Conservatoire National des Arts et Métiers - Paris

Spécification et Prototypage d'une Messagerie Industrielle à Contraintes Temporelles orientée Objets et Composants

THÈSE DE DOCTORAT

présentée et soutenue publiquement le 4 mai 2006

pour l'obtention du

Doctorat du Conservatoire National des Arts et Métiers

(Spécialité Informatique)

par

Erwan Becquet

Composition du jury

Président	Gérard Florin, Professeur au C.N.A.M.
Rapporteurs	Guy Juanole, Professeur au L.A.A.S. Pierre Sens, Professeur au L.I.P.6
Directeur	Eric Gressier-Soudan, Professeur au C.N.A.M.
Examinateurs	Claude Girault, Professeur au L.I.P.6 Jean-Dominique Decotignie, Professeur à l'E.P.F.L. François Horn, France Telecom R&D Laurent Bacon, EDF R&D

C.E.D.R.I.C. - Centre D' Etudes et De Recherche en Informatique du C.N.A.M.

Remerciements

Mes premiers remerciements vont à mon directeur de thèse, M. Eric Gressier-Soudan. De par son entrain communicatif, sa bonne humeur perpétuelle, son énergie quasi-surnaturelle, il a su me pousser pour que j'aille jusqu'au bout. Il a également été à l'écoute de mes souhaits quant aux choix qu'il a fallu faire tout au long de ces années, en sachant me recadrer lorsqu'il en était besoin. Merci encore, de par mes discussions avec d'autres doctorants qui n'ont pas eu ma chance, je peux apprécier à sa juste valeur son travail de directeur de thèse. Je ne peux souhaiter qu'une chose, qu'il continue encore longtemps à prodiguer ses cours, encadrements de mémoires et de thèses.

Ensuite ma famille, qui elle aussi a su prodiguer les encouragements nécessaires aux bons moments, au premier rang desquels mes parents et ma femme. Ils ont su me supporter, dans tous les sens du terme, afin que je mène à bien ce travail.

Je remercie également tous les doctorants, élèves ingénieur, secrétaires, maîtres de conférences et professeurs que j'ai pu croiser au CNAM pour leur sérieux, leur dynamisme et surtout leur joie de vivre et leur positivisme. Merci pour ces heures de discussion passionnée autour d'un café pour les uns ou d'un thé pour les autres sur des sujets très variés. Merci pour ces conseils, ces discussions techniques sur beaucoup de sujets liés à l'informatique mais aux antipodes du sujet de ma thèse, merci pour cette ouverture d'esprit, merci pour tout.

Je dédie cette thèse à mes parents, à ma femme et à mon fils.

"Si tu poses une question, tu sembleras ignorant
un instant. Si tu n'en poses pas, tu resteras
ignorant toute ta vie."
Un (sans doute) vieux (certainement) sage (probablement) chinois . . .

Résumé

Ce travail a pour origine une constatation et l'état d'un vrai besoin en ce qui concerne les applications de contrôle-commande dans le domaine de la production d'énergie. Un état de l'art a été réalisé via des études de faisabilité chez EDF R&D Chatou sur le remplacement des systèmes de contrôle-commande en centrales. Cette étude a mis en évidence le besoin d'une solution à base de composants sur étagère et qui pourrait Ã[a]tre déployée à tous les niveaux de l'architecture (intra-centre de production et inter-centres) afin d'améliorer l'interoperabilité, l'évolutivité et la pérennité.

En prenant comme point de départ fonctionnel une messagerie industrielle éprouvée dans le monde réel (IEC/TASE.2) le développement d'un prototype, complètement opérationnel, basé sur CORBA nous permet de valider l'intérêt de la solution. L'architecture de ce prototype est ensuite redéfinie à base de composants.

Nous concluons sur les évolutions possibles de ces prototypes, d'un point de vue fonctionnel (fonctionnalités optionnelles de TASE.2 à intégrer ou d'autres fonctionnalités), d'un point de vue architecture (programmation par aspects par exemple) et d'un point de vue cible puisque nous pensons que ce genre d'architecture peut adresser bien d'autres problématiques que la production/distribution d'énergie.

Mots-clés: ORB, Temps Réel, TASE.2, ICCP, messagerie industrielle, EDF, CORBA, Composants, CCM.

Abstract

This work is based on a fact and the discovering of a real need in the domain of the control command applications for energy production. A state of the art has been made and knowledge has been taken, when doing a study in the EDF Research & Development center of Chatou, about the possible replacement of actual control commands proprietary solutions in electric production centers. This study shows the need for a solution based on COTS (component off the shelf) and which could be installed at every architecture level (in the production centers and between them). Thus, we could improve interoperability, evolutivity, openness and perennity.

Taking as a start point a well-known and used messaging system like TASE.2 (aka ICCP), we ensure we implement all needed functions. We study TASE.2 services porting a TASE.2 compliant server from Windows NT to a real-time operating system. Then we develop an operational prototype on top of CORBA, thus validating the feasibility of this "open" messaging system. This existing prototype is then rewritten using components architecture, and implemented with a CORBA-RT compliant ORB. This second prototype is also fully functional, and we can achieve intersting comparisons between both of them : performances of course but also development costs, reusability, scaling, openness ...

We conclude on the possible evolutions for these two prototypes : from a functional point of view (add some optional functionalities of TASE.2 protocol or others), from an architecture point of view (using aspect programming), and from targeted domain point of view, because we think these king of timed messaging services can meet a lot of other applications requirements as network games for example.

Keywords: ORB, Real-Time, TASE.2, ICCP, messaging system, EDF, CORBA, Components, CCM.

Table des matières

Chapitre 2
Médiation & Supervision dans le domaine de la production d'énergie

Chapitre 3
La messagerie industrielle I.E.C./T.A.S.E.2

Table des figures

Liste des tableaux

Avant-Propos

Ce travail de thèse a mis du temps à se concrétiser, et nous en rappelons ici les dates-clé, pour mieux mettre en rapport le niveau de la recherche de l'époque. Ceci explique le développement objet dans un premier temps puis le passage au modèle par composants.

Les publications (rapports d'activité, conférences) sont cités dans la chronologie suivante associés aux travaux concernés et le détail est donné en fin de document après la bibliographie générale

- **Année 1998-1999 (Inscription en Thèse)**
  - Étude de Systèmes d'Exploitation Temps Réel [1]
  - Développement d'un banc de tests
  - Comparaison de pSOS+/vxWorks/LynxOS
  - Publications : [84] [85] [79] [82] [78] [81] [86] [80] [74]
- **Année 1999-2000**
  - Étude ORBS Temps Réel
  - Étude de conformité & Portage d'un serveur IEC/TASE.2 sur pSOS+
  - Publications : [83] [90] [91] [75] [89] [76]
- **Année 2000-2001**
  - Prototype IEC/TASE.2 en CORBA (TASE.2 en Objet)
  - Publications : [63] [70] [69] [65] [67]
- **Année 2001-2002**
  - Finalisation articles
  - Recherche de travail à temps plein / Formations
  - Publications : [66] [72] [64]
- **Année 2002-2003**
  - Activité professionnelle
  - Développement Prototype Composants
  - Publications : [73]
- **Année 2003-2004**
  - Activité professionnelle
  - Poursuite Rédaction Thèse

[1] Ce travail n'est pas exploité dans le corps de la thèse et se trouve en annexe

- ➞ Développement Prototype Composants
- ➞ Publications : [68] [71]

➡ **Année 2004-2005**

- ➞ Rédaction rapport final

Compte tenu des opportunités apparues dans le cadre des études d'architectures de systèmes distribués temps réel commanditées par EDF R&D et de la disparition de l'ORB COOL au dessus du noyau ChorusOS de Chorus Systèmes, devenu Sun entre temps, le sujet a évolué. Le centre de l'étude a migré des architectures pour le contrôle-commande vers les architectures de médiation pour le contrôle de la production d'énergie.

La thèse s'est déroulée effectivement de octobre 1998 à septembre 2001 avec un complément de travail en 2003 en parallèle de mon activité professionnelle. Ce travail est jalonné par les publications listées en fin de document après la bibliographie générale et citées dans la chronologie ci-avant.

Chapitre 1

Introduction

Sommaire

1.1 Problématique

Les centrales de production d'énergie électrique d'E.D.F. ont des besoins très spécifiques en ce qui concerne l'informatique. Il s'agit essentiellement d'applications distribuées par nature et ayant des contraintes temporelles fortes. L'essentiel des fonctionnalités attendues sont de type envoi et réception de données suivant différents schémas. Pour l'instant, l'essentiel de ces besoins est couvert par des solutions propriétaires coûteuses et hétérogènes. Pour la prochaine génération de centrales, E.D.F. veut évaluer des solutions souples, basées autant que faire se peut, sur des composants sur étagère et des standards. En parallèle, il semble intéressant d'adresser également les problèmes soulevés par les interactions de la centrale avec le monde extérieur (médiation) qui sont étroitement liés avec ceux déjà évoqués en termes de fonctionnalités et de données échangées.

Nous parlons là de la supervision de procédés. Faire de la supervision, c'est essentiellement :

① **Fonctionnellement** : Définir un outil visuel qui synthétise un procédé physique et l'ensemble des équipements associés (dans notre cas le procédé

de production d'énergie), définir une partie calcul qui fabrique un état du procédé physique en fonction de données acquises et de commandes et qui fournit des ordres pour faire évoluer ou non le procédé ou ses alarmes, et finalement et optionnellement offrir une base de données de journalisation

② **Techniquement** : Utiliser un protocole de médiation ou bien encore une messagerie industrielle pour acheminer les informations d'état et les commandes entre des équipements et un point de supervision

Nous voyons que cette définition, dans le domaine de la production d'énergie, peut s'appliquer autant sur les interactions internes à une centrale qu'aux interactions entre une centrale et un centre de régulation de la production. Nous nous intéressons aux messageries industrielles, dans les travaux normalisés, la première version (interne) correspond à UCA.2 (cf. la norme [9]) tandis que la deuxième correspond à IEC/TASE.2 (cf. la norme [55]).

Sémantiquement, ces deux messageries ne sont pas fondamentalement différentes. Ce qui les sépare, c'est :

- ➡ la prise en compte de la sécurité dans IEC/TASE.2, ce qui est logique puisqu'elle se déploie hors de la centrale, sur un réseau ouvert et non protégé
- ➡ la possibilité de faire des communications 1 vers n dans UCA.2, cependant n n'étant pas très grand dans la pratique (2 ou 3 pour des raisons de réplication pour garantir une tolérance aux pannes), il est acceptable de simuler ces communications par n relations 1 vers 1.

En conséquence, notre travail qui a porté sur TASE.2, est suffisamment générique pour être appliqué à UCA.2 voire à toute messagerie industrielle.

Le travail qui a été fait est de proposer une architecture à la fois souple, pérenne, évolutive et interoperable qui adresse les problèmes de supervision dans le domaine de la production d'énergie. Les fonctionnalités apportées par cette architecture sont des rôles pour les différents acteurs, de producteur et/ou de consommateurs d'informations.

Cette architecture, que nous proposons et qui souhaite répondre à la problématique scientifique abordée dans ce document, est née de l'objectification dans un premier temps, d'une messagerie industrielle. Partant de la spécification fonctionnelle d'une messagerie IEC/TASE.2, nous proposons une conception à base d'objets et orientée appel de méthodes (aka "objectification" de la version classique par passage de messages). En utilisant des technologies ouvertes et standardisées, nous adressons les contraintes de souplesse, d'interoperabilité et de pérennité. Dans un second temps, nous tirons parti du modèle par composants pour faire évoluer notre architecture en termes d'évolutivité et de réutilisabilité.

Le paragraphe suivant fait brièvement le point sur le concept de messagerie industrielle.

1.2 Messagerie industrielle

La définition reprise ici ainsi que les exemples sont largement inspirés de [38].

1.2.1 Définition

Une Messagerie Industrielle est un protocole d'application permettant à des équipements industriels variés de s'échanger des informations entre eux. Un protocole classique offre des services directement liés au problème qu'il adresse. On peut citer FTP [1] qui est spécialisé dans le transfert de fichiers, IMAP [2] dont le rôle est d'accéder des boites aux lettres ou bien encore SNMP [3] pour la gestion d'équipements réseaux. Les services offerts sont donc très ciblés et en nombre restreint, de l'ordre d'une dizaine.

Une messagerie industrielle est au contraire générique et offre un large éventail d'objets et de services pour permettre les communications dans un domaine industriel. Le champ est très vaste et le nombre de services s'en ressent. Ainsi la messagerie industrielle MMS [4], le standard ISO pour ce type de protocole, offre 80 services environ pour manipuler à distance une dizaine de classes d'objets. Citons entre autres tous les services de lecture de variables, modification, signalisation d'événements ... D'une manière simplifiée, MMS peut être vu comme un protocole de communication entre clients et serveurs, ces derniers contenant un certain nombre d'objets manipulables à distance par les services MMS (commandes numériques, robots, automates programmables ...).

Les messageries industrielles sont utilisées dans des domaines très variés tels que les centrales électriques ou bien encore les stations service.

1.2.2 Applications

Les messageries industrielles sont utilisées dans un grand nombre de contextes et leur champ d'application s'étend de jour en jour. Nous allons brièvement expliquer deux d'entre elles et terminer par une énumération brève d'autres applications réalisées à ce jour.

1.2.2.1 Le tunnel sous la Manche

On peut voir le tunnel sous la Manche de différents points de vue : les travaux de génie civil avec les tunneliers de trois cents mètres de long ; l'installation électrique pour éclairer, alimenter les ventilateurs, etc ... ; enfin l'informatique et les réseaux utilisés pour commander et surveiller les équipements du tunnel. Car un

[1] **File Transfer Protocol**
[2] **Internet Message Access Protocol**
[3] **Simple Network Management Protocol**
[4] **Manufacturing Message Specification**

tunnel moderne ce n'est pas simplement une route ou des rails, mais c'est aussi de l'informatique et des réseaux : pour le tunnel sous la Manche, 178 équipements de télémétrie, 467 automates programmables pour la commande de l'éclairage, de la signalisation, de la ventilation,etc. Tous ces appareils sont reliés par des réseaux à des postes de commande, deux de chaque côté de la Manche. En fait le Tunnel est un excellent exemple d'une application industrielle répartie (plus de huit cents machines coopèrent, réparties sur une distance de trente kilomètres), avec des contraintes de temps sévères (1,7 sec. entre la détection d'une alarme et son affichage sur écran) et des contraintes de disponibilité (quelques secondes d'indisponiblité acceptées par an). Les protocoles utilisés pour communiquer avec tous ces dispositifs doivent permettre la lecture à distance de variables, leur modification, le lancement de programmes et la signalisation d'événements et ceci en des temps courts et de manière fiable.

1.2.2.2 Une station essence

Jusqu'à une période récente, les stations essence échappaient à l'informatisation et à la mise en réseau. Dans les stations modernes, la pompe à essence est équipée d'un micro-ordinateur et d'une interface réseau lui permettant de communiquer avec la caisse. La caisse est elle-même reliée à une machine UNIX, située dans un bureau, et à laquelle sont connectés d'autres équipements : le réservoir d'essence de la station, dont on peut connaître le niveau et la station de lavage (car wash). Toutes ces installations ou équipements sont vus comme des serveurs pouvant être commandés à distance, pouvant fournir des informations ou être téléchargés. La pompe à essence par exemple pourra être chargée avec les prix des différentes essences. En retour, elle fournira des statistiques sur le nombre de clients servis, le type d'essence le plus servi et le nombre de litres pompés.

1.2.2.3 Autres applications

Parmi les applications non conventionnelles, citons : la commande à distance de satellites, la communication entre centrales électriques, la commande de l'éclairage de pistes dans un aéroport, etc ... On peut aussi citer le GRID (calcul largement distribué) qui pourrait tout à fait tirer parti de ces architectures.

1.2.3 Le standard MMS

MMS, pour Manufacturing Message Specification, est un protocole d'application défini par l'ISO entre 1986 et 1990. Cinq ans ont été nécessaires pour spécifier un des plus gros protocoles de communication de l'ISO. Deux modèles fondamentaux sont à la base de MMS. D'une part une abstraction appelée VMD [5] qui permet de définir un équipement de production virtuel : par exemple ce n'est

[5]Virtual Manufacturing Device

ni un automate, ni une commande numérique mais la VMD s'applique aussi bien au premier qu'au second. Le VMD est donc un modèle abstrait d'un dispositif industriel.

