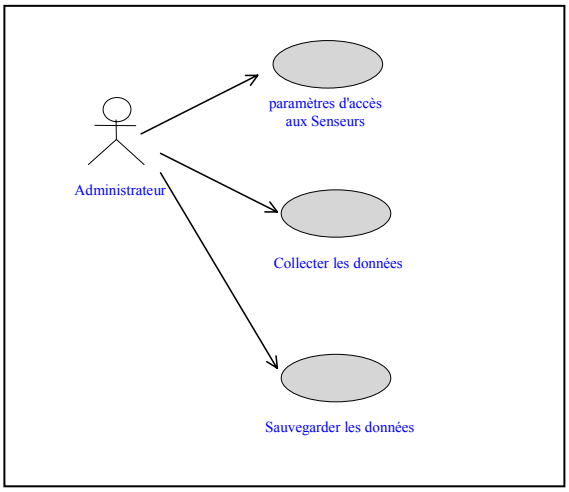

Figure 5.1 : Diagramme de cas d'utilisation pour l'acquisition de données

5.2 Développement du module d'acquisition

Le module d'acquisition de données est d'une importance capitale dans notre système. Ce module permet en premier, d'établir la communication entre la partie logicielle et la partie matérielle. En second, de prendre en charge les procédures d'acquisition et de sauvegarde des données. Dans notre système, la partie matérielle est représentée par le périphérique d'acquisition et les senseurs à fibres optiques (voir Figure 5.2).

Le module d'acquisition est principalement composé des sous modules suivants :

- *SensorDataCollection* : Ce sous module est le responsable direct de l'interrogation du périphérique matériel du système (*I/O passive Interface*).

Il permet : en premier, d'activer le périphérique et d'initialiser ses paramètres. En second, d'effectuer l'acquisition de données. Enfin, de fermer le périphérique et de sauvegarder les données dans la mémoire partagée *ShMemory* (voir Figures 5.3 et 5.4).

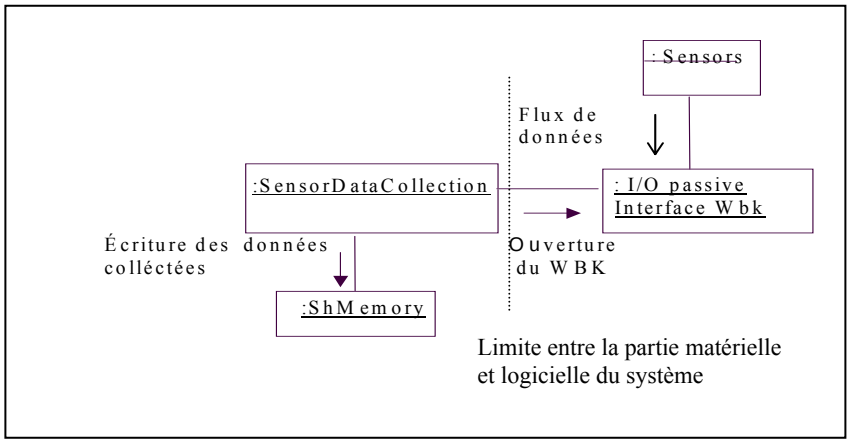

Figure 5.2 : Module d'acquisition

- *I/O passive interface (WBK)* : C'est une interface passive d'entrée/sortie, qui permet de recevoir et de convertir (analogique/numérique) les signaux émis par les senseurs optiques. Elle comporte plusieurs canaux sur lesquels le flux de données est transmis du senseur optique vers un tampon pour le stockage. Cette interface est passive (*time triggered*), son invocation se fait sur demande (horloge) et sa fermeture est effectuée une fois que le nombre de lectures exigées est réalisé. L'interface WBK utilise un tampon linéaire basé sur une file d'attente de type FIFO pour le stockage des données collectées (Figure 5.3).

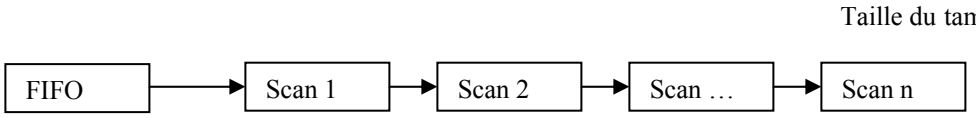

Figure 5.3 : Mode de stockage des données.

L'initialisation de l'interface WBK est gouvernée par les fonctions suivantes :

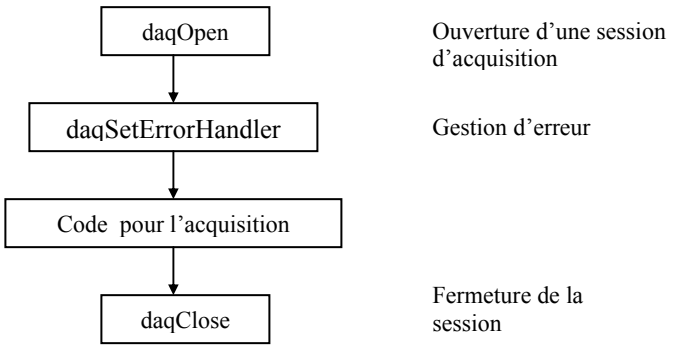

Figure 5.4 : Initialisation du périphérique matériel WBK.

Dans ce qui suit, nous présentons les principales étapes et fonctions utilisées afin de mener à terme une acquisition des données.

5.3 Étapes d'acquisition

Chaque opération d'acquisition peut être décomposée en six étapes de base:

- Configuration des canaux : Spécifier pour chaque senseur le canal utilisé pour l'acheminement du flux de données. Cette opération est réalisée par la fonction *daqAdcTransferSetBuffer* qui définit principalement la taille du tampon utilisé pour entreposer les données une fois collectées (pour les détails des fonctions, voir Figure 5.5)

- Configuration des événements d'acquisition : Définir les conditions de démarrage et d'arrêt de l'acquisition. La fonction *daqSetTriggerEvent* prend en compte le nombre de lecture à effectuer, le nombre de canaux impliqués, l'intensité du signal utilisé, etc.

- Définir le taux d'acquisition : La fonction *daqAdcSetRate* spécifie la vitesse à laquelle les canaux vont être scannés. La valeur de la vitesse est un des paramètres figurant dans les commandes d'acquisition des clients.

- Définir le mode de stockage des données : Il s'agit de définir le type du tampon utilisé pour le stockage des données collectées. Dans notre cas, nous utilisons un tampon linéaire d'une taille égale au nombre de lectures demandées. La fonction *daqAdcTransferSetBuffer* nous permet de spécifier ces informations.

- Commencer le transfert : Le transfert des données commence une fois que la fonction *daqAdcTransferStart* est appelée.

- Réception des données : Les données collectées sont sauvegardées dans le tampon déjà défini, par l'invocation de la fonction de transfert de données *daqAdcTransferBufData.*

```
/* Si l'initialisation de la carte d'acquisition se fait sans problème, handle
returnera 0 . WaveBook0 est le nom donné au périphérique matériel, représenté
dans la figure (5.2) par l'objet I/O passive Interface */
handle = wvbk.daqOpen("WaveBook0");

/*Si la communication avec la partie matérielle réussi, en commence une série
d'initialisation telle que : définir le nombre de senseurs impliqué dans cette
expérience, le nombre de lectures à faire, le mode d'acquisition (continue ou
non), le nombre de canaux impliqués, etc. */
wvbk.daqAdcSetAcq(handle,wvbk.DaqAdcAcqMode.DaamNShot,0,ScanCount);
wvbk.daqAdcSetScan(handle, Channels, Gains, Flags, ChanCount);

/* permet d'établir la fréquence à laquelle les lectures seront faites */
 int rateState = wvbk.DaqAdcAcqState.DaasPostTrig;

/* permet d'établir le mode de fréquence */
 int rateMode = wvbk.DaqAdcRateMode.DarmFrequency;

/* Ici commence la procédure d'acquisition des données selon les paramètres
initialisés*/
wvbk.daqAdcSetRate(handle,rateMode,rateState,ScanRate,ActualScanRate);
 wvbk.daqAdcTransferSetBuffer(handle, null, ScanCount, transmask);

/* Établir les conditions de fin d'acquisition: la fin est signalée une fois le
nombre de lectures voulu est réalisé */
wvbk.daqSetTriggerEvent(handle,
wvbk.DaqAdcTriggerSource.DatsScanCount,0, 0, 0, 0, 0, 0f, 0f,
wvbk.DaqTriggerEvent.DaqStopEvent);

/* commencement de l'acquisition */
wvbk.daqAdcTransferStart(handle);
wvbk.daqAdcTransferBufData(handle,rawbuf,ScanCount,xfermask,retCount);
wvbk.daqAdcTransferStop(handle);

/* fermeture du périphérique */
wvbk.daqClose(handle);
```

Figure 5.5 : Étapes principales pour l'acquisition de données.