D'autre part, les interactions entre programmes d'application sont vues comme des demandes de service d'un client à un serveur et des réponses du serveur au client. La grande majorité des services définis dans MMS suivent ce schéma de requête-réponse que l'on retrouve typiquement dans les systèmes de RPC (Remote Procedure Calls), comme par exemple NFS.

Les variables MMS peuvent être simples (booléens, entiers, réels, chaînes de caractères...) ou structurées (vecteurs et records). On peut lire et écrire des variables et en particulier accéder à des éléments spécifiques tels que des champs d'un record ou des éléments d'un tableau. La complexité des structures de données n'est pas limitée par MMS : on trouve souvent dans des applications automobiles des tableaux contenant des structures contenant elles-mêmes des tableaux.

Les programmes ou tâches MMS peuvent être démarrés, interrompus, relancés, réinitialisés à distance. Ils peuvent représenter des programmes de fabrication, des opérations comme la fermeture d'une vanne ou toute opération qui peut être lancée, arrêtée et reprise.

Deux classes de sémaphores sont introduites, l'une permettant d'assurer l'exclusion mutuelle entre clients accédant un objet ou groupe d'objets, l'autre permettant la synchronisation de clients.

La partie la plus complexe de MMS concerne la gestion des événements. Dans sa version la plus évoluée, MMS permet à un client de définir un événement, de spécifier une action à exécuter dans le serveur lorsque l'événement se produit et de préciser le nom d'un client pas nécessairement celui à avertir lorsque l'événement se produit.

D'autres classes d'objets existent mais nous ne les décrirons pas ici. Il s'agit des journaux, permettant d'archiver des données et des événements et des stations opérateurs, proposant un sous-ensemble de la fonctionnalité offerte par Telnet.

1.2.4 Historique

Les messageries industrielles existent depuis une vingtaine d'années. Initialement, elles étaient très rudimentaires, propres à un constructeur et surtout propres à chaque équipement. On peut dire que chaque équipement ou famille d'équipements avait sa propre langue, ce qui pour les utilisateurs posait un sérieux problème de coût, car les programmes informatiques qui les utilisaient étaient nécessairement spécialisés pour un type de dispositif. Aujourd'hui, quatre changements se produisent :

① une unification des protocoles : au lieu d'avoir une messagerie par famille d'équipements, on dispose d'une messagerie unique pour tous les équipements industriels d'un constructeur. Avec la norme MMS définie par l'ISO,

les utilisateurs ont maintenant à leur disposition un protocole unique, indépendant des constructeurs et qui s'applique à tous les équipements.

② un enrichissement en fonctionnalité : les messageries propriétaires sont en général simples et limitées dans leur capacité de commande à distance des machines. La messagerie MMS reprend non seulement la fonctionnalité offerte par les messageries existantes mais en plus introduit des mécanismes nouveaux.

③ une utilisation dans des domaines nouveaux : initialement cantonnées aux domaines du manufacturing et du contrôle de processus, les messageries industrielles ont été ensuite appliquées à des domaines nouvellement informatisés comme la voiture, la domotique, les tunnels.

④ une mise en oeuvre dans des équipements toujours plus petits. Le domaine classique des messageries industrielles est celui des automates programmables et commandes numériques, dont le prix permet l'utilisation de processeurs de la puissance d'un 8086. Les progrès continus dans la miniaturisation des processeurs permettent d'envisager l'intégration de ceux-ci directement dans des capteurs et actionneurs, donnant lieu à ce qui est appelé des capteurs intelligents, capables de traiter des données et de les communiquer via un réseau à des équipements de commande.

1.2.5 La recherche

Alors que dans les réseaux classiques la recherche se concentre sur les hauts débits (1 GigaBit/s et au dessus), dans le domaine des communications industrielles et en particulier sur MMS, la recherche porte sur le passage vers l'architecture TCP/IP d'une part. Le deuxième axe de recherche est partagé puisque ce sont les communications pour le multimédia.

Le premier axe de recherche donne lieu à trois grandes solutions : le portage de MMS à un niveau donné de l'architecture TCP/IP en utilisant des couches d'adaptation (RPC, RFC1006) ou directement au-dessus de TCP en portant le protocole [21] ou bien la définition d'un protocole fonctionnellement équivalent à MMS mais complètement objet et fonctionnant au dessus d'un bus logiciel [26][27].

Le deuxième axe donne lieu a des recherches sur la transmission de flux multimédia [57] et les deux axes peuvent être combinés avec la recherche sur les bus logiciels à capacités temps réel pour une gestion homogène des applications multimédia et des autres [62].

1.2.6 MMS et TASE.2

Le protocole TASE.2 que nous nous proposons d'étudier et de prendre comme base fonctionnelle dans ce travail, est une sur-couche de MMS. Il lui apporte

essentiellement des services avec des propriétés temporelles, absents et nécessaires dans le cadre de la production d'énergie.

Il est ainsi possible de demander des envois périodiques de données dans TASE.2. Il est aussi possible de demander de tamponner les données pendant un certain temps avant envoi des données, ceci prend tout son sens lorsque l'on sait que les variables surveillées correspondent à des capteurs surveillant un processus physique. En effet, il est rare qu'une seule des variables observées évolue, souvent ce sont toutes les variables corrélées à une même partie du procédé surveillé qui évoluent en même temps. Ce tamponnage permet d'éviter une avalanche de génération de données.

TASE.2 est un protocole massivement utilisé dans le domaine de la production d'énergie et il est en particulier utilisé par EDF distributeur, et à l'étude chez EDF producteur. C'est donc naturellement que l'on a pensé à lui comme référence fonctionnelle.

1.3 Plan de la Thèse

Dans un premier temps, nous rappelons le domaine des applications visées, à savoir la production et le transport d'énergie. Nous rappelons l'architecture des solutions existantes, faisons le point sur les contraintes et les besoins de ces applications. Nous concluons sur la nécessité de nouvelles solutions.

Nous présentons ensuite le protocole industriel TASE.2, avant de nous intéresser à sa richesse fonctionnelle au travers d'un portage d'un produit du marché. Ce travail permet de définir tout l'aspect fonctionnel de la solution recherchée, en s'assurant qu'il correspond bien aux besoins métier.

Ensuite nous détaillons nos solutions pour répondre au problème proposé, ce via deux prototypes, à base d'objets pour l'un, de composants pour l'autre. Ces deux prototypes sont opérationnels.

Enfin nous concluons sur l'intérêt de ce travail et sur les perspectives qu'il ouvre.

Chapitre 2

Médiation & Supervision dans le domaine de la production d'énergie

Sommaire

2.1 Introduction

Ce chapitre permet de faire le point sur le domaine applicatif visé dans le cadre de cette thèse. Il est important de bien cerner les besoins et le contexte pour spécifier, au final, un outil qui réponde à des besoins réels.

Le domaine applicatif est double : d'une part, au niveau supérieur, les échanges d'informations entre producteurs d'énergie,consommateurs et intermédiaires et

d'autre part, dans la centrale, les applications de contrôle-commande qui permettent de gérer les procédés de production d'énergie.

Ce chapitre est donc décomposé en deux parties qui présentent ce double domaine applicatif. Nous concluons sur les similitudes de ces deux domaines en ce qui concerne les besoins fonctionnels en terme de protocole de médiation.

2.2 Transport d'Énergie

2.2.1 Introduction

La deuxième mission d'un grand groupeénergétique comme l'est EDF, après celle bien connue de le production d'énergie au sein des centrales (électriques en particulier), c'est de distribuer cette énergie aux clients demandeurs. Il est donc nécessaire de mettre en oeuvre tout un ensemble de matériels, de logiciels, de protocoles et de procédures pour gérer cette énergie produite aux quatre coins de la France, d' éventuellement donner des ordres pour moduler la production en fonction de la demande et donc d'acheminer cette énergie vers la demande. Cette problématique est généralisable à toute production/gestion de "fluide" (eau/gaz/électricité/réseau de communication (FAI)/ téléphonie et plus généralement tous les services au public).

En France et pour l'électricité, ces fonctions sont à la charge du RTE [1], entité maintenant séparée de la partie Production d'EDF.

Dans la littérature et les offres commerciales, ces besoins sont adressés par ce qu'on appelle les architectures de médiation.

2.2.2 Architecture de Médiation

2.2.2.1 Introduction

L'utilisation d'internet comme medium de communication "à tout faire" s'étend très rapidement et dans tous les secteurs (énergie, banque, industrie, médecine ...) Beaucoup de terminaux, de matériels sont maintenant connectés sur le réseau et embarquent de plus en plus de données et de services. Il est nécessaire de fournir une couche qui pourra faire communiquer ces nouveaux terminaux intelligents avec les systèmes d'information des entreprises. Cette couche a principalement deux objectifs : d'abord, en gommant les différences entre tous ces matériels hétérogènes et en les pilotant d'une manière unique, elle diminue les coûts de l'administration, ensuite elle permet de mieux tirer parti de toute l'intelligence embarquée et de récupérer des données importantes voire vitales pour alimenter les applications métier de l'entreprise (typiquement, le RTE peut récupérer des informations sur les données de production par exemple).

[1] **R** éseau de **T** ransport d'**E**lectricité

D'un point de vue technique, ces deux objectifs sont atteints grâce à deux parties logicielles :

① un logiciel est installé sur les matériels distants, il permet l'installation et la configuration de composants logiciels, offre des fonctions de support système et des services à destination de clients distants.

② une couche logicielle pour collecter et traiter les informations en provenance des terminaux distants et les fournir aux applications clientes.

2.2.2.2 Définition

Le mot "médiation" est né dans le secteur des télécommunications, ou les systèmes de médiation sont maintenant reconnus comme un élément stratégique pour des applications métier comme la facturation ou la supervision. De nouvelles applications mettant en jeu des terminaux intelligents (et en particulier dans le domaine de l'énergie) font aujourd'hui face à des problèmes similaires. Cependant, les solutions télécom ne sont pas appropriées car ne prenant pas assez en compte les terminaux.

La médiation, c'est le fait de collecter et de traiter des données métier depuis des terminaux en réseau et les distribuer à des applications métier en "temps réel".

Une architecture de médiation, c'est l'infrastructure qui intègre des terminaux distribués largement sur le réseau et les systèmes d'informations via Internet au sein d'un réseau privé à grande échelle. La figure 2.1 illustre ce type de réseau.

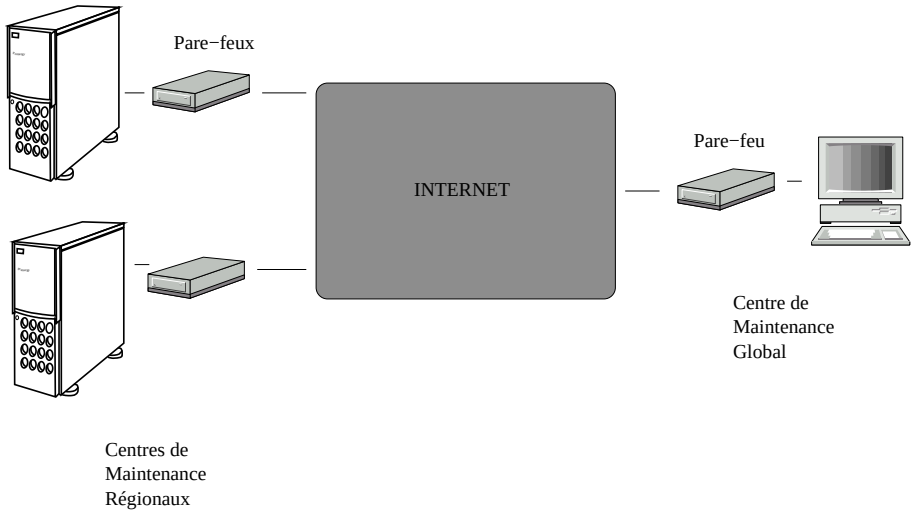

FIG. 2.1: Architecture de médiation

2.2.2.3 Contraintes & Caractéristiques

Les architectures de médiation doivent adresser les contraintes suivantes :

- **Passage à l'échelle** Le nombre de terminaux impliqués est très grand. De plus, dans certains cas, ces terminaux rejoignent et quittent le réseau très souvent. Le volume de données échangé est lui aussi très grand. Cela conduit à des contraintes techniques fortes sur la bande passante et sur la puissance de calcul nécessaire au traitement des données.
- **Souplesse** L'architecture de médiation doit s'adapter aux caractéristiques de l'environnement et aux besoins des utilisateurs. L'infrastructure doit être capable de répondre à l'arrivée de nouveaux terminaux amenant de nouveaux services. Cela nécessite, pour être efficace, des capacités de reconfiguration dynamique.
- **Coûts** Le coût de l'architecture doit rester acceptable. Cela parait aujourd'hui faisable, étant donnés les évolutions et la standardisation de tous ces composants.

Les protocoles de médiation ont les caractéristiques suivantes :

- Virtualisation des ressources accédées à distance,
- Actions déclenchables à distance,
- Gestion d'événements,
- Fiabilité,
- Sécurité,
- Ponctualité.

Dans le cas du Réseau de Transport d'Électricité, l'architecture de médiation doit être capable de s'appuyer sur Internet puisque les ordres sont échangés avec les centrales en France mais aussi avec des partenaires au niveau européen. Ceci implique des contraintes de sécurité. Au niveau fonctionnel, on doit pouvoir échanger des informations de production, de demandes de production, le tout avec des contraintes temps réel (au sens de la ponctualité).

Les contraintes et caractéristiques du domaine de la médiation pour le transport de l'énergie sont génériques. On les retrouve dans les architectures informatiques dédiées à la production d'énergie comme nous allons le voir ci-après.

2.3 Production d'énergie

Parmi les activités d'EDF, la production d'énergie occupe une place primordiale, les procédés s'y rattachant ont des caractéristiques communes. Le schéma 2.2 dépeint le procédé de production d'électricité d'une centrale thermique, un exemple représentatif des fonctions qu'il va falloir piloter. Ce pilotage est effectué

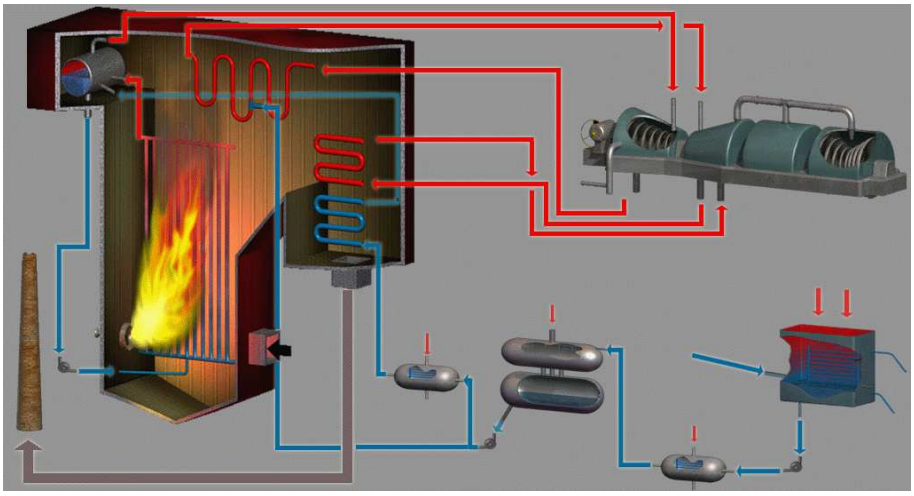

FIG. 2.2: Procédé de production d'une centrale électrique

par des équipes de conduite qui ont à leur disposition des systèmes de contrôle commande et d'informatique industrielle.

Dans les centrales thermiques classiques, la production d'électricité est réalisée à partir d'un cycle eau/vapeur. De l'eau, en présence d'une source chaude (une chaudière alimentée par du charbon, du gaz, du fioul...) est vaporisée et conditionnée pour avoir les caractéristiques thermodynamiques assurant un rendement optimal. Cette vapeur est ensuite injectée dans une turbine qui se met en rotation. L'arbre de cette dernière est solidaire d'un alternateur. C'est ce dernier élément qui, à l'instar d'une dynamo, produit l'électricité.

Une fois que cette vapeur a traversé la turbine, il est nécessaire de lui faire subir une nouvelle transformation thermodynamique avant de la réinjecter dans la chaudière. Cela se fait notamment par des condenseurs qui vont assurer sa liquéfaction.

Pour réaliser ce cycle, il existe de nombreux systèmes dans la centrale : des pompes et notamment des pompes d'alimentation qui assure la circulation de l'eau dans le circuit principal, des vannes qui fournissent les appoints, les régulations de débits, les régulations de température..., des réservoirs d'eau, de fioul, des ponts roulants pour acheminer le charbon, des broyeurs de charbon..., des disjoncteurs, des transformateurs, des cellules électriques, des redresseurs, des onduleurs...

Le processus simplement présenté sur le schéma précédent s'avère être d'une grande complexité tant par les systèmes mis en jeu que par la subtilité d'utilisation de certains organes.

Les équipes de conduite maîtrisent cette complexité : elles ont les compétences requises pour assurer le pilotage de la centrale en manuel. Cela dit, il existe de plus en plus de systèmes qui les aident à réaliser cette tâche en améliorant même

certains indicateurs.

Ce sont les systèmes de contrôle commande et d'informatique industrielle qui assurent cette " aide " par :

- ➡ l'acquisition des données du procédé (mesure de température, mesure de débit, mesure de pression, état de vannes -ouvert ou fermé-, état de disjoncteur...),
- ➡ la régulation,
- ➡ les actions de sécurité (ne pas lancer une pompe si le circuit n'est pas en eau, ne pas autoriser le remplissage d'un réservoir déjà trop plein...),
- ➡ la présentation des informations de synthèses aux équipes de conduites (mesures, alarmes, courbes...),
- ➡ la prise en compte des commandes des équipes de conduite (fermeture de vanne, consigne de régulation...).

Ce pilotage de la production d'électricité doit de plus être réalisé selon des contraintes fortes de fiabilité. La solution choisie est fournie par des choix d'architecture des systèmes, dans la plupart des cas, une redondance des fonctions est mise en oeuvre.

Il est également nécessaire de choisir des solutions pérennes, les systèmes élaborés ont une durée de vie de trente ans en moyenne.

2.3.1 Contraintes sur l'Architecture Informatique

Ces contraintes de fiabilité et de pérennité pèsent de tout leur poids sur les systèmes de contrôle commande et sur les systèmes d'informatique industrielle (surtout sur ces derniers compte tenu de la relative jeunesse de l'informatique, souvenez-vous de votre PC d'il y a trente ans !). Ainsi, le choix d'éléments " sur étagère[2] ", s'il a pu être effectué pour certains systèmes de contrôle commande, n'était pas envisageable pour les systèmes d'informatique industrielle.

Aujourd'hui, choisir des éléments " sur étagère " semble être envisageable compte tenu de la professionnalisation de l'informatique (qui était jusqu'à présent de l'artisanat) grâce aux techniques de génération de code, de réutilisation de l'existant, de suivi de versions, de procédure d'assurance qualité.

Dans l'avenir, les architectures vont être orientées réseau pour adresser les contraintes inhérentes de répartition géographique et d'extensibilité, et pour cette même raison, orientées composant.

Actuellement, les efforts de configuration des systèmes pour l'ajout d'un nouveau composant restent importants. Côté réseau, l'ajout d'un capteur ou d'un actionneur (pompe, vanne, moteur...) nécessite a minima une déclaration de ce nouveau composant dans un système centralisé et dans certains cas l'arrêt du

[2]Élément " sur étagère " : élément disponible pour un grand nombre d'utilisateurs, non réalisé pour un usage unique.

système de contrôle commande. De même, l'ajout d'un nouveau poste de présentation des données aux équipes de conduite nécessite a minima l'arrêt du système et une déclaration explicite de la nouvelle machine. Coté composant, compte tenu de l'aspect monolithique de certaines architectures de systèmes d'informatique industrielle, changer un composant revient (en caricaturant) à refaire le système...

Il est donc nécessaire d'étudier les nouveaux mécanismes que l'informatique actuelle nous propose, et cela nous amène directement à envisager des solutions réparties comme des bus logiciels. De plus, et pour répondre à la contrainte de pérennité, ces solutions doivent être élaborées à partir d'éléments " sur étagère " comme recommandé par EDF R&D [2].

Les caractéristiques plus directement liées aux processus eux-mêmes, comme les contraintes temporelles, nécessitent l'utilisation de systèmes supports temps réel (systèmes d'exploitation temps réel), c'est-à-dire de systèmes dans lesquels les temps d'exécution sont aussi importants que l'action effectuée proprement dite, et sont bornés. Typiquement, un temps de réponse d'une boucle de régulation doit être borné et, de même, certaines commandes des équipes de conduite nécessitent un temps maximum de prise en compte.

En résumant les contraintes et les besoins des systèmes de contrôle commande et d'informatique industrielle des centrales de production d'électricité, ces systèmes doivent présenter des architectures qui assurent la répartition, la fiabilité, la pérennité et également des garanties de qualité de service temporelle.

2.3.2 Architecture de Systèmes de Contrôle-Commande

L'architecture des systèmes dans les milieux industriels suit quasiment tout le temps la même logique de structuration en niveaux, le modèle CIM [3] [29]. Nous nous limitons ici aux trois premiers niveaux, qui sont ceux qui interviennent directement dans la définition des besoins de répartition (une architecture d'interconnexion avec les niveaux supérieurs en rupture avec les principes définis sur ces trois premiers est souvent admise, voire préférée afin de délimiter clairement le domaine industriel du reste du système d'information de l'entreprise). La figure 2.3 donne un aperçu de cette architecture. Les niveaux 0, 1 et 2 qui y apparaissent sont décrits dans la suite de ce paragraphe.

2.3.2.1 Niveau 0 : capteurs et actionneurs

La numérotation des niveaux dans le modèle CIM croît en s'éloignant du procédé à piloter. Ainsi les éléments du niveau 0 sont en contact direct avec le procédé industriel. Ce niveau représente l'ensemble des capteurs (température, pression, débit...) et actionneurs (vannes, pompes...) qui permettent d'agir directement sur les machines ou de relever des mesures.

[3] **C**omputer **I**ntegrated **M**anufacturing

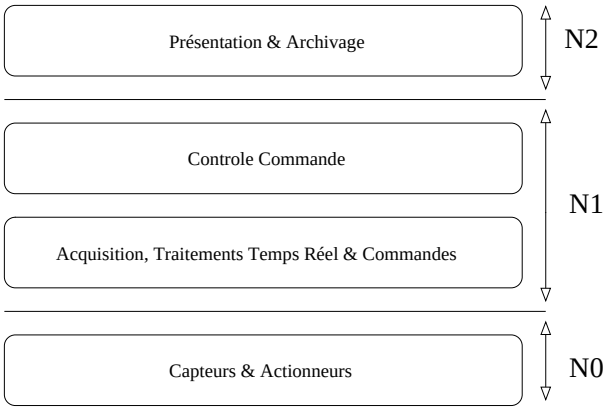

FIG. 2.3: Le modèle CIM

Le niveau 0 assure la conversion de grandeurs physiques (pression, débit, position de vanne, position de pompe) en grandeurs électriques (signaux analogiques -ANA- ou numériques -TOR-[4] -) non interprétés et non mis en forme à destination du niveau supérieur, le niveau 1. Et inversement, il assure la conversion de grandeurs électriques (commandes émanant du niveau 1) en grandeurs physiques (fermeture ou ouverture de vanne, enclenchement ou déclenchement de pompe).

Les informations entre le niveau 0 et le niveau 1 sont véhiculées par du " fil à fil ". Un capteur de température remonte son information sur deux fils de cuivre. Dans l'avenir, l'utilisation de bus de terrain serait une alternative au " fil à fil " actuel, car la solution " fil à fil ", même si elle a l'avantage de la robustesse, est gourmande en câble et présente un encombrement supérieur (les chemins de câbles actuels imposent des contraintes sur le génie civil). De même l'ajout, le calibrage (car le protocole serait plus riche) d'un capteur ou d'un actionneur seraient facilités par un réseau de terrain aux dépends d'un coût capteur ou actionneur supérieur compte tenu de " l'intelligence " embarquée. De plus, les dernières évolutions des bus de terrain sont hertziens (en particulier WiFi) d'où un gain évident au déploiement (modulo les tests de compatibilité électromagnétiques).

2.3.2.2 Niveau 1 : contrôle-commande, acquisition & traitements temps réel

Le niveau 1 correspond au contrôle commande. Le contrôle commande est assuré principalement par des automates programmables. De façon très grossière, ils exécutent une boucle qui correspond à :

- ➡ acquérir les informations en provenance du niveau 0 (capteurs et actionneurs) et du niveau 2 (commandes),

[4] **T**out **O**u **R**ien.

- ➡ effectuer des traitements,
- ➡ positionner les sorties à destination des niveaux 0 et 2.

Les traitements peuvent être classés en trois grandes catégories :

① **Les traitements de mesure** qui assurent juste le relais de l'information du niveau 0 vers le niveau 2. Typiquement, des valeurs correspondant à des capteurs de températures peuvent être directement envoyées au niveau 2 pour analyse statistique par un système périphérique. Dans ce cas l'automate est principalement utilisé pour ses capacités d'acquisition.

② **Les traitements de régulation** qui assurent la bonne commande de certains organes ou certaines parties du procédé. Typiquement, le débit d'eau est régulé dans le circuit principal. Le niveau 2 intervient pour fixer des points de consigne, le niveau 0 reçoit les commandes à réaliser sur les actionneurs. Dans ce cas, l'automate est pleinement utilisé : acquisition, calcul et commande.

③ **Les traitements logiques** qui assurent la réalisation de calculs logiques. Typiquement, inhibition de commande d'enclenchement de pompe en cas de défaut sur une pompe, synthèse d'alarme pour le niveau 2, suivi de bons enchaînements d'actions... Là encore, l'automate est pleinement utilisé : acquisition, calcul et commande. La moindre complexité des traitements logiques par rapport aux traitements de régulation autorise des temps de cycle inférieurs.

Tous ces traitements peuvent être imbriqués les uns dans les autres et posséder des temps de cycles propres.

Le niveau 1 a donc un rôle dans l'exécution de traitements systématiques au plus proche du procédé, de synthèse/filtrage des informations vers le niveau 2 (alarmes, mesures particulières, remontée de défaut), de prise en compte des commandes (fermeture de vannes, point de consigne d'une régulation...)

Par ailleurs, bien que dans l'ensemble ces automates soient des éléments " sur étagère ", dans certains cas (les essais, la surveillance ou le diagnostic, demandant une analyse très fine), il est possible de trouver des systèmes propriétaires.

2.3.2.3 Niveau 2 : présentation et archivage

Enfin, le niveau 2 correspond au niveau supervision et conduite. C'est à ce niveau principalement que les équipes de conduite interviennent. C'est également à ce niveau que l'on va traiter l'archivage des données dans des bases de données classiques.

Le niveau 2 est constitué de systèmes ayant une interface homme machine qui permet aux équipes de conduite de prendre connaissance de l'état du procédé à distance (supervision) et de passer des commandes sur le procédé (conduite).

A ce niveau, les systèmes de supervision et de conduite sont encore souvent conventionnels (pupitre avec des boutons et des enregistreurs papiers) et les systèmes informatiques présents sont majoritairement propriétaires.

2.3.3 Aspects fonctionnels

Ces systèmes de niveau 1 sont donc chargés de l'acquisition de données " bas niveau " : températures (signaux analogiques statiques), vibrations (signaux analogiques dynamiques), vannes (signaux logiques TOR). Les données acquises fournissent des informations de niveau N1 <date, état, valeur> auxquelles certains traitements de niveau 1 et de niveau 2 sont abonnés. Dans l'autre sens, les commandes provenant des postes de conduite sont interprétées et exécutées sur les procédés.

Au niveau supérieur, ces données vont être transformées en informations de niveau N2 qui synthétisent les données en entrée, et les regroupent éventuellement (par objet métier par exemple, ou bien encore par localisation). Ces données sont susceptibles d'être archivées dans une base de données pour présenter un historique de l'évolution (analyse a posteriori et traçabilité des actions effectuées).

Idéalement, plusieurs années de données peuvent être ainsi conservées dans cette base.

Les fonctions suivantes sont proposées (là encore idéalement) : compression, inhibition, élaboration. La fonction de compression sert à limiter la taille de cette base de données : il est possible d'éliminer des données qui ne présentent pas de différences significatives avec les données anciennement stockées. La fonction d'inhibition permet de stopper l'enregistrement d'une information (l'archivage des valeurs d'un capteur reconnu défaillant est inhibée). Enfin la fonction d'élaboration sert aussi à limiter la taille de la base en faisant un filtrage des données en fonction du mode d'exploitation de la centrale. Typiquement, si la turbine est mise au ralenti, ce mode crée plus de vibrations que la normale, en supposant que le système s'intéresse aux vibrations à la vitesse nominale, les valeurs acquises durant cette phase n'ont alors pas d'intérêt.

La figure 2.4 résume l'architecture type d'un système informatisé N1 et N2 :

Il est possible de trouver une redondance de certains éléments (par exemple les calculateurs du niveau 1) et ce pour des raisons de fiabilité.

2.4 Profils des systèmes distribués

2.4.1 Aspects généraux

Il est question de systèmes de contrôle commande et d'informatique industrielle et de systèmes de régulation, c'est-à-dire les systèmes qui vont mettre en œuvre toutes les fonctions décrites précédemment :

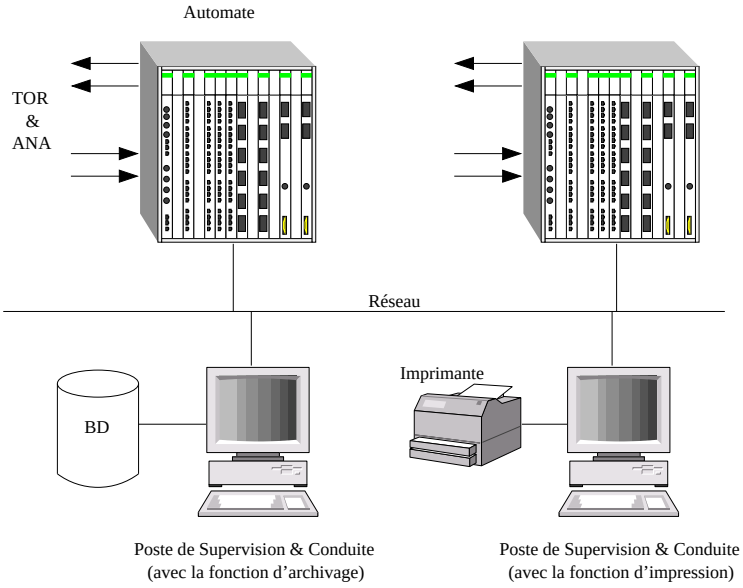

FIG. 2.4: Architecture type d'un système N1/N2

- acquisition de données à partir de capteurs,
- commande en réponse à un traitement logique ou de régulation,
- commande en provenance de l'interface homme machine à destination des actionneurs,
- point de consigne ou commande de synthèse en provenance de l'interface homme machine
- échange de données entre producteurs et médiateurs, distributeurs

Ces applications ont des contraintes distribuées fortes car :

- les capteurs et actionneurs mis en jeu sont distribués sur des sites à large échelle (une centrale de production électrique peut s'étendre sur plusieurs hectares),
- le contrôle commande est souvent trop important pour tenir dans un seul automate (l'application de contrôle commande est une application distribuée),
- la salle de conduite (contenant les interfaces homme machine) est éloignée de la salle contenant les systèmes de contrôle commande
- les fonctions de supervision et de conduite sont réparties sur différents postes (par groupes de fonctions par exemple ou pour la consultation à plusieurs de parties différentes du procédé ou bien encore pour des raisons de tolérance aux pannes) et sont réparties dans différentes régions/pays dans le cas de la médiation.

2.4.2 Contraintes de fiabilité

Les fonctions des applications de production sont souvent séparées en deux catégories distinctes : les fonctions vitales et les fonctions importantes. Aucune fonction vitale n'est contrôlée par l'informatique dans la centrale. Dans le cas de la médiation aucune fonction n'est vitale.

Les contraintes de fiabilité exigées influent sur les deux aspects de l'architecture : le matériel et le logiciel. Au niveau matériel, les calculateurs du niveau 1 sont souvent redondés et le couple architecture matérielle / architecture logicielle doit permettre le basculement aisé sur le calculateur de secours en cas de défaillance.

Dans le cas du niveau médiation, la fiabilité est la aussi assurée par la redondance des systèmes, en utilisant par exemple le fait que les informations transitent sur des systèmes fortement maillés. Les systèmes de médiation peuvent utiliser aussi un réseau distinct et propriétaire pour l'échange des données (cela apporte aussi des garanties de sécurité/confidentialité).

2.4.3 Contraintes temps réel/temporelles

Il y a quatre façons de gérer les données transmises, que ce soit au niveau de la production ou de la médiation : synchrone, asynchrone, événementielle et cyclique.