5.3 Choix du langage de programmation

En général, pour implanter ce type de module, le langage C est considéré comme le choix naturel. En effet, C permet un contrôle étroit de la partie matérielle. Toutefois, l'utilisation de Java permet de produire un code plus propre dû à des caractéristiques telles que les exceptions, le typage, etc. De plus, avoir une interface de communication moitié Java moitié C entre la partie matérielle et la partie logicielle, rendra les tâches de maintenance et de documentation du code plus ardues. Avec Java, nous avons obtenu une base de code cohérente et manipulable par un seul ensemble d'outils.

5.4 Conclusion

Dans ce chapitre, nous avons présenté le module d'acquisition de données de notre système. Nous avons décrit son architecture et ces principales interactions. L'utilisation de Java pour l'implantation de ce module nous a permis le développement d'un système complètement portable et facile à maintenir.

Chapitre 6

Sécurité

Dans ce chapitre, nous présentons les mécanismes d'authentification sous-jacent à notre système d'acquisition de données. En premier, nous présentons le protocole SSL (Secure Socket Layer) utilisé pour l'encryptage des données transmises à travers le réseau. En second, nous présentons les mécanismes d'authentification et les algorithmes assurant la confidentialité des données dans notre système. Finalement, nous allons décrire, en détail, les sous modules responsables des communications sécurisées.

6.1 Généralité

Le réseau Internet est un moyen puissant de transmission et d'échange d'informations. Cependant, sa dimension publique peut permettre à des intrus

d'intercepter et de modifier les données transmises. En effet, afin de garantir un échange sécuritaire et d'éviter tout risque d'interception des données à l'insu des parties communicantes, il est très important d'implanter des mécanismes de sécurité permettant l'authentification, la confidentialité et l'intégrité des données transmises sur le réseau public.

Le protocole SSL (***S**ecure **S**ocket **L**ayer)*, conçu à l'origine par Netscape en collaboration avec *Mastercard*, *Bank of America*, *MCI* et *Silicon Graphics,* est un protocole qui repose sur un procédé de cryptographie par clé publique afin de garantir la sécurité des transmissions de données sur Internet.

Le système SSL est indépendant du protocole utilisé. Cela signifie qu'il peut aussi bien sécuriser des transactions faites sur le Web par le protocole HTTP que des connexions via le protocole FTP. En effet, SSL est une couche supplémentaire située entre la couche application et la couche transport du modèle TCP/IP (Figure 6.1). SSL définit un protocole de négociation et de gestion de clé de session pour établir une connexion privée. De plus, l'identité des extrémités peut être authentifiée. SSL emploie différents algorithmes de chiffrement tels que RSA (Rivest-Shamir-Adleman), DES (Data Encryption Standard), EDH (Error Detecting and Handling), etc. Les fonctionnalités principales de SSL [17] sont :

- L'authentification du serveur et du client,
- Le cryptage des données dans le but de préserver la confidentialité des communications.

En Java, les mécanismes de sécurité des données sont prises en charge par le paquetage JSSE (*Java Secure Socket Extension*) en liaison avec le JRE (*Java Run-Time Environment*). Le paquetage JSSE est une librairie qui contient les outils nécessaires (sockets SSL) permettant de réaliser des communications sécurisées entre une application client et une application serveur développées en Java. Pour notre application, nous avons mis en application ces mécanismes de sécurité. Les détails de cette opération sont présentés dans la section suivante.

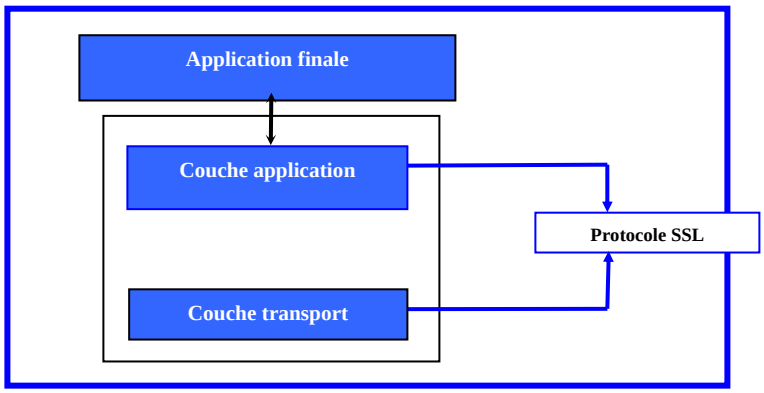

Figure 6.1 : Place de SSL dans le modèle TCP/IP

6.2 Implantation des mécanismes de sécurité

6.2.1 Authentification

Pour l'authentification du serveur, nous avons utilisé les concepts de certificat tel qu'établi par Netscape. L'identité du serveur est vérifiable seulement si ce dernier détient un certificat signé par une autorité publique ou par une autorité privée

« auto signé ». Si l'application client n'est pas en mesure de valider le certificat du serveur, elle peut procéder à l'affichage du condensé « empreinte » du certificat du serveur. Cette alternative permettra de contacter l'administrateur du serveur et de demander une confirmation du condensé ou, tout simplement, d'accepter une connexion non authentifiée avec le serveur.

6.2.2 Confidentialité

Une fois le serveur authentifié, les deux parties doivent négocier leur clé secrète « clé de session ». La confidentialité, lors de l'échange d'une clé de session est réalisée par l'utilisation de la cryptographie symétrique. Ce mécanisme de sécurité permet d'assurer l'intégrité et la confidentialité des données transmises contre toute tentative de violation ou d'altération. Le cryptage des données s'avère une solution sûre pour palier à ce type de problématique.

Nous avons choisi RSA à 1024 bits pour l'échange des clés symétriques. La clé symétrique est générée aléatoirement par l'une des parties communicantes et distribuée par la suite à l'autre partie. Elle est utilisée pour le chiffrement des données durant toute la session. Cette technique, connue sous le nom « *Ephemeral keying* » permet de regénérer une nouvelle clé lors de chaque nouvelle session. De cette façon, même si un intrus réussit à intercepter une clé de session, cette dernière ne lui servira que temporairement.

Dans ce qui suit, nous présentons en détail les objets du système qui encapsulent et implémentent les mécanismes de sécurité définis ci-dessus.

6.3 Module de sécurité

L'application client et l'application serveur comportent chacune un objet qui assure des communications sécurisées avec l'autre partie. Il s'agit de l'objet *ServeurSSL* et de l'objet *ClientSSL* qui offrent les fonctionnalités SSL pour le serveur et pour le client. Pour mener à bien ce processus, ces deux objets créent respectivement un contexte SSL qui identifie l'algorithme de cryptage utilisé, les mots de passe des fichiers contenant les clés publiques et privées et s'assurent que les certificats respectent la syntaxe X.509.

Le standard X.509 de l'UIT (l'Union Internationale des Télécommunications) spécifie le contenu d'un certificat numérique. Ce dernier représente un document émis par un organisme digne de confiance, associant ainsi la clé publique d'un algorithme de chiffrement asymétrique à des informations relatives à un sujet (nom d'une personne ou d'un serveur, adresse courriel, etc.). Un certificat numérique est établi par une autorité de certification (*Certificate Authority ou CA*) qui emploie une signature numérique. Cette signature est le résultat d'une fonction de hachage appliquée au contenu à certifier et chiffré à l'aide de la clé privée du CA. Sa vérification ne nécessitera donc que la connaissance de la clé publique du CA.

Pour pouvoir créer une communication sécurisée avec le client, nous avons utilisé les sockets « SSL » définies dans la librairie JSSE aux paquetages javax.net, javax.net.ssl et javax.security.cert. Ces sockets peuvent être du type :

- SSLSocket
- SSLServerSocket
- SSLSocketFactory
- SSLServerSocketFactory.

SSLSocket supporte les méthodes de la classe java.net.Socket.class utilisées pour la manipulation des *sockets* standards en ajoutant des méthodes spécifiques aux *sockets* sécurisées. La classe SSLServerSocket permet de créer des instances de *serversockets* par le biais de la méthode createSocket(). La Figures 6.2 présente un exemple de code pour la création d'un serveur SSL.

La classe *SocketFactory* et la *ServerSocketFactory* sont des classes abstraites implantées respectivement par les classes SSLSocketFactory et SSLServerSocketFactory. L'utilisation des classes de type "Factory" permet d'instancier des objets dont on ne connaît pas à l'avance la sous-classe utilisée pour l'instanciation d'une socket SSL. La classe "Factory" va permettre de le décider à l'exécution. Ainsi la classe SSLServerSocketFactory va instancier une sous-classe de SSLServerSocket, correspondant aux besoins de l'application. Le choix est fait en fonction du système à développer et il s'agit donc d'encapsuler tout ce qui n'appartient pas à l'environnement Java.

Les paramètres du contexte SSL sont initialisés au moyen de keytool : un utilitaire disponible avec JDK 1.4. Par exemple l'authentification est réalisée à l'aide d'une paire de clés (une clé publique/une clé privée) et d'un certificat. On peut créer ces clés et un certificat auto-signé en utilisant la syntaxe suivante :

keytool -genkey -v -keyalg RSA -keystore keystore

et en saisissant un certain nombre d'informations, dont le mot de passe du "keystore".

```
/* Exemple de fonctions utilisées pour la création d'un serveur sécurisé */
SSLServerSocketFactory sf = (SSLServerSocketFactory)

/* getDefault permet d'obtenir les paramètres SSL définis par défaut */
SSLServerSocketFactory.getDefault();

/* creation de la socket */
SSLServerSocket s =   (SSLServerSocket)sf.createServerSocket(port);

/* le serveur commence l' écoute */
SSLSocket in = (SSLSocket)s.accept();

/* Oblige le client de  s'authentifié. Dans le cas où le client ne détient pas un
CA une erreur sera générée et par conséquent la communication, sera refusée
*/
s.setNeedClientAuth(true);

        /** récupération des flux d'entrés et de sorties comme  dans
        les sockets classiques. L'envoie et la réception des  messages
        à sécuriser **/
```

Figure 6.2 : Exemple de code pour un serveur SSL

6.4 Conclusion

Dans ce chapitre, nous avons présenté les mécanismes de sécurité implantés dans notre système. Nous avons décrit le standard SSL (*Secure Socket Layer*), son mode

de fonctionnement et ses principes. Nous avons aussi présenté les fonctions SSL définies dans la librairie JSSE de Java. Ces mécanismes de sécurité, robustes et efficaces, nous permettent d'assurer l'authentification, la confidentialité et l'intégrité des données de mesures transmises sur le réseau.

Chapitre 7

Protocoles de communication

Dans ce chapitre, nous présentons les protocoles de communication utilisés dans notre système distribué d'acquisition de données. Tout d'abord, nous allons faire la description du format des messages échangés. Ensuite, nous présentons l'ensemble des règles de communication entre les deux parties du système. Enfin, nous allons décrire le rôle du module responsable de la formulation et de l'interprétation des protocoles de communication dans notre système.

7.1 Introduction

Un protocole de communication est une série d'étapes, impliquant deux ou plusieurs entités, conçu pour accomplir une tâche. Il contient un ensemble de règles gouvernant la communication entres les participants.

Le protocole de communication que nous avons établi est défini par :

1. Le format des messages échangés entre le client et le serveur.
2. L'ensemble des règles de communication entre ces deux entités.

7.2 Format des messages

L'application client permet à un utilisateur de se connecter à plusieurs serveurs d'acquisition de données. Pour chaque connexion, les commandes 'client' sont conformes à un format standard défini par notre protocole de communication.

La commande 'client' est essentiellement composée d'un champ spécifiant le type de la requête, d'un nombre d'arguments et d'un numéro d'identification. Ce dernier distingue chaque requête d'une façon unique, ce qui permet au serveur de lui associer une réponse.

- Le protocole utilisé pour spécifier le format des requêtes 'client' respecte la forme suivante :

type de commande	argument	valeur de l'argument	numéro de commande

Où :

Type de commande : Identifie la nature de la commande envoyée. En effet, un client peut par exemple envoyer

une commande d'acquisition ou une commande d'annulation.

Argument : Identifie de façon unique la valeur d'un argument.

Valeur d'argument : Indique une information représentant le nom du senseur, la fréquence de lecture, etc. Le nombre d'arguments varie selon la commande envoyée.

Numéro de commande : Distingue chaque commande d'une façon unique.

Pour mécaniser ce protocole, nous avons utilisé JavaCC, une nouvelle technologie développée par Sun Microsystems. Il s'agit d'un outil qui a comme fonctionnalité la génération des parseurs pour Java. JavaCC est caractérisé par une syntaxe proche de celle de Java. Il permet d'avoir une sémantique et une grammaire qui permettent de simplifier l'interprétation des commandes. De plus, une fois compilé, les fichiers JavaCC s'intègrent aisément aux programmes Java.

L'utilisation de JavaCC est un atout pour notre application. En effet, de par ces caractéristiques, la portabilité de notre application ne sera pas compromise.

Dans ce qui suit, nous présentons les étapes de notre protocole de communication avec les messages que s'envoient le client et le serveur. Ces messages construisent en partie les messages échangés pour établir une communication sécurisée avec SSL. L'autre partie constitue les messages des données sécurisées échangées entre le client et le serveur.

7.3 Règles du protocole

Le protocole est gouverné par les règles suivantes :

1. Le client se connecte au serveur. Il lui envoie le message CLIENT-HELLO, qui contient entre autre sa version du protocole SSL et ses paramètres de chiffrement.

2. Le serveur répond au client en envoyant le message SERVER-HELLO. Ce message contient également sa version SSL et ses paramètres de chiffrement. Il envoie également son propre certificat et en demande un au client (message *REQUEST-CERTIFICATE*).

3. Le client authentifie le serveur grâce aux informations qu'il vient de recevoir. Si cette opération échoue, la connexion n'aura pas lieu.

4. Le client envoie le message *CLIENT-MASTER-KEY*. Celui-ci contient une pré-clef secrète, chiffrée avec la clé publique du serveur. Il envoie aussi un certificat dans un message *CLIENT-CERTIFICATE*.

5. Le serveur et le client mettent au point chacun de leur côté une clef secrète à partir de la pré-clef secrète.

6. Le client et le serveur utilisent ensuite cette clef secrète pour générer des clefs de session. Ces clefs symétriques seront dorénavant utilisées pour chiffrer, déchiffrer et authentifier les données.

7. Le client envoie le message *CLIENT-FINISHED* au serveur. De cette manière il le prévient que les prochains messages seront chiffrés avec les nouvelles clefs. Il indique également qu'il est satisfait du serveur et que la phase de négociation est terminée.