Le modèle synchrone correspond à un appel de procédure à distance : l'appelant est bloqué jusqu'à la réception des données. Dans le modèle asynchrone, l'appelant continuera à s'exécuter et sera prévenu lors de l'arrivée des données par une interruption. Le modèle événementiel indique que c'est le processus physique qui génère des données à des moments précis, sans demande du niveau supérieur, et il lui envoie. Quant au modèle cyclique, il correspond à une interrogation ou envoi à intervalles réguliers.

Le mode de fonctionnement cyclique permet d'aller interroger les processus qui ne produisent pas d'événements particuliers dans le cas de la production, dans le cas de la médiation c'est plutôt le producteur qui envoie les données. La période du cycle peut prendre des valeurs très variables : 50 ms, 1 seconde, 5 secondes, 10 secondes, 1 heure, 1 jour... Évidemment, plus le cycle est court et plus le volume de données à traiter sera important.

En ce qui concerne les contraintes temps réel proprement dites, elles s'appliquent sur les informations en entrées et sorties du niveau 1, elles sont de plusieurs types : échéances temporelles ou débit de données, à valeur moyenne et à valeur maximale, locales (s'appliquant à un seul niveau du modèle CIM) ou de bout en bout (traversée de toutes les couches). Ce dernier type de contrainte suppose une architecture répartie, avec des calculs répartis, pour adresser un délai de bout en bout, il faut assurer des temps de parcours bornés d'un bout à l'autre de l'architecture, entre la fonction qui produit la donnée et la (les) fonction(s)

qui la consomme(nt).

Sémantiquement, une application peut donc nécessiter le respect d'échéances ou d'un débit. Même si dans les deux cas, la solution est souvent la même, à savoir qu'un débit suffisant permet en général de satisfaire les contraintes de type échéances, les deux correspondent à des besoins différents.

Pour les données échangées au niveau de la médiation, on parle plutôt de contraintes temporelles car les besoins ne sont pas les mêmes que dans la production. Les délais peuvent être dépassés et d'ailleurs les protocoles de médiation prennent en charge ces contraintes. D'autre part le temps réel est incompatible encore maintenant avec en particulier le fait de pouvoir s'exécuter au dessus d'Internet.

2.4.4 Architecture système distribuée requise

Pour répondre aux spécifications précitées, un système d'exploitation temps réel s'avère souvent nécessaire (avec un certain nombre d'abstractions : tâches, mémoire partagée, files de messages, signaux, sémaphores) même si le raisonnement en termes de débit autorise parfois l'utilisation de systèmes choisis pour leurs performances. Il reste à choisir parmi les différents produits sur étagère proposés en fonction de leurs propriétés : architecture micro-noyau (pSOS+ [78], VxWorks [79], ChorusOs ...), systèmes conformes POSIX[5] [14] ((LynxOS [87]...). Pour faire ce choix, les utilisateurs de ces applications mettent en avant plusieurs critères [2].

Le système d'exploitation choisi doit être connu dans le domaine industriel et connu de plusieurs intégrateurs, en particulier il a déjà fait ses preuves dans des applications développées (donc qu'un savoir-faire sur ce produit existe déjà au sein de l'entreprise). Il offre des gages de pérennité qui se traduisent par l'ancienneté du produit, l'ancienneté du constructeur, la politique du constructeur. Les personnes interrogées ont particulièrement insisté sur ce dernier point. Certaines applications nécessitent également d'avoir accès aux sources du système d'exploitation, il est préférable qu'il existe déjà des BSP[6] pour les cartes qui vont être utilisées et que ce système supporte un grand nombre de processeurs et de composants matériels.

Pour le système proprement dit, il doit être fiable, performant (il existe des informations sur ses performances et elles correspondent aux besoins applicatifs, nous avons à disposition les temps des principaux appels système, les temps de changement de contexte et le temps de latence dans le traitement des interruptions). Il doit supporter les standards de programmation (POSIX) et les protocoles de communication classiques (pile TCP/IP obligatoire). Il doit également offrir le support d'un bus logiciel (ORB [7]).

[5] POSIX 1003.1b (extensions temps réel), 1003.1c (threads)

[6] **B**oard **S**upport **P**ackage, logiciel de base pour faire fonctionner une carte processeur.

[7] **O**bject **R**equest **B**roker

Enfin, en ce qui concerne l'ingénierie, les outils de développement doivent être pratiques et suffisants. Il doit exister des outils d'analyse et d'optimisation des performances. Il est souhaitable que les ingénieurs impliqués puissent suivre une formation sur l'utilisation de cet environnement de développement. Il existe un service technique compétent et efficace, la documentation doit être complète et bien structurée. Le coût entre bien sur également en ligne de compte lors du choix du système.

Les développements dans le domaine du temps réel à EDF sont essentiellement réalisés en C/C++ et en ADA. ADA est souvent utilisé pour des raisons de sûreté car il permet de faire de la compilation séparée, de plus beaucoup de vérifications sont faites durant cette phase (typage, dépassement de bornes...). En fait le choix d'ADA dépend de la taille du développement et du nombre de personnes qui vont y participer. Les langages supportés par les OS retenus ont donc un rôle important.

2.4.5 Profil communications

Au niveau réseau couches basses, Ethernet dans sa version 100 Mb/s connecté et full-duplex est aujourd'hui très utilisé, souvent avec d'autres réseaux, déterministes. Pour ce qui est des protocoles, une évolution récente a mené à une utilisation massive de TCP/IP. La pile OSI est également utilisée, avec la messagerie industrielle MMS (donc nécessité d'avoir une pile OSI ou pour les nouveaux systèmes d'utiliser la RFC1006 [45] ou la RFC2126 [39] plus récente qui assurent une glue entre TCP/IP et les couches hautes du modèle OSI), essentiellement pour les couches supérieures et pour communiquer avec l'extérieur.

Les architectures de médiation utilisent quant à elles des accès via modem RTC, RNIS mais on peut aussi imaginer des accès via ADSL ou via des tehnologies sans fil.

Comme les machines sont hétérogènes (au niveau matériel et système), il est souvent fait usage de l'abstraction sockets que l'on retrouve implantée dans la majorité des systèmes.

Dans le cas de la production essentiellement, il est nécessaire de disposer de modes de diffusion (voir les travaux de C. Toinard sur les diffusions multicast [52]), ne serait-ce que pour effectuer les synchronisations d'horloge ou bien encore pour gérer la redondance des applicatifs ou des processeurs (dans ce cas, du 1 vers n avec des groupes à composition dynamique). De plus, les études de besoins font ressortir que les protocoles de diffusion doivent avoir les propriétés suivantes : ordre local à l'émetteur, gestion de groupes, groupes statiques avec tous les membres connus, génération d'événements sur entrée et sortie d'un membre du groupe.

Au niveau de la médiation et pour assurer la sécurité, l'environnement nécessite également la mise en œuvre de confidentialité et, au niveau applicatif, la présence d'un système d'authentification des accès, accès qui peuvent être exté-

rieurs (gestion au niveau national, téléconduite). Ceci conduit tout naturellement à la mise en place d'un pare-feu entre l'architecture applicative sur le site de production d'énergie et le monde extérieur.

2.4.6 Profil bus logiciel

En ce qui concerne la partie supervision de la production d'énergie, les demandes des utilisateurs concernant une utilisation possible d'un bus logiciel pour les applications d'informatique industrielle correspondent aux points ci-dessous :

- Il faut se poser la question de la cohabitation entre les contraintes temps réel et le fonctionnement du bus logiciel. En d'autres termes, dans quelle mesure les propriétés assurées par le système temps réel sous-jacent sont conservées par le bus logiciel mis en œuvre. Par exemple, l'utilisation d'un bus logiciel à usage général peut faire craindre le non-respect de ces contraintes. CORBA-RT [32] définit un cahier des charges et un canevas pour la mise en œuvre d'un bus logiciel à objets dans ce type de contexte.
- Il faut également que le bus logiciel soit disponible sur un grand nombre de systèmes d'exploitation. En effet, c'est bien l'un des buts de l'utilisation d'un bus logiciel que de permettre de masquer l'hétérogénéité des matériels et systèmes d'exploitation utilisés par l'application.
- Enfin, il doit rester facile à utiliser et bien maîtrisé : nombre de recopies de tampons minimal, performances, nombre maximum de serveurs et d'objets pouvant être lancés en parallèle sur une machine. Évidemment il doit être déterministe, tant du point de vue temporel que logique.

,

Ces contraintes sont très proches de celles évoquées pour les architectures de médiation. L'on s'aperçoit que les attendus dans le double-domaine visé sont très proches, que ce soit en termes fonctionnels ou en termes techniques (virtualisation de procédé, passage à l'échelle, sécurité, aspects "temporels", etc...)

De plus, l'organisation même des procédés à superviser nous amène à penser aux objets répartis pour adresser le type d'application citées. En effet, les bus à objets répartis apportent des propriétés intéressantes pour ce type d'application, à savoir l'extensibilité, la réutilisabilité et une adéquation entre les objets supervisés et les objets systèmes qui peut permettre des développements plus rapides. On pense donc à une approche système réparti et une approche composants.

Au moment de l'étude, les bus logiciels (orientés RPC comme OSF-DC ou orienés objets distribués comme CORBA ou Java-RMI) étaient largement utilisés. Les Web Services étaient émergents, il n'existait pas d'implantation solide de SOAP-XML, cette solution n'a donc pas été retenue à l'époque mais serait très pertinente aujourd'hui.

De plus l'utilisation de CORBA nous permettait espérer passer dans un deuxième temps à un CORBA RT (TAO, ORBExpress, eORB ...), TAO est

maintenant assez mur en ce qui concerne le temps réel. De plus le choix de CORBA nous permettait une évolution vers les composants.

2.5 Conclusion

Il nous faut donc trouver une classe d'applications permettant de concilier les besoins dans la supervision de la production dans la centrale, avec des contraintes temporelles assez fortes et les besoins dans la médiation inter-sites, avec des besoins similaires (besoins suffisamment proches pour espérer trouver un candidat) pour la régulation de la production. Elle doit être architecturée à base d'objets ou de composants pour atteindre les besoins d'extensibilité et d'ingénierie. Les messageries industrielles abordées dans le chapitre suivant présentent des caractéristiques intéressantes mais sont basées sur des solutions par passage de messages conventionnels.

Il nous faut donc définir un protocole équivalent fonctionnellement mais conçu avec une technologie objets et/ou composants.

Chapitre 3

La messagerie industrielle I.E.C./T.A.S.E.2

Sommaire

3.1 Introduction

Nous avons vu dans le chapitre précédent que les besoins dans supervision de la production et la médiation sont très proches et correspondent aux fonctionnalités offertes par les messageries industrielles. Nous nous intéressons donc aux messageries industrielles, dans les travaux normalisés, la première version (interne) correspond à UCA.2 [9] tandis que la deuxième correspond à **TASE.2**.

[1] [55][54][56] ,encore dénommé **ICCP** [2], récemment normalisé, initialement développé pour les échanges inter-centres de conduite, dont les principes généraux et les caractéristiques techniques sont décrits dans notre rapport sur le portage d'un serveur TASE.2 sur le systme temps réel pSOS+ [90].

Sémantiquement, ces deux messageries ne sont pas fondamentalement différentes. Ce qui les sépare, c'est la prise en compte de la sécurité dans IEC/TASE.2 (ce qui est logique puisqu'elle se déploie hors de la centrale, sur un réseau ouvert et non protégé) et la possibilité de faire des communications 1 vers n dans UCA.2 (cependant n n'étant pas très grand dans la pratique , il est acceptable de simuler ces communications par n relations 1 vers 1).

En conséquence, le choix de la messagerie TASE.2, est suffisamment générique pour être appliqué à UCA.2 voire à toute messagerie industrielle.

TASE.2 est un protocole de télé-conduite. Il est prévu pour permettre la transmission de données (TéléMesure et TéléSurveillance), de commandes, de plannings de production, et de messages en ASCII. TASE.2 est une messagerie industrielle normalisée, c'est une façon classique de résoudre le problème de l'échange de données en milieu industriel.

Dans le cadre de notre évaluation des différentes solutions de répartition susceptibles de convenir aux besoins des applications industrielles EDF, nous avons d'abord évalué TASE.2 sous la forme d'un portage d'une souche logicielle TASE.2 serveur sur un noyau temps réel pSOS+ [3] (certains flux comme le télé-réglage ou les ordres d'urgence ont des caractéristiques pouvant nécessiter dans l'architecture cible un environnement temps réel). La souche serveur portée ne peut être divulguée pour des raisons de confidentialité et les détails du portage ne peuvent être communiqués pour des raisons de priopriété intellectuelle d'EDF R&D. Toutefois les informations que nous pouvons fournir sont suffisantes pour décrire le travail de thèse.

Ce travail commence par une étude complète de la norme TASE.2, qui nous permet, déjà, de dégager les principes fonctionnels de cette messagerie. Il est utile ensuite de comparer, dans le chapitre suivant, la norme, avec les produits censés s'y conformer.

3.2 Architecture

TASE.2 est une sur-couche de la messagerie industrielle **ISO-MMS** [4] [58]. La dénomination ICCP est aussi utilisée pour référencer TASE.2, elle est souvent utilisée aux États-Unis.

[1] **T**elecontrol **A**pplication **S**ervice **E**lement **2**

[2] **I**nter-**C**ontrol **C**enter **P**rotocol

[3] le noyau pSOS+ avait été retenu lors de la phase de choix d'un système d'exploitation temps réel pour les applications de contrôle commande EDF, cf.

[4] **M**anufacturing **M**essage **S**pecification

Les services offerts par TASE.2 reposent donc sur ceux de ISO-MMS. Le concept de machine virtuelle de fabrication, VMD, permet de représenter une centrale sous la forme d'un équipement virtuel. Cette centrale virtuelle, modélise alors les ressources réelles d'une centrale quelles que soient leur nature et leurs caractéristiques spécifiques. L'accès à l'état des ressources réelles s'effectue alors de façon uniforme à travers des abstractions commodes pour la conception de logiciels de supervision et de régulation de production.

Les interactions TASE.2 fonctionnent sur le principe du modèle client/serveur. TASE.2 est généralement considéré comme un protocole pour le "temps réel". Mais comparé aux caractéristiques fonctionnelles des bus de terrains [3], le temps réel adressé n'est pas un temps réel à contraintes strictes.

3.3 Couches de communication associées à TASE.2

Le protocole TASE.2 est situé dans la 7^{eme} couche du modèle ISO, au-dessus de la messagerie industrielle MMS. MMS, dans sa version "complètement ISO" est prévu pour fonctionner au-dessus du profil MAP (Manufacturing Automation Protocol) [58]. Pour tenir compte de certaines critiques formulées à l'encontre de MMS-MAP, comme sa complexité d'implantation ou son coût élevé, certains industriels ont développé des solutions mixte MAP-TCP/IP. Ces solutions s'appuient sur la couche d'adaptation RFC 2126 [39] (anciennement RFC1006 [45]) placée au-dessus du protocole de Transport fiable TCP orienté connexion. Dans ce cas, seulement les couches 5 à 7 de la pile OSI du profil MAP sont mises en oeuvre. La couche d'adaptation RFC 2126 au-dessus de TCP/IP permet d'utiliser un protocole de Transport OSI minimal de classe 0.

TASE.2 est prévu pour fonctionner avec les deux profils de communication précédents (OSI complet ou utilisation de TCP/IP conformément à RFC2126).

3.4 Interactions Client/Serveur

Le modèle d'interaction sélectionné pour TASE.2 est intimement lié à celui de MMS. C'est le modèle Client/Serveur. Il reprend les principes de la modélisation objet de MMS en spécifiant des objets directement adaptés à l'échange de données de production d'énergie. Par exemple, TASE.2 définit un centre de contrôle virtuel (VCC[5]) construit sur l'abstraction VMD de MMS.

En tant que serveur, conformément à la norme MMS, un centre de contrôle rend ses ressources réelles visibles sous la forme de ressources virtuelles. Les ressources visibles peuvent n'être qu'un sous-ensemble de ses ressources. Plusieurs clients peuvent accéder à ces données, en fait d'autres centres de contrôle. Chaque

[5] **V**irtual **C**ontrol **C**enter

client peut avoir une vue différente. Les clients peuvent être des serveurs eux-mêmes.