8. Le serveur, lui aussi satisfait du handshake du client, répond par un *message SERVER-FINISHED*. Il est prêt à passer au niveau supérieur de chiffrement proposé par le client. Il indique lui aussi que la phase de négociation est terminée.

9. Le client envoie sa première requête, conforme au format des messages, demandant au serveur la liste des senseurs disponibles. Avant d'être envoyée, le parseur généré par JavaCC effectue la vérification grammaticale de la requête.

10. Le serveur, lui aussi, au moyen de son propre parseur, effectue la vérification grammaticale de la requête. Si la requête ne satisfait pas le format des messages auquel il s'attend, une erreur est générée et par la suite envoyée au client. Dans le cas contraire, le serveur procède à l'extraction des paramètres de la requête afin de lui associer la réponse adéquate. Le résultat est avant tout crypté, puis envoyé au client approprié.

11. Le client, tout d'abord, décrypte les données reçues, ensuite, selon la commande envoyée, il les interprète en les associant à l'objet graphique approprié.

12. Le client procède à des acquisitions en envoyant des commandes spécifiques à ses besoins.

13. Le serveur en recevant une commande d'acquisition, lui attribue une priorité d'exécution et une horloge pour s'assurer du temps de son exécution. Le moment arrivé, il réalise : la collecte, le filtrage et l'envoi des données.

14. Si le client veut mettre fin à la connexion, il envoie une commande de déconnexion au serveur.

15. Le serveur, en recevant cette commande, supprime le client de la liste des connexions ainsi que toutes les commandes provenant de ce dernier.

L'authentification du serveur (client) est réalisée selon les étapes suivantes :

1. Vérification de la date de validité du certificat du serveur (client).

2. Vérification de l'autorité de certification.

3. Vérification de la clé publique à partir de la signature. Le serveur (client) vérifie la validité de la signature (chiffrée avec la clé privée) fournie dans le certificat grâce à la clé publique qui a été fournie elle aussi dans le certificat. A partir de ce point, le certificat du serveur (client) est considéré comme valable.

La figure suivante illustre une communication telle qu'établie par notre protocole de communication. Elle résume la communication analysée dans la partie précédente.

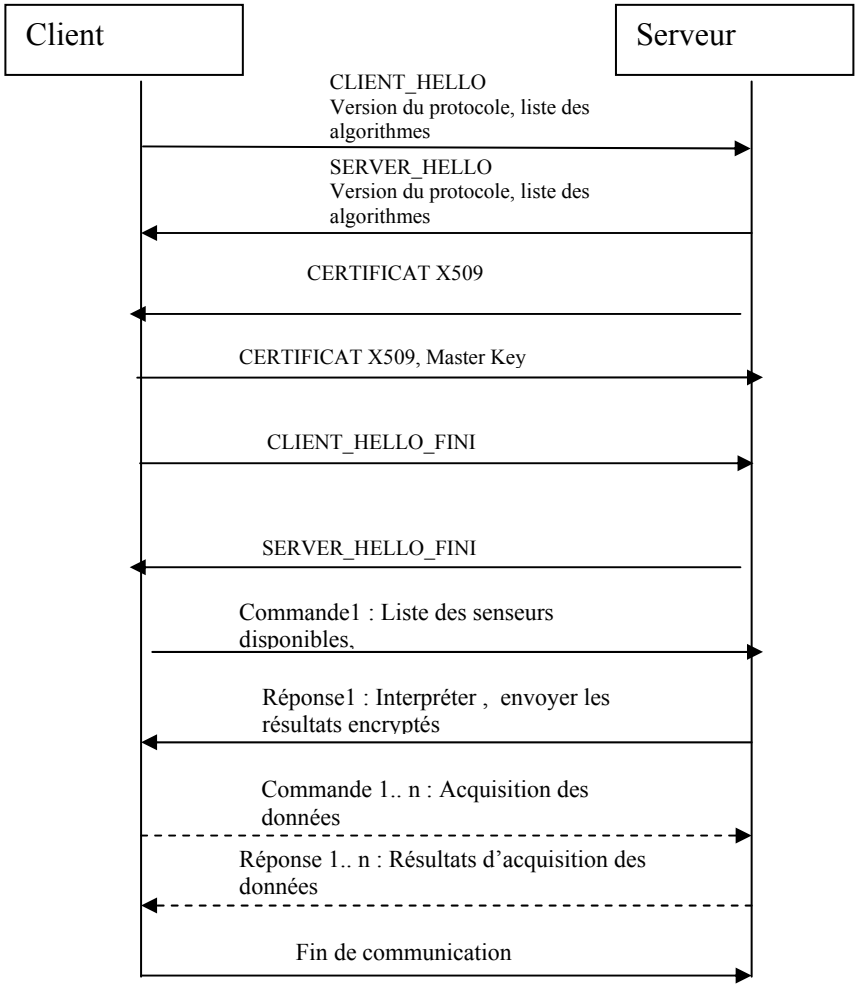

Figure 7.1 : Règles de communication entre le client et le serveur

7.4 Implantation des mécanismes d'interprétation et de vérification des messages

Dans notre système, les commandes 'client' sont vérifiées et interprétées lors de l'envoi et de la réception. En effet, *ICS* et *ICC* sont deux modules (Figures 3.3 et 3.5), chargés de l'interprétation et de la formulation des commandes. Chaque module d'interprétation est composé de deux sous modules : le premier permet d'établir un format standard pour l'ensemble des commandes reçues et vérifie qu'elles satisfont une certaine syntaxe. Le deuxième est responsable du décodage des commandes envoyées afin d'extraire les paramètres pertinents à son exécution. par exemple, pour une demande d'acquisition de données, le serveur doit connaître le nom du senseur à utiliser, le nombre de lectures à effectuer, etc. Le rôle de ce module est capital puisqu'il permet d'assurer les réponses appropriées au client. La Figure 7.2 montre une partie du code développé avec JavaCC pour les formats des messages échangés.

```
/* identifier chaque token avec son expression
régulière */
TOKEN :
{  <LETTER:  (["a"-"z","A"-"Z","_"])+ >
|  <DIGIT:   (["0"-"9"])+ >

/*  formulation des commandes */
|  <Command1: "get"" " "-all">

/* Commande 2 inclue le token LETTER  représenté
par une série de lettres alphabétiques */
|  <Command2: "read"" " "-p" " " <LETTER>>
}
```

Figure 7.2 : Formats des messages

Le client et le serveur s'attendent tout les deux à la réception d'une des commandes textes définies par le protocole. Dans le cas contraire, un code d'erreur sera levé et envoyé au client (voir Figure 7.3). Les commandes 'client' sont présentées sous format texte afin d'êtres conformes au standard de l'OMG.

```
ParsingError: Encountered "get.
Was expecting one of:

Command1: get -all
Command2: read -p [SensorName]
Command3: set -n [SensorName] -t [period (time)] –r
[recurrence (int)] –s [ScansCount (int)] –f [frequency (KHz)] –v [Voltage (V)]
Command4: fin
Command5: abort -n[SensorName]
```

Figure 7.3 : Exemple d'erreurs à la réception d'une commande erronée

7.5 Conclusion

Dans ce chapitre, nous avons présenté le protocole de communication de notre système spécialement conçu pour gouverner les échanges entre le client et le serveur. Dans ce protocole, nous avons décrit les messages sous format texte pour qu'ils soient conformes au standard de l'OMG.

Ce protocole, nous permet de gérer les échanges entre le client et le serveur et d'assurer la conformité des commandes émanant d'un usagé. Aussi, l'utilisation de JavaCC est une autre garantie de portabilité de notre application.

Chapitre 8

Interface utilisateur

Dans ce chapitre, nous présentons l'approche utilisée pour la conception de l'interface utilisateur graphique. Nous expliquons le but principal d'une interface configurable et nous présentons en détail les fonctionnalités du module principal contenant notre interface.

8.1 Introduction

Contrairement à la règle générale de construction des interfaces utilisateurs figées, l'interface que nous avons développée a la propriété d'être configurable. En effet, dans la revue de la littérature, tous les systèmes d'acquisition de données développés, de par leurs interfaces graphiques figées, leur champs d'application

sera désigné pour un seul type d'application. Nous pensons qu'une telle approche nous poussera vers une logique qui force le développement d'un système d'acquisition de données pour chaque champ d'application. Cette façon de faire, constitue sans doute la cause principale de la profusion des systèmes d'acquisition de données.

8.2 Construction de l'interface configurable

L'interface utilisateur que nous avons développée permet une grande flexibilité aux utilisateurs. En effet, la généricité de l'interface graphique permet à chaque utilisateur d'effectuer des mesures personnalisées par la reconfiguration de l'interface utilisateur selon les besoins de l'expérience. Par exemple, l'utilisateur peut reconfigurer l'interface client pour pouvoir surveiller un senseur de température sur un site et un senseur de pression sur un autre site (Figures 8.1 et 8.2). Aussi, l'utilisateur a la possibilité d'ajouter le nombre désiré de composants graphiques afin de mener adéquatement ces séances d'acquisition de données. Chaque composant graphique est un objet actif conçu de façon à offrir au client deux manières d'utilisation :

- Il peut interpréter des données transmises par le réseau. Dans ce cas, le composant graphique par exemple le *GraphPanel* associe chaque résultat de lecture provenant d'un serveur à un senseur donné.

- Il peut interpréter des données qui existent localement sur sa machine. Dans ce cas l'utilisateur n'aura pas besoin d'une connexion à distance avec le serveur pour afficher les données.

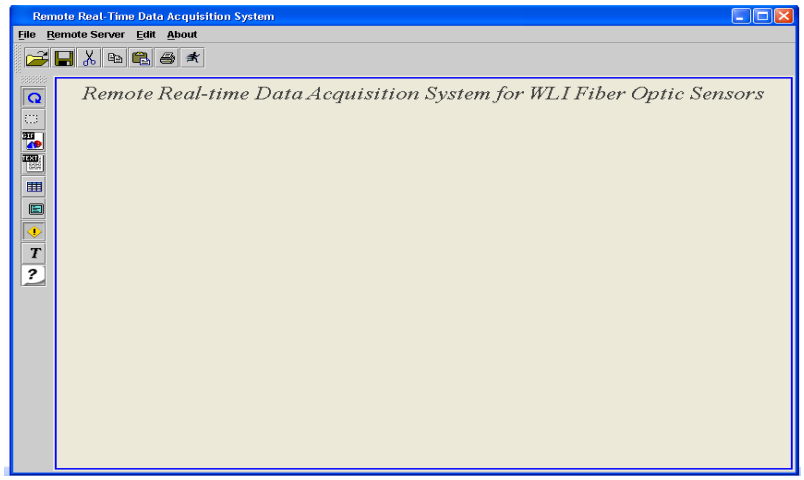

Figure 8.1 : Interface utilisateur de l'application client

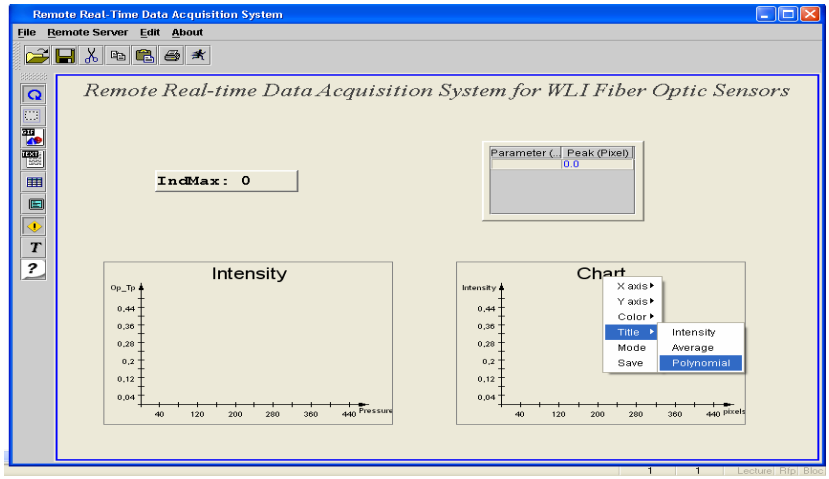

Figure 8.2 : Configuration de l'interface utilisateur

Dans ce qui suit, nous présentons l'autre partie de notre interface graphique. Celle-ci, a été conçue afin de permettre aux utilisateurs de contrôler la partie matérielle du système et le périphérique d'acquisition de données.

8.3 Interface virtuelle pour le contrôle du périphérique d'acquisition de données

Pour les utilisateurs, cette interface représente un panneau virtuel qui permet un contrôle total sur le périphérique d'acquisition de données. Une fois la communication avec le serveur établie, l'utilisateur dispose d'un panneau de contrôle virtuel (Figure 8.3) qui lui permet de contrôler les instruments en toute sécurité selon les exigences de l'expérience à mener. En exécutant le protocole de communication, l'utilisateur peut par exemple : spécifier la fréquence à laquelle la collecte des données sera réalisée, l'intensité du signal émis, le nom du senseur optique utilisé et le nombre de lectures à effectuer. Une fois les résultats reçus, ils seront interprétés et affichés (Figure 8.4)

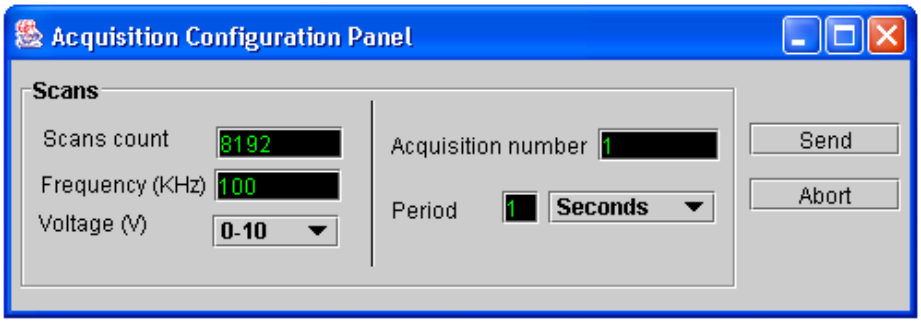

Figure 8.3 : Panneau de contrôle virtuel pour l'acquisition de données.

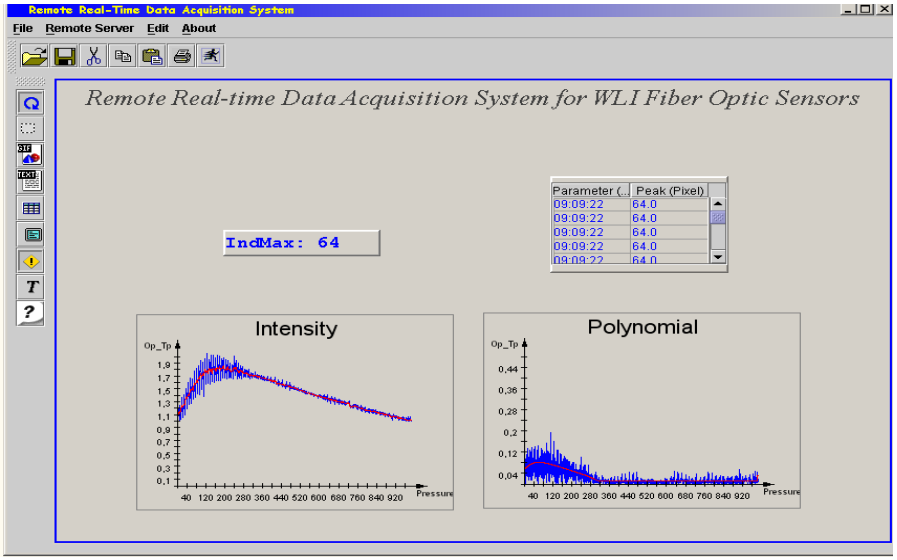

Figure 8.4 : Signal de sortie tel que demandé.