Deux types d'interactions sont définies dans le modèle TASE.2 :

① Une interaction d'un client vers un serveur est appelée opération. Elle correspond à un schéma de type service confirmé de la couche MMS. Le client attend donc un résultat pour ce type d'interaction. *Exemple : le client demande la valeur d'une variable au serveur.*

② Une interaction d'un serveur vers un client est appelée action. Elle correspond à un schéma de type service non confirmé de la couche MMS, comme *UnsollicitedStatus* ou *InformationReport*. *Exemple : Le serveur signifie au client la fin d'une association.*

3.5 Objets TASE.2

Les objets TASE.2 représentent tous les concepts utilisables dans la norme et peuvent être aussi bien des vues virtuelles d'équipements physiques (Device, ...) que des constructions logiques (Ensemble de données, ...). Une bonne partie de la description s'inspire de du guide utilisateur de ICCP [24]. Quand les informations ne nous ont pas semblé assez précises, nous avons complété en utilisant la norme décrivant les services TASE.2 [55], et parfois celle sur les objets [54]. [24] et [55] ne sont pas toujours cohérents pour les noms d'attributs.

Notre description est partielle, et ne concerne que les aspects associés aux blocs 1, 2 et 5[6] requis par les applications d'Informatique Industrielle d'EDF.

Dans le cadre des blocs fonctionnels, des informations complémentaires sont apportées par rapport aux objets.

3.5.1 Contrôle d'Accès

Le contrôle d'accès sur un objet dans TASE.2 ce fait via la définition de sa portée. Cette notion de portée est héritée de MMS, qui définit une portée VMD et une portée Domaine. Pour TASE.2, cette notion se traduit respectivement en portée VCC et en portée Domaine (ou ICC[7]). Les objets de portée VCC sont potentiellement accessibles par tous les clients. Les objets de portée domaine, sont accessibles par un client particulier. Le tableau 3.1, issu de [55] relie les différents objets de TASE.2 et leur portée.

[6]Pour mémoire, retenons pour l'instant que le bloc 1 regroupe les informations à envoyer au Client périodiquement, le bloc 2 regroupe les données à envoyer au Client sur événement, et que le bloc 5 est la commande à distance d'un " actionneur " virtuel du Serveur par le Client. Le bloc 4 (non pris en compte ici) correspond à l'envoi de message en ASCII, et le bloc 8 (non pris en compte ici) à l'envoi de programmes exécutables associés à le production d'énergie, directement associés aux invocations de programmes MMS.

[7]**I**nter- **C**ontrol **C**enter

Type d'Objet	**Portée VCC**	**Portée Domaine (ICC)**
Data Value	✓	✓
Data Set	✓	✓
Account	✗	✓
Information Message	✗	✓
Transfer Set	✗	✓
Device	✓	✓
Program	✓	✓
Special Transfer objects	✗	✓
Event Enrollment	✗	✓
Event Condition	✓	✗

✓ Oui **✗ Non**

TAB. 3.1: Portée des Objets TASE.2

3.5.1.1 VCC

L'abstraction VCC définit une centrale virtuelle ou un centre de contrôle virtuel. Elle englobe un ensemble de ressources dédiées à la production d'énergie ou de gestion de production d'énergie, à la façon du VMD de la messagerie industrielle MMS.

Dans le contexte du standard TASE.2, un processus d'application peut être à la fois client et serveur TASE.2. Un VCC existe dans un processus d'application jouant un rôle de serveur. Un serveur TASE.2 peut contenir un ou plusieurs VCC (§ 6.2 de la norme [55]).

Le contrôle de l'accès aux ressources modélisées par un VCC s'exerce à plusieurs endroits dans TASE.2 : au moment de l'établissement de l'association, par le concept de domaine qui est étroitement associé à un client, désigné aussi par ICC et par des droits d'accès associés à chaque variable et aux services de ces variables. Il n'est pas fait mention de fonctions de sécurité comme l'authentification par exemple, de ce fait, les échanges peuvent être sujets à des attaques de type "déguisement", "re-jeu" évoquées classiquement dans les échanges sur l'Internet... Toutefois, la norme spécifie clairement que le Contrôle d'Accès est du ressort de l'implanteur, et qu'il doit veiller à préserver l'interopérabilité entre souches TASE.2. Nous reviendrons sur le sens du terme "contrôle d'accès" tel qu'il est utilisé dans la norme TASE.2 dans les prochains paragraphes.

Remarque : La prise en compte de la sécurité des échanges dans TASE.2 est un enjeu important. En effet, l'échange de données relatives au contrôle de production d'énergie peut revêtir un caractère stratégique aussi bien du point de

vue technique (existence d'ordres d'urgence) qu'économique (données sur les niveaux de production et l'état à venir de la production de telle ou telle centrale). Il semble donc indispensable de prévoir des fonctions de sécurité complètes, en plus de l'authentification déjà manquante, on peut envisager : intégrité, confidentialité, et non-répudiation. On pourra, pour ce faire, s'appuyer sur des technologies existantes (e.g. **SSL**, **PSEC**, **PKI** ...).

3.5.1.2 Association

Une association est établie entre un client et un serveur TASE.2. Les rôles d'un équipement TASE.2 peuvent être : client uniquement, serveur uniquement, client et serveur simultanément. L'établissement est toujours à l'initiative du client. La norme [55] spécifie clairement qu'un serveur uniquement ne peut établir une association si l'équipement TASE.2 est à la fois serveur et client, l'initiateur de l'association doit être désigné d'avance entre les deux utilisateurs. Les implantations de TASE.2 doivent donc être capables de supporter ces différentes configurations. En fonction des rôles, il peut être établi éventuellement plusieurs associations. Le nombre d'associations entre un client et un serveur dépend des capacités du serveur.

Une association TASE.2 repose sur le modèle d'association de MMS. L'objet association contient les attributs suivants :

- Identificateur d'association.
- Identité du client (au sens de l'identification couche 7 OSI : Application Process et Application Entity)
- Paramètres de Qualité de Service du niveau réseau si utilisés
- Blocs Fonctionnels TASE.2 supportés (1... 9)

Les services relatifs aux associations sont :

- opération : *Associate* (mise en relation),
- opération ou action : *Conclude* (fermeture d'association négociée),
- opération ou action : *Abort* (rupture d'association brutale), dans ce cas, le client doit veiller à rétablir l'association.

Lors de l'établissement d'une association, des paramètres de Qualité de Service (QoS) peuvent être négociés pour le niveau réseau. Encore faut-il que le niveau réseau implante une gestion de Qualité de Service. La gestion de QoS est plutôt relative à l'utilisation de la pile OSI en couche réseau, il n'existe pas encore de paramètres de QoS avec la famille des protocoles IP dans le cas où la RFC 1006/2126 serait utilisée. A l'avenir, avec l'avènement des modèles **IntServ** [20] ou **DiffServ** [19], cela pourrait être envisagé).

Un premier contrôle d'accès s'exerce, s'il est supporté, à l'établissement de l'association. Il repose essentiellement sur l'identification de l'entité client, par

ses paramètres d'identité couche 7. Au moment de l'Associate, le serveur effectue la vérification d'identité avant d'établir la connexion.

3.5.1.3 Table Bilatérale et Domaine

Le partage et l'accès à des informations entre deux centres en interaction sont gérés au niveau institutionnel. L'accord institutionnel et bilatéral est dénommé accord bilatéral (**Bilateral Agreement**). C'est un document formel, orienté production d'énergie, qui traduit des besoins fonctionnels qui vont se transcrire en ressources TASE.2.

L'accord bilatéral spécifie pour un centre de contrôle distant le nom, le type, les informations, les droits d'accès associés sur les ressources d'un serveur. La représentation réelle par l'ordinateur de cet accord institutionnel dans le contexte TASE.2 est appelée table bilatérale (**Bilateral Table**). Le centre de contrôle local (serveur) doit posséder une table bilatérale pour chaque centre de contrôle distant (client) avec lesquels il peut être mis en relation. Il est donc de la responsabilité du serveur d'assurer le contrôle d'accès au niveau TASE.2.

D'après la description donnée dans la norme [55] (p.12), la table bilatérale est un objet global qui représente les ressources associées à un VCC. En fait, elle ne représente qu'une vue partielle de ces ressources, celles fournies à un client identifié. La projection des domaines TASE.2 sur MMS et les attributs d'une table bilatérale, dans ce même document apportent quelques éclaircissements et relient les notions de domaines et de table bilatérale. Il est bien dit que dans un VCC, il existe un et un seul domaine par client (il faut remarquer que les termes clients et ICC sont synonymes). Il y a une unique table bilatérale par domaine. Par conséquent, il faut bien distinguer deux niveaux de prise en compte d'un objet réel : son instance physique (unique par essence), et sa modélisation dans l'univers TASE.2. Il peut être modélisé par plusieurs objets TASE.2, avec des identificateurs TASE.2 différents dans des domaines différents en fonction des contraintes de visibilité opérationnelles vis à vis des différents clients.

Une table bilatérale contient, en résumé l'identificateur du client auquel elle est associée et la liste des objets auxquels le client a le droit d'accéder ainsi que les droits d'accès autorisés sur ces objets (lecture, écriture, ou bien les deux).

Il n'y a pas d'opérations ou d'actions associées à une table bilatérale. Ceci signifie en particulier qu'une mise-à-jour n'est possible que localement par le propriétaire du serveur. Et donc que la création de nouveaux clients n'est pas dynamique.

3.5.2 Gestion des variables

TASE.2 définit des objets "Données" et des objets "Ensemble de Données" (supportés via les variables nommées et listes de variables nommées de MMS) gérés par le serveur. Les fonctions de gestion des données et des ensembles de

données permettent de faire des recherches, des créations, des destructions. En pratique, les données dénotent des informations telles que des *Indication Point* (informations d'états, valeurs analogiques, attributs ...)

Les objets "Ensemble de Transfert d'Ensemble de Données" décrivent la façon dont les rapports (objets contenant les valeurs des données) vont être envoyées du serveur vers le client (envoi périodique, sur changement de valeur ...) Les paramètres et leur signification sont donnés dans le tableau 3.2.

Attributs			**Signification**
EventCodeRequested			Application event specification
DS Transmission Pars	**Interval** ❶		Intervalle de temps entre deux rapports
	Start Time ❶		Date à laquelle commencer les rapports périodiques
	RBE [8] ❷		N'inclure que les valeurs modifiées dans les rapports (TRUE) ou toutes (FALSE)
	Integrity Check ②		Intervalle pour l'envoi de rapport d'intégrité(valable si Integrity Time Out est TRUE).
	TLE [9] ❷		Échéance pour la génération du rapport
	Buffer Time ❷		Intervalle de tamponnage des objets modifié (Object Change) avant envoi de rapport.
	Critical ❷		Le client doit accuser réception des rapports
	DS Condition Requested (Spécification du type de rapport voulu)	**Interval Time Out** ❶	Demande d'envoi périodique ou non
		Operator Request	Indique si le serveur doit envoyer un rapport quand un opérateur le demande.

[8] **R**eport **B**y **E**xception
[9] **T**ime **L**imit for **E**xecution

suite ...			
Attributs			**Signification**
		Object Change ❷	Indique si le serveur doit envoyer un rapport sur changement de valeur d'un objet.
		Integrity Time Out ❷	Indique si le serveur doit envoyer un rapport de toutes les données quand Integrity Check expire.
		Other External Event	Indique si le serveur doit envoyer un rapport sur un événement particulier.

❶ bloc 1, RBE est false **❷ bloc 2, RBE est true**

TAB. 3.2: Attributs d'un Ensemble Transfert d'Ensemble de Données

La figure 3.1 donne un aperçu de la façon dont sont gérées les données dans TASE.2.

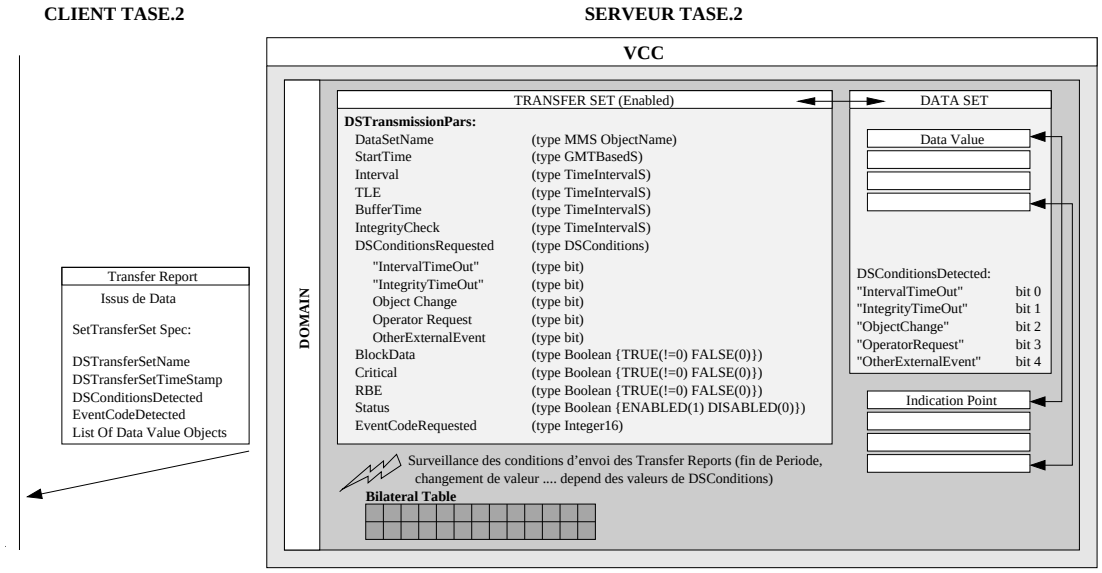

FIG. 3.1: Gestion des variables

3.5.3 Gestion des Équipements dans TASE.2

L'objet Equipement est l'abstraction utilisée pour représenter un équipement physique qui peut être contrôlé par un client TASE.2. La façon dont est contrôlé l'équipement du côté du serveur est en dehors du champ d'application du standard TASE.2. Deux types d'équipements sont définis dans TASE.2 suivant qu'ils doivent être accédés de manière exclusive ou non.

3.5.3.1 Droits d'Accès

La gestion des droits d'accès est dépendante de l'implantation. Il faut alors spécifier donnée par donnée les droits d'accès et les services accessibles.

3.5.3.2 Projections de ces Abstractions sur les abstractions MMS

Les abstractions TASE.2 ci-dessus reposent sur des abstractions MMS par la nature même de la construction de TASE.2. La connaissance des correspondances entre abstractions apporte une aide à la compréhension de TASE.2.

Les projections sont les suivantes, mais des informations plus précises peuvent être trouvées dans la norme [55] (p.48-...) :

- Le VCC est projeté sur le VMD de MMS.
- Un Domaine TASE.2 correspond à un Domaine MMS, un Domaine est associé à un et un seul client (ICC).
- Une Association TASE.2 correspond à une Association MMS.
- Une Variable TASE.2 est représentée par une Variable Nommée MMS.
- Un Ensemble de Variables TASE.2 est représenté par une Liste de Variables Nommées MMS.
- Un Descripteur de compte-rendu (Transfer Set) est représenté par une Variable Nommée MMS.
- Un Transfer Report, qui n'est pas à proprement parler une abstraction, mais plutôt un service, utilise le service non sollicité Information Report de MMS.
- Une Table Bilatérale est un objet composite dont les attributs non relatifs à une fonction de désignation sont représentés par des Variables Nommées et des Listes de Variables Nommées.

Les Variables Nommées, et les Listes de Variables Nommées de MMS sont donc particulièrement utilisées dans TASE.2.

3.5.4 Device

L'objet **Device** représente un équipement virtuel contrôlable à distance. En termes d'implantation, l'objet équipement TASE.2 peut représenter un équipement réel à commander à distance, mais la manière de transmettre l'ordre reçu par l'équipement virtuel à l'équipement réel (et la gestion en général de cet équipement réel en termes de sélection d'équipement, de consignation d'équipement...) sort du cadre de la norme TASE.2. Deux types d'objets Device sont disponibles :

① les objets sensibles à l'inter-blocage (susceptibles d'être accédés en concurrence mais dont l'accès doit être sérialisé),

② les objets sans inter-blocage (susceptibles d'être accédés en concurrence mais pour lesquels il n'y a pas de problème lors d'un accès simultané).

Les premiers doivent faire l'objet d'une sélection avant toute opération.
Les opérations possibles pour ces objets sont :

➡ Sélection, *Select*, (pour un objet à inter-blocage seulement), si la sélection réussit, son état est changé de libre (*IDLE*) à occupé (*ARMED*).

➡ Exécution d'une commande, *Operate*, pour un objet à inter-blocage son état doit être préalablement *ARMED*.

➡ Modification de la marque portée sur l'objet Device, *Set Tag*, la marque indique si l'équipement est accessible ou non par un client, attention ; la notion d'accessibilité est différente de la notion de sélection, elle dénote l'opérabilité de l'équipement (l'équipement peut être entièrement ou partiellement consigné, même vis-à-vis de clients autorisés à lui envoyer des consignes).