8.4 Conclusion

Dans ce chapitre, nous avons présenté notre interface utilisateur configurable. Cette interface offre une grande flexibilité aux utilisateurs en leurs permettant d'effectuer des mesures personnalisées selon les besoins de l'expérience. Cette propriété permet à notre système un grand champ d'applications industrielles. De plus, la possibilité du contrôle à distance des périphériques d'acquisition de données, offerte par le panneau de contrôle virtuel, constitue en soit un très grand avantage pour notre système distribué d'acquisition de données.

Chapitre 9

Temps réel

Dans ce chapitre, nous présentons l'aspect temps réel de notre système. Ainsi, nous explorons l'analyse de performance du système dans son environnement réel. Cette analyse est réalisée grâce à l'application des théories des algorithmes d'ordonnancement pour les systèmes temps réel distribués. L'analyse de performance nous permet de vérifier si les dates limites des événements du système vont être respectées et par conséquent si une restructuration du système serait nécessaire.

9.1 Généralités

L'environnement dans lequel les systèmes d'acquisition de données temps réel opèrent est un facteur important à considérer lors de leur conception. L'exactitude d'un tel système ne dépend pas seulement de ses résultats logiques, mais aussi du

temps mis pour les produire et de sa capacité à maintenir une bonne interaction avec son environnement externe. En effet, lors de la conception de ce type de système, plusieurs facteurs devraient être pris en compte afin de pouvoir estimer le temps d'exécution des différents événements. Parmi ces facteurs nous pouvons citer les temps de:

- Exécution des tâches : Temps CPU (*Central Processing Unit*) exigé pour le traitement d'une tâche.
- Interruption : Temps d'interruption par une tâche à priorité plus élevée.
- Blocage : Temps de blocage par une tâche à priorité moins élevée.

9.2 Analyse de performance du système d'acquisition de données

L'analyse de performance pour la conception logicielle est particulièrement importante pour les systèmes temps réel. Une analyse quantitative permet une détection avancée des problèmes de performance et offre la possibilité d'explorer des solutions alternatives pour concevoir ces types de systèmes.

L'application des théories des algorithmes d'ordonnancement temps réel est une des approches largement utilisée pour la réalisation d'une telle analyse. Cependant, ces théories doivent être appliquées à une conception logicielle basée sur une architecture de tâches concurrentes.

Dans ce qui suit, nous présentons tout d'abord les éléments d'analyse de performance pour les systèmes temps réel tels que définis dans [8]. Ensuite, nous

les appliquerons à notre système. Enfin, une conclusion sur les résultats obtenus sera présentée.

9.2.1 Eléments d'analyse

9.2.1.1 Algorithmes d'ordonnancement

Les priorités jouent un rôle très important pour l'ordonnancement des tâches dans les systèmes temps réel. Le principe d'analyse de performance pour un ensemble de tâches dont les durées d'exécution sont connues consiste à déterminer si les contraintes de temps associées seront respectée [8]. Dans la littérature, quatre algorithmes sont utilisés pour réaliser ce type d'analyse. Le choix est effectué selon la situation réelle de chaque système et c'est aux concepteurs d'en décider :

1. ***Taux monotonique*** : Pour les tâches périodiques indépendantes, nous considérons la théorie du taux monotonique. Les tâches sont considérées par leurs périodicités T et leurs durées d'exécution C (temps CPU nécessaire durant la période). Une tâche est dite ordonnançeable si ses dates limites (deadlines) sont atteintes: la tâche complète son exécution avant la prochaine période. Dans l'algorithme de taux monotonique la priorité d'une tâche est inversement proportionnelle à sa période.

2. ***Théorème des limites*** : Un groupe de tâches périodiques et indépendantes répond toujours aux contraintes de temps de réponse si le rapport C/T pour chaque tâche est toujours inférieur à une valeur donnée

$$C_1/T_1 + C_2/T_2 + \ldots. C_n/T_n <= n\left(2^{1/n} - 1\right) = U_{(n)}$$

La limite U(n) converge vers (ln 2) lorsque le nombre des tâches s'approche de l'infini.

3. ***Théorème de temps de complétude*** : Lorsque les utilisations des tâches sont plus grandes que la limite du théorème monotonique, un deuxième algorithme sera utilisé. Le second théorème fournit d'autres critères d'ordonnançeabilité. Le théorème de temps de complétude utilise le pire cas dans lequel les tâches sont prêtes à s'exécuter en parallèle. Dans le pire des cas, si une tâche termine son traitement avant la prochaine période, alors ses contraintes temporelles seront toujours respectées.

4. ***Théorème des limites généralisé*** : Le théorème des limites a été étendue pour prendre en compte les temps de blocages et d'interruptions des tâches. Une tâche est dite ordonnançeable si la valeur de Ui pour chaque tâche est toujours inférieure à ln 2.

$$U_i = \left(\sum_{j \in H_n} C_j / T_j \right) + \frac{1}{T_i} \left(C_i + B_i + \sum_{k \in H_1} C_k \right)$$

Les facteurs de l'équation sont respectivement représentés par :

1. Temps d'interruption par une tâche à plus grande priorité et une période plus courte.
2. Temps d'exécution de l'événement estimé par le théorème des limites par $U_i = C_i / T_i$.

Où

Ci : représente le temps CPU de la tâche i

Ti : représente la période de la tâche i

3. Temps de blocage par une tâche à priorité moins élevée (nous considérons, seulement le pire cas).
4. Temps d'interruption total par les tâches à plus grande priorité avec une période plus longue que Ti.

9.2.2.2 Structuration de tâche de l'architecture

Durant la phase de structuration de tâche d'une architecture, un ensemble de critères de structuration doivent être appliqués sur l'architecture du système. Ces critères sont fournis afin d'assister les concepteurs pendant la transformation du modèle orienté objet en un modèle concurrent [8].

9.3 Analyse de performance pour le système proposé

9.3.1 Objectifs

Analyser la performance du système d'acquisition de données nous permettra, tout d'abord, de déterminer s'il existe un algorithme qui satisfait les contraintes temporelles du système proposé. Ensuite, de vérifier si les dates limites des événements du système vont être respectées. Enfin, de décider si le système sera considéré comme un système temps réel ou non. Dans le cas où les dates limites des événements n'ont pas été respectées, une alternative de conception doit être envisagée. Cette alternative sera considérée sur deux niveaux :

- le choix de l'algorithme d'ordonnancement et l'attribution des priorités d'exécution des événements.
- La restructuration du système.

9.3.2 Éléments de bases

Dans notre système, deux éléments sont à tenir en compte :

1. la fonction principale de l'application serveur consiste à répondre aux demandes clients pour l'acquisition de données. Toutefois, pour que le serveur puisse satisfaire ces demandes, il doit réaliser les tâches suivantes :

    - Réception, interprétation et ordonnancement de la commande
    - Acquisition et sauvegarde des données
    - Lecture et échantillonnage des données
    - Envoi des résultats au client.

2. Selon les principes de l'analyse de performance déjà évoqués, nous devons déterminer le temps CPU moyen alloué pour chaque tâche. Pour ce faire, une simulation nous permet d'estimer le temps moyen d'exécution de chaque tâche d'acquisition et d'envoi de données au client.

Pour y parvenir, nous allons faire les suppositions suivantes:

1. le serveur de mesure possède un seul processeur CPU pour desservir les requêtes 'client' .