➡ Obtention de la valeur de la marque, *Get Tag*.

Quatre actions sont possibles pour ces objets (cf.le guide utilisateur [24]), mais le bloc de conformité 5 n'en retient qu'une seule dans la norme [55] : Réinitialisation locale d'un équipement réel sur le serveur, *Local Reset*, l'objet Device passe de l'état *ARMED* à *IDLE*.

3.5.5 Autres Objets

L'objet *Program* est équivalent à l'objet Programs Invocations de MMS, et correspond à la notion de tâche des systèmes d'exploitation. Les objets *Event Enrollment* et *Event Conditions* permettent la spécification des conditions d'occurrence d'événements en relation avec les erreurs, changements d'état... Les objets de type *Account Object* représentent des informations de comptabilité et de planification relatives à un fournisseur d'énergie (la notion de fournisseur est plus large que la notion de producteur, en effet elle englobe les intermédiaires). Les messages (en format texte ou fichiers) sont représentés par l'objet *Information Message*.

3.6 Blocs de Conformité

3.6.1 Généralités

Les fonctionnalités de TASE.2 sont séparées et regroupées en blocs de conformité (Conformance blocks). La norme [55] définit neuf blocs de conformité , chaque bloc offrant de nouvelles fonctionnalités qui s'ajoutent à celles des autres blocs. Cette modularité permet au protocole TASE.2 d'avoir une souplesse dans le choix de la mise en œuvre, le fournisseur étant libre d'implanter n'importe quel sous-ensemble de blocs. Seul le premier bloc est obligatoire.

① **Le bloc 1** correspond aux services de base, il permet le transfert d'informations de manière périodique. Il fonctionne en utilisant les objets : *Association*, *Data Value*, *Data Set*, et *Data Set Transfer Set*.

② **Le bloc 2** permet le transfert d'information sur changement d'état. Ces services sont les mêmes que ceux offerts par le bloc 1 excepté pour le service *Condition Monitoring* qui s'étend en ajoutant des contrôles périodiques d'intégrité et l'utilisation de l'attribut *ObjectChange* (envoi d'un *Transfer Report* quand un objet du *Data Set* est modifié).

③ **Le bloc 3** ajoute le transfert des objets *Data Set* en utilisant la condition de blocage, *Block Data*, d'un *Transfer Report*.

④ **Le bloc 4** s'occupe de la transmission des objets *Information Message*.

⑤ **Le bloc 5** fournit un mécanisme pour commander un équipement (*Device*) virtuel à distance.

⑥ **Le bloc 6** concerne l'utilisation des objets *Program*.

⑦ **Le bloc 7** est relatif aux événements.

⑧ **Le bloc 8** permet la gestion de tous les objets *Account*.

⑨ **Le bloc 9** permet d'échanger des données échantillonnées.

Dans le cadre des projets de Contrôle-Commande EDF, les blocs fonctionnels 1, 2, et 5 sont nécessaires. Les paragraphes ci-dessous présentent les caractéristiques des blocs fonctionnels TASE.2 supportés par le produit que nous avons porté, les blocs 1, 2 et 5.

On remarque dans la norme que les blocs 1 et 2 utilisent les mêmes services mais qu'ils diffèrent sur l'utilisation de l'attribut *DSConditionsRequested* d'un objet *Transfer Set*.

Dans les produits que nous avons utilisés, il n'est jamais possible de spécifier en un quelconque endroit les blocs de conformité dont on veut se servir. Par conséquent, ce sont les logiciels qui déduisent à travers les attributs configurés définissant l'occurrence de comptes-rendus quels sont les blocs à mettre en oeuvre.

Remarque : Le découpage en blocs fonctionnels peut être vu comme l'ébauche d'une approche composants.

3.6.2 Gestion des Données : Blocs 1 et 2

3.6.2.1 Bloc 1

Dans le guide utilisateur [24], ce bloc est associé aux données produites périodiquement. Le Bloc 1 correspond aux services de base, il est donc implanté obligatoirement dans tout produit conforme à la norme TASE.2 [55]. Les Objets et les Services associés implantés dans le bloc 1 sont donnés dans le tableau 3.3.

Ce tableau indique, d'après le PICS [10] et l'annexe A de la norme TASE.2 [55] :

- ➡ Le statut du service vis à vis de l'implantation Client ou Serveur : obligatoire (●) ou optionnel (❍).
- ➡ La colonne type indique la nature du service du point de vue de la sémantique client/serveur de TASE.2 : opération (oper), quand le service est supporté par un service MMS de type confirmé (sollicitation à l'initiative du client), action (act), quand le service TASE.2 est supporté par un service MMS de type non confirmé (sollicitation à l'initiative du serveur).

Les objets *Data Value* peuvent être de portée VCC ou de portée Domaine, toutefois la mise en œuvre de la portée VCC est optionnelle. Pour les objets *Transfer Set*, le bloc 1 gère les attributs dans le tableau 3.4.

L'indicateur RBE est déterminant puisqu'il conditionne le comportement de la souche TASE.2 en rôle serveur. RBE n'est pas mentionné dans le bloc 1, on peut supposer pour des raisons d'implantation qu'il existe mais vaut faux pour le bloc 1.

Le bloc 1 introduit le transfert d'objets de type Points de Mesure, (Indication Point). Ils se déclinent sous trois différents types :

① Information d'état (Status Point) : *Data_State*.

② Mesure numérique : *Data_Discrete*.

③ Mesure analogique (Analog Point) : *Data_Real*.

La qualité de l'information est estimée à travers un attribut de qualité, *Quality Code* (Data_Flags), qui traduit des informations de pertinence, de validité. L'attribut de qualité peut aussi indiquer comment la valeur a été obtenue : par télémesure, par calcul, par saisie manuelle, estimée. Des informations de fraîcheur peuvent être fournies avec une estampille temporelle, Time Stamp (Data_TimeStamp). Le nombre de changements de valeur peut être obtenu par un attribut compteur, Change of Value (COV) Counter (COV_Counter) s'il est présent.

On peut combiner ces attributs pour former des objets complexes, on obtient alors le tableau de combinaisons 3.5. Le principe de la combinaison est qu'on ajoute des attributs aux types de base.

[10] **P**rotocol **I**mplementation **C**onformance **S**tatement

Objet	Type	Services TASE.2	Serveur	Client
Association	oper	Initiate	●	●
	oper	Conclude	●	●
	oper	Abort	●	●
VCC & Table Bilatérale		TASE.2 Version	●	●
		Supported Features	●	●
		Bilateral Table	●	●
		Bilateral Table Version	●	●
		Access Control	❍	❍
Data Value	oper	Get Data Value	●	❍
	oper	Set Data Value	●	❍
	oper	Get Data Value Names	●	❍
	oper	Get Data Value Type	●	❍
Data Set	oper	Create Data Set	❍	❍
	oper	Delete Data Set	❍	❍
	oper	Get Data Set Element Values	●	❍
	oper	Set Data Set Element Values	●	❍
	oper	Get Data Set Names	●	❍
	oper	Get Data Set Element Names	●	❍
Data Set Transfer Set	oper	Start Transfer	●	●
	oper	Stop Transfer	●	●
	act	Condition Monitoring [a]	●	●
	act	Transfer Report	●	●
Next DSTransfert Set	oper	Get Next DSTransfert Set Value	●	●

● **Obligatoire** ❍ **Optionnel**

TAB. 3.3: Services liés au Bloc 1

[a] Dans le cadre du Bloc 1, ce service porte sur les conditions associées aux paramètres *IntervalTimeOut* et *OperatorRequest* et *OtherExternalEvent* d'après la suite, il utilise le service *Transfer Report*.

Attributs			Serveur	Client
EventCodeRequested			●	❍
DS Transmission Parameters	StartTime		●	●
	Interval		●	●
	DS Condition Requested	Interval TimeOut	●	❍
		Operator Request	●	❍
		Other External Event	●	❍

● **Obligatoire** ❍ **Optionnel**

TAB. 3.4: Attributs du Data Set Transfer Set pour le bloc 1

Types simples	+ **Code Qualité**	+ **Estampille Date**	+ **Compteur Chgts Valeur**
Data_State	Data_StateQ	Data_StateQTimeTag	Data_StateExtended
Data_Discrete	Data_DiscreteQ	Data_DiscreteQTimeTag	Data_DiscreteExtended
Data_Real	Data_RealQ	Data_RealQTimeTag	Data_RealExtended

TAB. 3.5: Types Indication Point

Une définition formelle des types Indication Point est donnée dans la norme sur les bojets TASE.2 [54]. Elle est rappelée dans la table 3.2.

Il nous semble qu'il faille distinguer les attributs propres des variables, et les attributs gérés pour les comptes-rendus par TASE.2. Par exemple, les sources normale et courante de la variable (TELEMETERED, CALCULATED, ENTERED, ESTIMATED), la date de production de la valeur, la validité (VALID, HELD, SUSPECT, NOTVALID), l'appartenance au domaine de définition prévu (NORMAL, ABNORMAL) sont des informations qui ne peuvent provenir que du procédé. Par contre la comptabilisation du nombre de changements de valeur depuis le dernier envoi ne peut être géré que par la souche TASE.2 via des fonctions de mise en correspondance avec le calculateur de supervision ou de régulation de production !

3.6.2.2 Bloc 2

Le bloc 2 est dénommé "**Extended Data Set Condition Monitoring**". Alors que le bloc 1 permet de transférer des informations périodiquement, ce qui peut ne pas être adapté à certaines situations car trop consommateur de bande passante. Le bloc 2 permet de transférer des informations sur changement d'état ou changement de valeur. A cette fin, le serveur surveille un certain nombre de conditions supplémentaires initialisées par le client. Quand il y a un changement, la nouvelle valeur est envoyée au client. Ces conditions supplémentaires peuvent être :

```
Object: IndicationPoint (Read Only)
  Key Attribute: PointName
  Attribute: PointType (REAL, STATE, DISCRETE)
  Constraint PointType=REAL
    Attribute: PointRealValue
  Constraint PointType=STATE
    Attribute:PointStateValue
  Constraint PointType=DISCRETE
    Attribute: PointDiscreteValue
  Attribute: QualityClass: (QUALITY, NOQUALITY)
  Constraint: QualityClass = QUALITY
    Attribute: Validity (VALID, HELD, SUSPECT, NOTVALID)
    Attribute: CurrentSource (TELEMETERED, CALCULATED, ENTERED, ESTIMATED)
    Attribute: NormalSource (TELEMETERED, CALCULATED, ENTERED, ESTIMATED)
    Attribute: NormalValue (NORMAL,ABNORMAL)
  Attribute: TimeStampClass: (TIMESTAMP, NOTIMESTAMP)
  Constraint: TimeStampClass = TIMESTAMP
    Attribute: TimeStamp
    Attribute: TimeStampQuality: (VALID, INVALID)
  Attribute: COVClass: (COV, NOCOV)
  Constraint: COVClass = COV
    Attribute: COVCounter
```

FIG. 3.2: Définition formelle des types Indication Point

Attributs			Serveur	Client
EventCodeRequested			●	❍
DS Transmission Parameters	RBE		●	❍
	Integrity Check		●	❍
	TLE		●	❍
	Buffer Time (lié à ObjectChange)		●	❍
	Critical		●	❍
	Start Time		●	●
	Interval (lié à IntervalTimeOut)		●	●
	DS Condition Requested	ObjectChange	●	❍
		IntegrityTimeOut	●	❍
		IntervalTimeOut	●	❍
		OperatorRequest	●	❍
		OtherExternalEvent	●	❍

● **Obligatoire** ❍ **Optionnel**

TAB. 3.6: Attributs du DS Transfer Set pour le bloc 2

- ➡ Report-By-Exception (*RBE*) si vrai implique un comportement événementiel.
- ➡ Fin de période (*IntervalTimeOut*,identique bloc 1).
- ➡ Vérification périodique de la qualité des informations entre le serveur et le client (*IntegrityTimeOut*).
- ➡ La valeur, l'état, le code qualité associé à une valeur ont changé (*ObjectChange*).
- ➡ L'opérateur côté serveur a demandé que la valeur soit envoyée au client (*OperatorRequest*,identique bloc 1).
- ➡ Conditions autres (*OtherExternalEvents*).

Les services offerts par ce bloc sont les mêmes que ceux offerts par le bloc 1 sauf pour la partie *Condition Monitoring* qui est enrichie. Les paramètres **supplémentaires** sont décrits dans le tableau 3.6.

La corrélation des attributs et indicateurs est synthétisée dans le tableau 3.7

Attributs		Signification
EventCodeRequested		Définition d'événement applicatif spécifique
	Interval ❶	Période
	Start Time ❶	Première activation de la période

suite ...

Attributs			Signification
DS Transmission Pars	**RBE, Report By Exception ❷**		Booléen, faux : tous les objets envoyés, vrai : uniquement les objets qui ont changé sont envoyés
	Integrity Check ❷		Date de péremption d'une valeur
	TLE, Time Limit for Execution ❷		Échéance pour la transmission d'un compte-rendu
	Buffer Time ❷		Bande morte pour la mémorisation de changements de valeurs, seule la dernière valeur obtenue est prise en compte, si 0 toutes les valeurs sont prises en compte.
	Critical ❷		Le Client doit acquitter le compte-rendu
	DS Condition Requested (indicateurs autorisant la génération de comptes-rendus)	**Interval Time Out ❶**	Active l'envoi de comptes-rendus contenant toutes les données d'un Data Set Transfer Set en fin de période, corrélé à Interval et RBE faux
		Interval Time Out ❷	Active l'envoi de comptes-rendus contenant uniquement les données qui ont changé d'un Data Set Transfer Set en fin de période, corrélé à Interval et RBE vrai
		Operator Request ❶	Active l'envoi de comptes-rendus sur demande opérateur
		Object Change ❷	Active l'envoi de comptes-rendus sur changement de valeur, corrélé à Buffer Time et RBE vrai
		Integrity Time Out ❷	Active l'envoi de comptes-rendus sur péremption de valeur, corrélé à Integrity Check et RBE vrai

suite ...			
Attributs			**Signification**
		Other External Event ❷	Active l'envoi de comptes-rendus sur évènements applicatifs

❶ Attribut du bloc 1 **❷ Attribut ajouté au Bloc 2**

TAB. 3.7: Corrélation entre attributs et indicateurs

Nous avons observé que certaines implantations effectuent des vérifications sur le positionnement de certains attributs. Par exemple, dans AXS4-ICCP [11], RBE à vrai sans aucun autre attribut n'est pas autorisé car aucun indicateur de détection de condition de génération de compte-rendu n'est positionné. Une étude exhaustive des produits du marché devrait indiquer si ce comportement du logiciel est d'usage ou non. Nous ne pouvons faire autrement en l'absence de tests de conformité normalisés.

3.6.3 Bloc 5

Alors que les blocs 1 et 2 servent à partager et à maintenir des informations issues de mesures, de calculs ou d'entrées opérateur, le bloc 5 concerne le contrôle d'équipements virtuels distants sur un serveur TASE.2. Le principe de ce contrôle distant dépend des opérations à effectuer : en exclusion mutuelle (Select Before Operate) ou non (Direct Control). Comme pour MMS, TASE.2 ne contrôle pas directement l'équipement mais fournit les moyens de le faire, une partie opérative (sous la forme d'une bibliothèque de pilotes d'accès à l'équipement réel) doit être associée à la messagerie TASE.2 (sa description sort du cadre de la norme).

Les services spécifiques du bloc 5 sont décrits dans le tableau 3.8. La notion de "Tag" correspond au marquage d'un élément (consignation) afin de l'exclure d'un service ou de le rendre inaccessible (cf. le guide utilisateur [24]).

Les objets du bloc 5 tentent de refléter les caractéristiques les plus importantes des équipements temps réel. Les objets de contrôle (*ControlPoint)* représentent des objets de l'équipement réel. Ils distinguent les commandes, (device operation, *Command*) et les valeurs modifiables (*Setpoint)*, qui peuvent être entières (*SetpointDiscreteValue)* ou réelles (*SetpointRealValue*). D'après la norme [54], dans le cadre d'applications SCADA [12], les objets device de type *Command* servent à modéliser en TASE.2 des interrupteurs (switch) tandis que ceux de type Setpoint servent pour des transformateurs d'énergie (transformer power units). Dans le

[11] produit de SISCO.
[12] **S**upervisory **C**ontrol **A**nd **D**ata **A**cquisition

Objet	Type	Services TASE.2	Serveur	Client
Devices	operation	Select	●	●
	operation	Operate	●	●
	operation	Get Tag	❍	❍
	operation	Set Tag	❍	❍
	action	Local Reset	❍	❍

● **Obligatoire** ❍ **Optionnel**

TAB. 3.8: Services spécifiques du bloc 5

cadre du bloc 5, les objets de type *ControlPoint* sont modifiés par l'opération *Operate*. Suivant le type de l'objet *ControlPoint*, une opération de sélection, *Select*, peut être nécessaire.

Une requête du bloc 5 peut être sujette à inter-blocage ou non (*interlocked* ou *noninterlocked*). Une requête sujette à inter-blocage doit préalablement faire l'objet d'une sélection de l'équipement, *Select* avant toute autre chose (attribut *select-before-operate*, *SBO*). Ce service correspond à un mécanisme de transaction temps réel verrouillant, semble-t-il, tout l'équipement.

Lors d'une requête de sélection, le serveur effectue les vérifications d'usage (existence du "device", droits d'accès dans la table bilatérale, disponibilité de l'équipement pour l'opération distante via la consignation associée décrite plus loin). Un attribut *CheckBackName* sur lequel se sont mis d'accord le client et le serveur (grâce à la table bilatérale) permet d'identifier le device . Une temporisation est associée à la durée de sélection d'un équipement, ainsi il existe une borne de temps maximale à l'utilisation d'une commande sujette à inter-blocage. Cette temporisation est gérée par une action du serveur, qui déselectionnera l'équipement et enverra une notification au client qui avait effectué la sélection (ceci évite par exemple qu'un équipement sélectionné par un client qui tombe en panne ne reste inaccessible).

Les consignations, tags, se rapportent aux objets de contrôle. Les consignations traduisent l'opérabilité de ces objets de contrôle. Le client peut obtenir 3 valeurs possibles pour une consignation :

① **NO-TAG** : Pas consigné, n'empêche pas l'accès à l'équipement réel.

② **OPEN-AND-CLOSE-INHIBIT** : Consigné "inhibé pour ouverture et fermeture", cette valeur traduit l'indisponibilité totale de l'équipement.

③ **CLOSE-ONLY-INHIBIT** : Consigné "inhibé pour fermeture seulement", cette valeur restreint les opérations possibles[13]

[13][24] est en contradiction avec la norme sur les objets TASE.2 [54] où les valeurs de marques 1 et 2 sont inversées.

Après avoir sélectionné un équipement, et vérifié son accessibilité, un client peut effectuer une opération ou l'annuler. Le serveur envoie alors une réponse positive au client si la requête a été acceptée, une réponse négative si elle n'a pas été reçue ou pas acceptée. Ce sont seulement des actions (initiées par le serveur donc) qui permettent de signifier au client, du point de vue de la sémantique applicative, l'échec (Failure) ou le succès (Success) de sa commande, et ceci n'est supporté qu'en bloc 7.

3.7 Conclusion

Le protocole TASE.2 et les services qu'il fournit (en particulier ceux correspondant aux blocs 1,2 et 5) sont intéressants. Ils permettent l'échange de variables en prenant en compte des aspects temporels. Une plateforme TASE.2 offre donc les mêmes fonctionnalités qu'une architecture de supervision. La différence fonctionnelle restant que celle-ci offre des communications de un vers N (multicast) alors que TASE.2 ne propose que des relations un vers un.

Mais c'est peu comparé à toutes les abstractions offertes par TASE.2 et qui sont directement utilisables dans une architecture de médiation. Les propriétés temporelles sont également suffisantes pour répondre aux besoins à la fois dans la centrale et entre les centres de régulation.

Nous avons donc trouvé ici un bon candidat pour la définition fonctionnelle de notre messagerie industrielle avec des propriétés temporelles. Il reste à approfondir cette connaissance et à la comparer aux outils existant qui se disent conformes au standard TASE.2, pour voir si les fonctionnalités sur le terrain sont les mêmes que sur le papier.

De plus, il est intéressant de noter que les abstractions et services proposés par la spécification et décrits dans ce chapitre nous apporte un début d'organisation objet que l'on peut mettre à profit.

Chapitre 4

Étude de conformité d'un serveur TASE.2

Sommaire

4.1 Introduction

La connaissance du protocole TASE.2, qui occupe une place centrale dans ce travail de thèse, a été acquise au travers d'un travail de portage d'une souche serveur s'exécutant sous Windows NT vers un système d'exploitation temps réel, pSOS+. Le choix de pSOS+ résulte d'une étude plus globale sur les architectures distribuées temps réel pour le contrôle-commande des centrales EDF ([84] [85] [79] [82] [78] [81] [86] [80] [74] [90] [91] [75] [89] [76]).

Ce travail s'est révélé être autant un problème de portage que de test de conformité, et nous a donc poussé à nous investir dans une étude très approfondie de la norme TASE.2 en elle-même, nous permettant d'obtenir des connaissances qui nous ont été utiles par la suite.

Ce travail a été effectué dans le cadre d'un contrat passé avec EDF R&D et pour des raisons de propriété intellectuelle, nous ne donnerons pas la nom du produit évalué (souche serveur TASE.2). Pour de plus amples informations, il est requis de prendre contact avec EDF R&D directement.

4.2 Caractéristiques générales du serveur TASE.2

La souche TASE.2 que nous nous proposons de porter n'implante que la partie serveur de TASE.2 ; il existe également une souche du même logiciel qui est capable de jouer les fonctions Serveur et Client, mais elle ne fait pas partie du contexte d'expérimentation de la présente étude. Le logiciel est fourni avec un client léger MMS s'exécutant sous Windows NT et permettant de réaliser des tests de fonctionnement, au niveau MMS et non au niveau TASE.2. Cet outil permet de capturer, lors d'interactions prévues dans le protocole TASE.2, le comportement du serveur face à un client mais du point de vue MMS.

Le serveur TASE.2 utilise la pile de communication TCP/IP, la RFC 1006 [45] et les couches OSI 5 à 7 associées au protocole ISO MMS dans le profil MAP. L'interface avec TCP/IP se fait via les sockets.

Ce serveur est une implantation de ISO-MMS et de TASE.2 (ainsi que partiellement de CASM-GOMSFE [10] [11], autre protocole en cours de normalisation, ayant de forts liens de parenté avec TASE.2 et souvent présenté comme son successeur, mais ce n'est pas l'objet de l'étude), il est spécialement destiné aux systèmes embarqués. Il a été développé en utilisant le langage standard ANSI-C, et a été porté avec succès sur plusieurs processeurs.

Le logiciel est fourni dans sa version code source puisque le but de notre travail est d'effectuer son portage. Les codes sources fournis sont ceux de la version Windows NT et comprennent donc tous les fichiers makefiles, les fichiers de configuration, les scripts et autres fichiers nécessaires à la compilation, l'édition de liens et la création de l'exécutable sous Visual C++.

4.2.1 Organisation des fichiers

Nous avons une version de référence qui s'exécute sous Windows NT, elle inclut des fichiers contenant les directives de compilation (Makefiles). Le code source est réparti en plusieurs répertoires :

- ➡ code indépendant de la plate-forme : initialisation, gestionnaires de données, moniteurs de données pour générer des rapports de transfert (Transfer Report), interface avec les API système normalisées ou standard,
- ➡ code spécifique dépendant de la plate-forme : accès aux services réseau par exemple, API système spécifique.

4.2.2 Versions fournies

Une première version (0.2) nous a été fournie le 18/03/2000 incluant d'autres fichiers liés au projet UCA[1] qui n'étaient nulle part utilisé ni dans notre portage ni dans les tests. Cette version n'implantait que les blocs fonctionnels 1 et 2.

Une deuxième version (0.3) nous a été fournie le 12/05/2000.

Une troisième version, notre version de travail définitive (0.4) est arrivée le 16/06/2000.

Ces évolutions montrent la jeunesse du produit, au moins pour sa partie TASE.2 et expliquent en partie le temps consacré au portage proprement dit (3,5 hommes-mois).

4.2.3 Configuration et Gestion des droits d'accès

Les objets TASE.2 que va exporter le serveur à ses clients, sont décrits dans un fichier de configuration appelé fichier modèle (extension .mdl). Ce fichier est complété par des informations de programmation dans un fichier complémentaire (.end) dont la syntaxe est conforme au langage C. Ces fichiers sont utilisés par un pré-processeur qui génère des fichiers source en langage C (un fichier .c et un .h), pour implanter le dictionnaire de données du serveur et permettre l'accès aux données à travers des gestionnaires appropriés (handlers).

Les attributs des objets peuvent être spécifiés dans le fichier de configuration. Le contrôle d'accès est prévu par la norme TASE.2 mais est laissé libre à l'implanteur pourvu qu'il prenne à sa charge tous les problèmes d'interopérabilité qui pourraient survenir. L'accès à un objet de type READONLY provoque l'échec de toute tentative d'écriture ('writeFailure'). L'accès à un objet de type WRITEONLY provoque l'échec de toute lecture. L'utilisation des attributs readDenied et writeDenied permet de gérer le contrôle d'accès (non implanté ici).

[1] Utility Communication Architecture, projet originaire des États-Unis ayant pour objectif de proposer des solutions pour les communications à tous les niveaux : contrôle-commande, liaison poste-dispatching, liaison inter-dispatchings, et ayant abouti d'une part à TASE.2, d'autre part à CASM/GOMSFE

Une fois les fichiers modèle et complémentaire (.end) définis, il ne reste plus qu'à générer les librairies et l'exécutable en utilisant les menus adéquats.

4.3 Portage

4.3.1 Contexte

Nous rappelons ici que le choix de pSOS+ s'est fait au terme d'une étude complète (cf. annexe A) comparant les systèmes d'exploitation temps réel, tant du point de vue des performances que de l'environnement d'ingénierie.

Les résultats de cette étude sont consignés en annexe B.

4.3.2 Rappels sur pSOS+ et son environnement de développement

4.3.2.1 Architecture générale du noyau pSOS+

Le système d'exploitation pSOS+ est un système modulaire qui comprend le système multitâche en lui-même et des composants logiciels optionnels dont on peut spécifier la présence ou non dans le fichier de configuration système propre à chaque application. On trouve parmi ces composants logiciels : la librairie C standard (**pREPC**), la librairie RPC (**pRPC**), la pile TCP/IP (**pNA**), le module de communication port série, etc ...

Le noyau de base est pSOS+, le noyau multitâche temps réel. Tous les composants cités ci-dessus se présentent sous la forme de modules de code relatif (modules objet) et complètement indépendants du processeur et de l'architecture sous-jacente, tous les renseignements nécessaires étant fournis par l'intermédiaire de tables de configuration à ces composants, tables dont les entrées seront renseignées dans ce même fichier déjà cité ci-dessus. La figure 4.1 présente l'architecture modulaire de pSOS+ et en particulier les composants optionnels et les composants obligatoires (Ordonnanceur par exemple).

4.3.2.2 Environnement de développement

Le cœur de l'environnement de développement ISI pour pSOS+, correspond au programme **pRISM+** qui se présente sous la forme d'une barre de lancement qui permet d'ouvrir un projet et de lancer tous les outils nécessaires à son développement, en renseignant pour chacun les paramètres voulus. Il est ainsi inutile de dire au dévermineur quel est le fichier binaire cible : il aura été passé en paramètre par pRISM+. C'est également à ce niveau qu'on choisit le type de cible et qu'on la configure en terme de communications, à savoir choix du port série ou du réseau, et configuration des paramètres associés au type de communication.

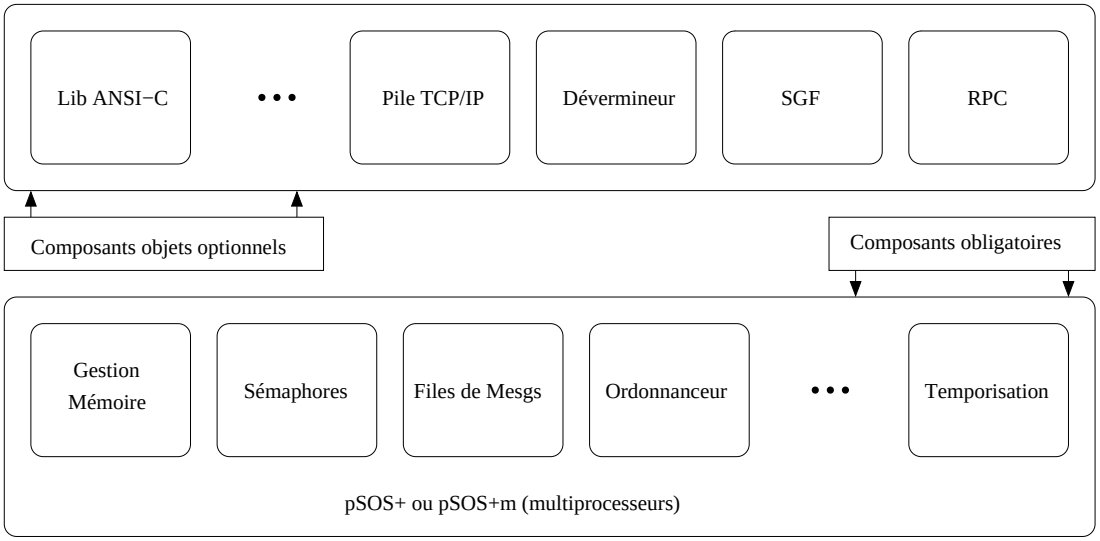

FIG. 4.1: Architecture modulaire de pSOS+

On peut aussi télécharger l'application, booter pSOS+ et lancer l'application à partir de cette barre.

A partir de pRISM+, on peut donc lancer :

- ***SniFF+*** *: outil de gestion de sources.*
- ***pWIZARD*** *: outil de configuration du fichier sys_conf.h.*
- ***SingleStep*** *: dévermineur hôte/cible.*
- ***SearchLight*** *: dévermineur (plus simple que SingleStep).*
- ***ESp*** *: Outil de trace et d'étude de performances (profiling).*
- ***Object Browser*** *: Outil de dimensionnement.*

Tous ces outils communiquent via un bus logiciel à objets CORBA, **orbix**, qui permet la transmission des paramètres du projet entre les différents composants de l'environnement.

Dans un environnement de développement hôte/cible, il est nécessaire de disposer d'un compilateur croisé, i.e. qui peut produire du code pour une machine de type B à partir d'une machine de type A. Sous pSOS+, ce compilateur s'appelle **dcc**. Dans notre cas, l'hôte est une machine sous Windows NT et la cible une carte MBX Motorola.

La procédure pour lancer et mettre au point une application sous pSOS+ se résume donc à la liste des opérations qui suit :

- Développement sur la machine hôte (Windows NT)
- Compilation et génération d'un binaire pour la cible choisie (carte MBX)
- Téléchargement de ce binaire sur la cible via le dévermineur ou pRISM+

- ➡ Début de l'exécution de l'application sur la cible
- ➡ Suivi/Interactions avec l'application via un terminal quelconque
- ➡ Déverminage via SingleStep (dévermineur hôte/cible fourni)
- ➡ Améliorations éventuelles
- ➡ ...

L'arborescence des fichiers est totalement masquée par ces outils, excepté lorsqu'il s'agit de choisir le type de la carte cible c'est-à-dire lorsqu'il s'agit de choisir le BSP. Tout est transparent pour le développeur, on ne détaillera donc pas plus avant la partie organisation des fichiers.

4.3.3 Modifications effectuées sur la souche TASE.2

4.3.3.1 Modifications au niveau des fichiers Makefile

Les makefile n'ont nécessité quasiment aucun changement pour ceux que nous avons conservés. En effet, les makefile permettent de générer une librairie ou l'exécutable. La génération de l'exécutable est un peu particulière puisqu'elle nécessite de lier les fichiers objets relatifs à la machine (le BSP). La solution retenue lors du portage est donc la suivante :

Les fichiers makefile générant les librairies sont récupérés et utilisés tels quels à deux légères modifications près :

Il a fallu modifier les noms de répertoires comprenant des '\' et les remplacer par des '/' pour que les outils pSOS+ fonctionnent. En effet, bien que l'environnement de développement fonctionne dans un environnement Windows NT, il suit beaucoup de règles UNIX.

Il a fallu adapter les règles de type condition sur définition (#ifdef) , non reconnues sous pRISM+, cette syntaxe n'a pas d'équivalence sémantique entre les deux environnements.

Le fichier makefile générant l'exécutable est abandonné. On le remplace par un makefile pSOS+ type dans lequel on rajoute les directives de compilation nécessaires à la compilation des fichiers de l'exécutable et à l'inclusion des librairies externes.