2. le temps CPU est entièrement utilisé par l'application serveur. En effet, il n y'a pas d'autres applications qui s'exécutent en même temps.

La simulation, une fois réalisée, a permis de dresser le tableau suivant :

Tâches	**Temps CPU Ci (ms)**
Interprétation des commandes	5
Acquisition des données	2450
lecture et échantillonnage des données	60
Communication avec le client	10

Tableau 9.1 : Temps d'exécution des principales tâches du système ServeurAD.

9.3.3 Scénario d'utilisation

En se basant sur l'environnement réel dans lequel le système opère, nous allons considérer que : à un temps t, le serveur d'acquisition de données recevra quatre événements e1, e2, e3 et e4. Soient, e1, e2, e3, des événements périodiques (acquisition de données), alors que e4 sera un événement apériodique (demande d'informations sur un senseur).

L'utilisateur peut faire des acquisitions dans un intervalle de temps t compris entre une borne inférieure égale à une minute et une borne supérieure ou égale à une journée. Dans notre cas, nous avons choisi des périodes d'exécution d'une minute, de quinze minutes, et d'une journée pour les événements e1, e2, e3. L'événement

e4, par contre, aura une périodicité égale à celle estimée au pire cas d'interarrivée (*worst case interarrival*), c'est à dire cinq minutes.

9.3.4 Application des théorèmes d'ordonnancement

L'application des théorèmes d'ordonnancement consiste en deux étapes :

1. Attribution des priorités aux événements du système : Les priorités d'exécution des événements e1, e2, e3 et e4 sont évaluées selon le théorème du taux monotonique . Cet algorithme attribue les priorités selon la périodicité des événements. Plus la période est courte, plus la priorité est élevée (voir le tableau 9.2).

	Période (ms)	Temps CPU (ms)	Priorité
e1	60000	2525	1
e2	900000	2525	3
e3	86400000	2525	4
e4	300000	60	2

Tableau 9.2 : Ordonnancement des événements

2. Choix et application de l'algorithme d'ordonnancement : Pour choisir l'algorithme d'ordonnancement le plus adapté au système proposé, nous allons évaluer le temps de blocage et le temps d'interruption pour

chaque événement. En effet, l'accès aux senseurs optiques (ressources partagées) peut être bloquant dés que deux utilisateurs ou plus tentent de les accéder en même temps (voir Figure 9.1).

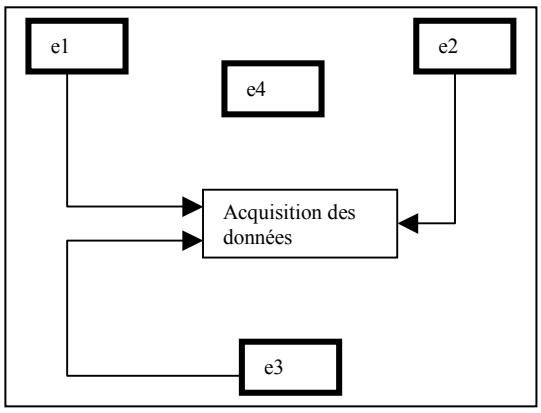

Figure 9.1 : Ressources partagées dans le système d'acquisition de données.

Les événements e1, e2 et e3 utilisent la ressource partagée « acquisition de données ». L'événement e1 peut être bloqué par e2 et e3, alors que l'événement e2 peut être bloqué par l'événement e3.

De plus, les interruptions peuvent également se produire dans notre système si un événement à priorité plus élevée veut détenir une ressource utilisée par un événement à priorité moins élevé en cours d'exécution : par exemple e2 et e3.

Pour évaluer cette situation, nous allons présenter un diagramme de séquence pour les événements de notre système (voir Figure 9.2). Nous considérons leurs exécutions jusqu'à l'occurrence de trois périodes :

soit une période de trois jours. L'ordre d'exécution est attribué selon l'ordre de priorité des événements.

Le diagramme de séquence suivant, montre clairement qu'aucun événement ne sera interrompu pendant son exécution par un autre. Par contre, des temps de blocage sont observés pour quelques événements. Par conséquent, nous optons pour le théorème des limites généralisé pour analyser la performance de notre système d'acquisition de données.

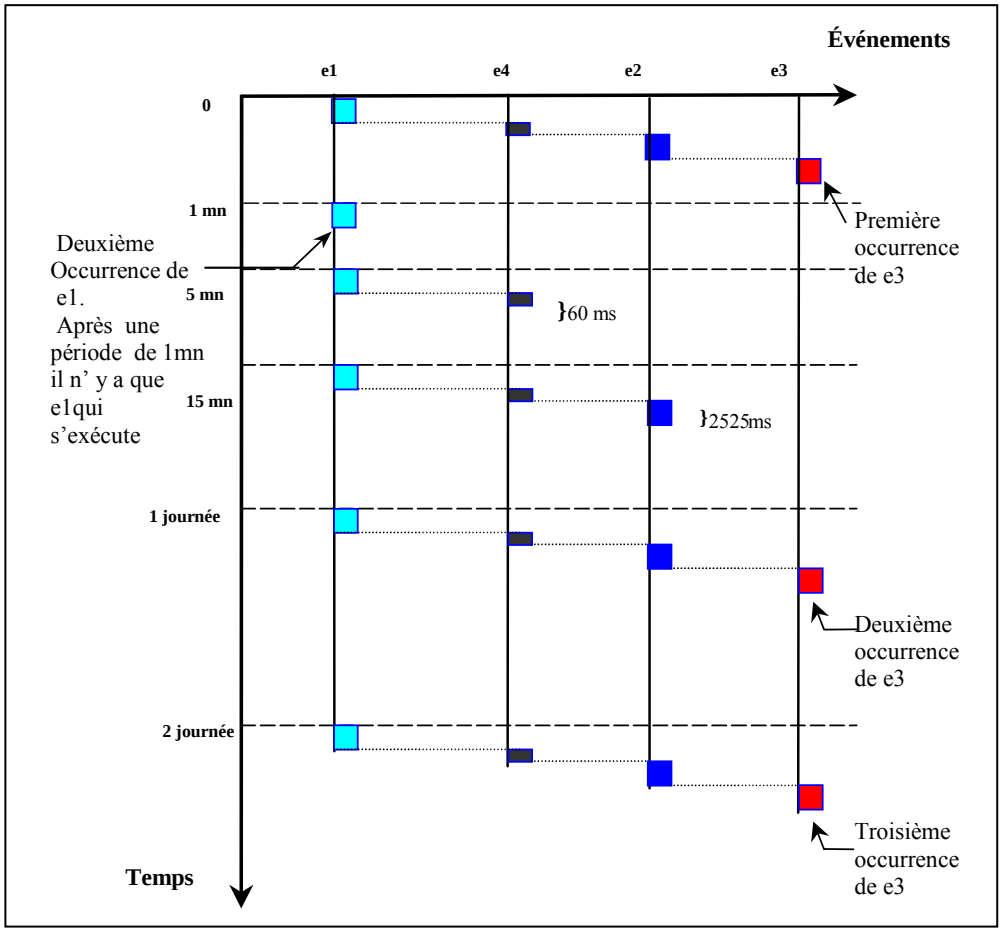

Figure 9.2 : Diagramme de séquences des événements du système dans le temps .

Dans ce diagramme, les durées temporelles ne sont pas respectées.
L'accent est plutôt mis sur les séquences des événements.

Dans notre cas, l'application du théorème des limites généralisé implique la vérification de deux facteurs pour chaque événement du système : le temps d'exécution Ui (Tableau 9.3) et le temps de blocage.

	Période (ms)	Temps CPU (ms)	Utilisation Ui	Priorité
e1	60000	2525	0,042	1
e2	900000	2525	0,0028	3
e3	86400000	2525	0,000029	4
e4	300000	60	0,0002	2

Tableau 9.3 : Bornes d'utilisations des événements

- Événement e1
  - ✓ **Temps d'exécution C1 pour l'événement e1 =** 2525. La limite d'utilisation $U_1 = C_1 / T_1 = 0{,}042$
  - ✓ **Temps de blocage par une tâche à priorité moins élevée**. e2 et e3 peuvent potentiellement bloquer e1. Le temps de blocage de e1 est égal au temps de blocage maximal provoqué par l'un de ces deux événements. Soit : 2450/60000 = 0,0408.

Le temps d'utilisation maximal de l'événement e1, U_{e1} = 0,0828 (0,042 +0,0408) < 0,69 (la borne limite est égal a 0,69 pour un nombre infini de tâches ou événement). Par conséquent l'événement e1 va rencontrer sa date limite sans aucun problème.

- Événement e2
  - ✓ **Temps d'exécution C2 pour l'événement e2 = 900000**. L'utilisation du CPU = U2 = 0,0028
  - ✓ **Temps de blocage par une tâche à priorité moins élevée**. L'événement e3 peut bloquée e2 pour une durée égale à 2450/900000 = 0,0027

Le temps d'utilisation maximal de l'événement e2, $U_{e2} = 0,0055 < 0,69$. Nous concluons, que l'événement e2 va également rencontrer sa date limite d'exécution sans aucune difficulté.

- Événement e3
  - ✓ **Temps d'exécution C3 pour l'événement e3 =** 2525. L'utilisation du CPU est estimée par U_{e3}, est sera égale à 0, 000029.
  - ✓ **Temps de blocage par une tâche de moindre priorité**. Ce cas ne s'applique pas à l'événement e3. Aucun événement ne détient une priorité moins élevée que e3.

Le temps d'utilisation maximal de l'événement e3, $U_{e3} = 0,000029 < 0,69$. Par conséquent e3 va aussi rencontrer sa date limite d'exécution.

- Événement e4
  - ✓ **Temps d'exécution C4 pour l'événement e4 =** 60. L'utilisation du CPU = U4 = 60/ 300000 = 0,0002
  - ✓ **Temps de blocage par une tâche à moindre priorité**. L'événement e4 n'utilise pas les ressources partagées.

Le temps d'utilisation maximal de l'événement e4, $U_{e4} = 0,0002 < 0,69$. L'événement e4 va également rencontrer sa date limite.

9.3 Conclusion

Dans ce chapitre, nous avons considéré l'aspect temps réel du système d'acquisition de données. Ainsi, nous avons réalisé une analyse de performance en prenant en compte sa situation dans l'environnement réel. Nous avons constaté, selon le scénario établie, que les événements du système vont sans exception et sans difficulté rencontrer leurs dates limites. Nous considérons, qu'initialement, la performance du système s'annonce plutôt bonne et par conséquent, le système pourra répondre aux requêtes 'client' sans aucune difficulté.

Chapitre 10

Conclusion

Suite à l'étude que nous venons de présenter, nous avons pu développer une architecture logicielle à plate-forme indépendante, temps réel et distribuée pour un système d'acquisition de données sécurisé, doté d'une interface utilisateur configurable et spécialement conçu pour les senseurs photoniques.

Tout d'abord, cette architecture, assure l'intégrité et la confidentialité des données transmises via le réseau, grâce à des mécanismes de sécurité robustes et efficaces. Elle est validée par la production d'un logiciel [6] qui permet aux utilisateurs d'accéder simultanément à des instruments localisés dans des sites différents pour l'acquisition, le traitement et l'affichage de données. Aussi, l'analyse de performance réalisée sur ce logiciel, démontre que nous avons conçu un système performant, qui peut satisfaire les commandes 'client' sans aucune difficulté.

De plus, le choix du langage Java pour l'implantation de l'architecture conçue, a permit d'obtenir un système dont l'ensemble de ses composants est portable y compris l'application client et l'application serveur. Les atouts du langage Java ne se limitent pas seulement à cet avantage, mais entraînent aussi d'autres en terme de documentation et de maintenance de code. En effet, au lieu d'avoir différents ensembles de code nécessitant différents ensembles d'outils, nous avons une seule base de code cohérente et manipulable par un seul ensemble d'outils.

Aussi, l'interface graphique permet à chaque utilisateur d'effectuer des mesures personnalisées par la reconfiguration de l'interface utilisateur selon les besoins de l'expérience. Cette façon de faire permet à notre système un vaste champ d'applications industrielles.

Cette étude nous a permis de combiner deux technologies : les senseurs photoniques et les systèmes d'acquisition distribués et temps réel. Nous pensons donc qu'une amélioration du logiciel produit sera possible sur deux niveaux :

- Informatique : Par exemple, l'utilisation des communications sans fils.
- Senseurs photoniques : L'élargissement de la gamme des paramètres mesurés, citons : la salinité, la pureté dans l'eau, la qualité spectrale, etc.

Bibliographie

[1] M. Bertocco, M. Parvis, “Platform Independent Architecture for Distributed Measurement System”, *IEEE Instrumentation and Measurement Technology Conferences,* USA, Mai 2000.

[2] W. J. Bock, W. Urbanczyk, M. R. Voet, A. Barwicz, "Multiplexing of Fiber-optic Sensors for Stress Measurements in Civil Engineering Applications", *Proc.SPIE*, USA, vol. 3414, pp. 86-92, 1998.

[3] W. J. Bock and W. Urbańczyk, "Coherence Multiplexing of fiber-optic pressure and temperature sensors based on highly birefringent fibers", *Proc. 16th IEEE Instrumentation and Measurement Technology Conference*, Venice, Italy, vol 3, pp. 1541-1545, 1999.

[4] G. Booch, *Object-oriented analysis and design with applications,* 2nd Edition, Benjamin/Cummings Pub. Co., 1994.

[5] B. Culshaw, *Optical fiber sensing and signal processing*, 2nd Edition, 1984.

[6] F. Djebbar, K. Adi, W.J. Bock, "Remote data acquisition system for white-light interferometric fiber-optic sensors", Soumis pour publication à *IEEE Sensors,* 2003, Toronto, Canada.

[7] G. Fortino, L. Nigro, “A measurement on-demand service for access and delivery process acquisition data”, *IEEE Instrumentation and Measurement Technology Conferences*, USA, Mai 2000.

[8] Hassan Gomaa, *Designing Concurrent, Distributed, and Real-Time Applications with UML,* Addison Wesley, 1st Edition , 2000.

[9] IVI Foundation, “Interchangeable Virtual Instruments Standard”, 1997. http://www.ivifoundation.org/.

[10] I. Jacobson, *Object-oriented software engineering : a use case driven approach*, Addison-Wesley, 1992.

[11] C. Larman, *Applying UML and patterns, an introduction to object-oriented analysis and design and the unified process*, Prentice Hall, 2nd Edition, 2002.

[12] T. Nieva, A. Wegmann. "A Conceptual Model for Remote Data Acquisition Systems". *Proc. 19th International Conference on Conceptual Modeling - ER'2000*, USA, October 2000.

[13] ODAA, "Open Data Acquisition Standard", 1998. http://www.opendaq.org/.

[14] OPC Foundation, "OLE for Process and Control Standard", 1997. http://www.opcfoundation.org.

[15] J. Pereira, A. Castro, D. Ronda, B.Arcay, A.Pazos, "Development of a System for Access to and Exploitation of Medical Images", *IEEE Symposium on Computer-Based Medical Systems*, USA, vol 323, pp. 14-309, Juin 2000.

[16] F. Toran, D. Ramirez, S. Casans, A.E.Navarro, J.Pelegri, "Distributed virtual instrument for water quality monitoring across the Internet", *IEEE Instrumentation and Measurement Technology Conferences*, USA, Mai 2000.

[17] John Viega, Matt Messier, Pravir Chandra, *Network Security with OpenSSL Cryptography for Secure Communications*, O'Reilly, 1st Edition, 2002.