Les deux fichiers compiler.mak et options.mak permettent de modifier le paramétrage et la génération des binaires. Ils sont tous deux inclus en début de chaque makefile et sont communs à toutes les librairies et à l'exécutable.

Le fichier options.mak permet de positionner les différentes définitions pour choisir telle ou telle configuration du serveur (pile OSI, pile TCP, fonctions GOMSFE, environnement embarqué, affichage console ...). Nous n'avons fait aucune modification dans ce fichier puisque malheureusement les options qui nous étaient utiles n'étaient pas dans celui-ci (options déverminage et traces étaient présentes dans les fichiers source).

Le fichier compiler.mak permet de configurer tous les outils de l'environnement et les options de compilation. C'est dans ce fichier que sont indiqués les outils spécifiques à notre portage : le nom du compilateur, les options de compilation, le nom de l'archiveur, le nom de l'éditeur de liens... Notons également que nous avons rajouté la définition de l'opération concaténation qui ne se fait pas de la même manière sous Win32 et sous pSOS+/UNIX.

Nous avons changé l'organisation des makefile qui se trouvaient situés dans le répertoire make7. Nous avons adopté une organisation plus courante sous pSOS+ (et dans le monde UNIX), à savoir un makefile dans chaque répertoire donnant naissance à une librairie ou à l'exécutable. Nous avons donc maintenant des fichiers nommés makefile dans chaque répertoire où celui-ci est nécessaire en lieu et place des fichiers make<lib>.mak qui permettaient de générer la librairie <lib> pour Windows.

4.3.3.2 Modifications de fichiers scripts

Réécriture complète du fichier batch MS-DOS maketab.bat en fichier script SHELL maketab.sh : Ce script sert essentiellement à générer les codes source C à partir des grammaires à l'aide du logiciel TYACC (Ce logiciel permet de générer des fonctions C qui vont décoder des flux suivant une certaine grammaire spécifiée en entrée).

4.3.3.3 Modifications dans les sources

Il a fallu changer la définition des entiers, des entiers non signés et des intervalles de temps (basés sur les entiers non signés) pour prendre en compte une taille des entiers différente sur Intel (2 octets) et sous Motorola (4 octets). Cette modification est due aux processeurs qui sont différents, et non à des systèmes d'exploitation différents. Un portage sous pSOS+ sur un processeur Intel n'aurait pas nécessité ce changement.

Ce changement a entraîné la redéfinition de certains types TASE.2 (*TASE2_ - TimeIntervalD, TASE2_ TimeIntervalH, TASE2_ TimeIntervalM, TASE2_ TimeIntervalS, TASE2_ TimeIntervalL16)* fondés sur les types MMS mais qui redéfinissent eux-mêmes la taille du type ! Ainsi le type TASE.2 *TASE2_ TimeIntervalD* hérite du type MMS *Server_ Integer* mais redéfinit la taille à deux octets. De façon générale cette façon de procédé en programmation est source d'erreurs, difficiles à détecter et particulièrement pénibles à déverminer, nous en avons d'ailleurs été victime.

Il a fallu ajouter des morceaux de code pour pouvoir gérer le temps, ce sont des appels système spécifiques. Nous entrons pour l'instant une date au démarrage du système pour pouvoir faire fonctionner le serveur. En effet, la cible MBX ne dispose pas d'une horloge interne, son origine des temps est donc la dernière réinitialisation.

4.3.3.4 Modifications non liées au portage (erreurs dans le produit source)

Plusieurs bugs sont apparus lors du portage, certains ont été corrigés par la société éditrice de notre serveur (les problèmes de mauvaise génération du fichier source par le préprocesseur par exemple) mais d'autres subsistent.

Un problème majeur provient de la déclaration de l'identificateur de la table bilatérale en tant que caractère au lieu de chaîne de caractères. Les problèmes liés sont particulièrement insidieux puisque du coup, il y a recouvrement des adresses et écrasement potentiel et aléatoire (dépendant de la structure du fichier modèle) des données. Ainsi, l'un de nos serveurs voyait son identificateur de table bilatéral partager son adresse avec le tic horloge, cet identificateur changeait donc toutes les millisecondes ! Une autre fois, l'adresse était en conflit avec les blocs supportés et l'écriture de l'identificateur de table bilatérale redéfinissait ces blocs. Ce problème a été traité mais uniquement au niveau du fichier généré, une mise à jour du préprocesseur est nécessaire.

Un autre problème est lié à la gestion des devices, la fonction de balayage de la structure qui permet de les stocker est incorrecte. Il manque une incrémentation : la boucle ne balaye que la première entrée. La fonction ne s'exécute donc bien que dans le cas ou il n'y a qu'un device et boucle indéfiniment en cas d'ajout d'un deuxième. Ce problème a été corrigé.

4.4 Bilan du Portage

4.4.1 Problèmes liés au code source

Le gros problème rencontré dans ce portage correspond à l'insuffisance de documentation qui nous a obligée à plonger dans le code plus que cela était nécessaire. Par exemple, la présence d'options de trace et de déverminage aurait pu être bien utile dès le début du travail si leur localisation dans les fichiers de configuration avait été correctement documentée.

La manière de gérer les fichiers d'inclusion nous semble sujette à erreurs. En effet, la recopie de certains fichiers entêtes, headers, (utilisés par plusieurs librairies) du répertoire d'origine vers un répertoire de partage est dangereux. Cela signifie qu'il existe à un moment donné plusieurs versions d'un même fichier. Dans notre portage, nous avons placé ces fichiers directement dans le répertoire de partage sans problèmes.

La structuration du fichier de configuration, divisé en deux fichiers distincts (.mdl et .end) paraît dangereuse, il faut aller modifier deux fichiers pour changer une configuration. On peut se demander pour quelle raison tout n'est pas intégré dans un seul fichier.

4.4.2 Durée de l'étude et du portage

Le portage proprement dit correspond à 3 hommes-mois et demi au total. La version source qui nous a été livrée était prévue pour une machine Windows NT. La translation de cet environnement à environnement pSOS+ sur MBX n'a pas été trop difficile, un homme-mois et demi. Cette translation se décompose comme suit. La mise au point a été un peu plus longue, deux hommes-mois y compris le développement d'un module de test spécifique qui sera décrit plus loin. Les erreurs rencontrées étaient plutôt liées à des règles d'écriture du code source (redéfinition de types à plusieurs endroits par exemple) et ont été résolues au début. Par contre, une investigation approfondie du code source a été nécessaire pour valider chaque fonction. Le portage a été ralenti essentiellement à cause de la méconnaissance de la norme TASE.2, sa maîtrise est indispensable pour la compréhension du code. Le fait de ne pas connaître TASE.2 était une condition initiale de l'étude fixée par EDF R&D (en effet, nous avions une expertise de pSOS+ [78] [81] mais pas de TASE.2, ce type d'approche reproduit dans une phase d'étude des conditions qui pourraient être réalisée par une société de service si TASE.2 devait être porté dans d'autres environnements à des fins de déploiement opérationnel). Nous avons par ailleurs du reprendre plusieurs fois le portage car nous avons reçu au total 3 versions de la souche. On peut citer parmi les autres obstacles au portage :

- Une documentation déficiente du logiciel.
- Les jetons pSOS+ à durée de validité courte qu'il a fallu redemander plusieurs fois, y compris juste avant une pré-recette, enfin à peine avant la pré-recette nous avons obtenu un jeton définitif ce qui a assoupli considérablement les conditions de développement et de test.
- La nécessité d'acquérir Windows NT et Visual C++ pour générer une version de référence du serveur.

4.4.2.1 Durée des tests

La spécification et la conception des tests fonctionnels ont duré trois homme-mois. Elles ont été plus longues que prévues. Cette phase se découpe en trois phases de durées approximativement égales :

① La formation aux outils et au protocole

- La formation à l'environnement et à TASE.2 a été longue. Cette partie a nécessité l'apprentissage de différents outils, dont un client TASE.2 extérieur et le client léger MMS fourni avec le serveur (même si finalement celui-ci n'a pas été utilisé lors de la recette).
- Par rapport au logiciel : La formation de quelques jours faite par la société éditrice a permis un débroussaillage de la structure du logiciel, il a fallu finalement un certain temps pour avoir une maîtrise qui permette de vérifier le comportement de la souche après portage.

- ➡ Les outils associés au Client et au Serveur ne sont pas très difficiles à manipuler, mais la validation du portage nécessite une maîtrise fine ce qui est beaucoup plus long. En effet, l'évaluation du portage a été quasiment un test de conformité.

② La conception des tests

- ➡ La rédaction et la mise au point du document de test et de portage. La mise au point est relative à la partie test proprement dite.
- ➡ L'exploration des blocs fonctionnels 1 et 2 de la norme TASE.2, la compréhension du bloc 5 a été plus simple.
- ➡ La spécification et la conception des tests fonctionnels ont été réalisées par une équipe différente de celle du portage afin de ne pas être influencé par la connaissance des internes de la souche portée. Toutefois, nous avons amendé et validé ces tests.

③ La mise en œuvre des tests fonctionnels a été faite par l'équipe portage seule. La préparation des tests de pré-recette et de recette a nécessité un homme-mois et demi. Il faut inclure dans cette partie le développement du moniteur de tests décrit ci-après.

4.4.3 Portage vers d'autres OS Temps Réel

A priori il devrait être aisé de porter ce serveur vers d'autres systèmes d'exploitation temps réel. Ceux que nous connaissons (ChorusOS, VxWorks, LynxOS) ont en effet des fonctionnements comparables à pSOS+ en terme d'environnement. De plus, les problèmes ont déjà été identifiés et une bonne connaissance des sources nous autorise une progression plus rapide.

Notre banc de tests portable développé au moment de notre étude sur les systèmes d'exploitation temps réel (cf. Annexe A) sera également très utile car il comporte une couche d'homogénéisation et d'abstraction qui permet d'identifier les différences entre les systèmes.

4.5 Développement d'un moniteur d'introspection pour les tests

Un moniteur pour le serveur a été développé. Il s'agit d'un ensemble de fonctions d'introspection des différentes structures internes du logiciel, accessibles depuis l'extérieur par un client développé en C. La connexion se fait via TCP/IP avec une socket, les informations échangées sont uniquement du type chaînes de caractères pour éviter des problèmes de typage dues à des plates-formes d'utilisation différentes pour le client et le serveur.

Dans un premier temps cet outil a été réalisé pour aider le déverminage de la souche serveur et faciliter le portage. Il est apparu que cet outil pouvait se révéler extrêmement pratique lors des tests fonctionnels, par exemple, pour vérifier le bon fonctionnement d'un client TASE.2, en changeant la valeur d'une variable sur le moniteur et en vérifiant l'effet de ce changement côté client. Le moniteur est devenu indispensable pour tester la modification des attributs de Transfer Set et la génération de Transfer Report associée. Le moniteur est un outil qui participe à des tests boîte blanche ce qui contredit notre hypothèse initiale de tests en approche boite noire. Mais ceci a été rendu indispensable pour certifier le portage et attester de sa conformité par rapport à la norme TASE.2.

Le client et la partie serveur du moniteur ont été portées avec succès et sont donc utilisables indifféremment sur Win9x/WinNT et pSOS+. Ils devraient aussi fonctionner sans problèmes sur des plate-formes UNIX. Le moniteur a été testé avec succès dans les configurations client/serveur suivantes : Win32/Win32 et Win32/pSOS+.

Le moniteur apporte des fonctions de surveillance (affichage des domaines, des data set, des transfert set, des devices, etc ...) et de modification (écrire des transfer set, des valeurs, des data set, des device, etc...).

4.6 Tests Fonctionnels sur le Prototype

4.6.1 Préalable aux Tests

Les tests requis par EDF R&D sur le résultat du portage de la souche TASE.2 sont des tests fonctionnels ne portant que sur la messagerie industrielle elle-même. Par conséquent, il n'est pas testé d'interactions avec un équipement réel. Tout ce qui concerne l'attribut *EventCode Requested* de l'objet *Data Set Transfer Set*, et l'indicateur *OtherExternalEvent* dans les tests *Data Set Condition Monitoring* pour le Bloc 1 comme pour le Bloc 2 a été de ce fait éliminé.

Telle que l'étude a été définie, la spécification et la mise en oeuvre des tests fonctionnels impliquent des tests boîte noire. Nous nous en tiendrons à cette approche dans la mesure du possible.

4.6.2 Plate-forme de tests

Le portage de la souche serveur TASE.2 sur pSOS+ s'effectue sur une carte MBX à base de processeur PowerPC Motorola. Afin d'évaluer le résultat du portage la plate-forme suivante est proposée :

- Un Hub 10 BASE T Ethernet : un hub a été préféré à un switch car il répète les trames sur tous les brins qui lui sont attachés, ce qui est essentiel pour l'utilisation du logiciel d'observation.

- ➡ Un PC, machine de développement sous Windows NT, relié à la cible MBX par une liaison série. Une autre solution aurait pu emprunter le réseau mais le trafic généré aurait pu perturber le trafic lié à l'évaluation.
- ➡ Une carte PPC MBX sous pSOS+ supportera la souche TASE.2 portée.
- ➡ Un PC sous Windows NT avec le serveur TASE.2, pour servir de référentiel aux échanges client-serveur.
- ➡ Un PC sous Windows NT avec un client TASE.2.

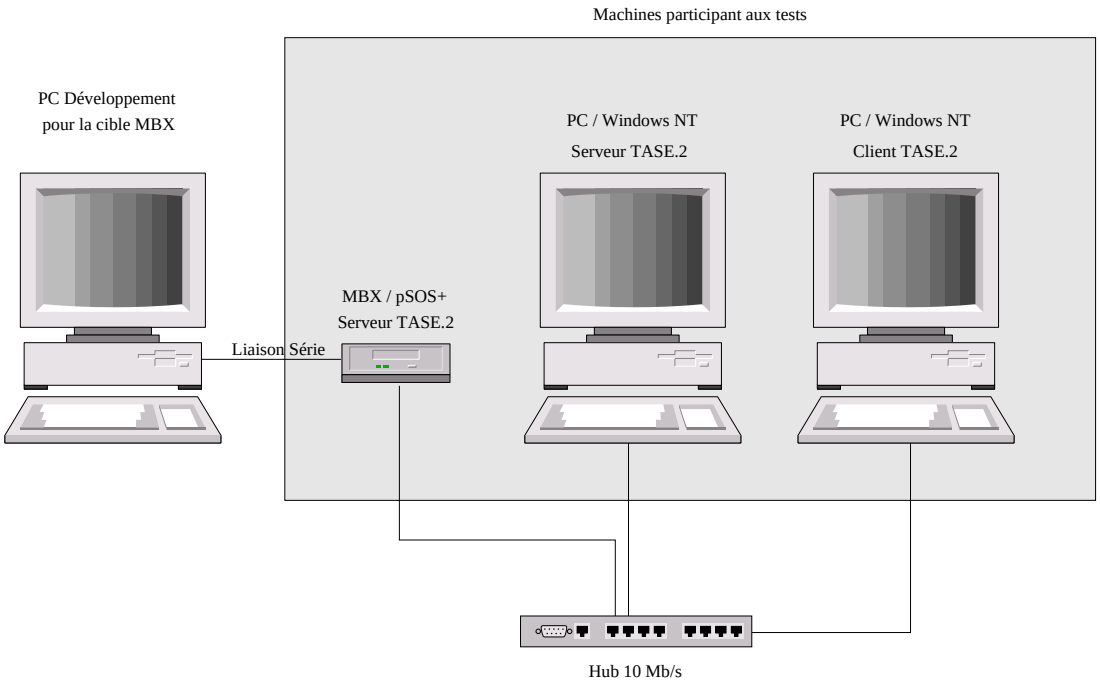

FIG. 4.2: Plate-forme de tests

4.6.3 Mode Opératoire des Tests

Les tests sont des tests "boite noire". Ce qui nous intéresse, c'est l'effet des services plutôt que leur fonctionnement interne. Toutefois, en raison des différences entre les blocs fonctionnels 1 et 2 qui portent sur la prise en compte d'attributs des objets *Transfer Set*, nous sommes contraints à avoir parfois une approche boite blanche pour déterminer l'occurrence de *Transfert Report* à partir de modifications sur le serveur TASE.2. Le moniteur est utilisé pour ce type de tests.

Comme nous avons affaire à un portage, nous devons évaluer si le résultat du portage produit les mêmes effets que ceux de notre version de référence sur une

autre plate-forme. La machine de référence est une machine sous Windows NT qui supporte notre serveur en version binaire.

Notre travail n'est pas de vérifier la conformité du serveur avec le standard TASE.2 bien qu'au fur et à mesure de l'étude, notre travail s'en est très fortement rapproché. De la même manière, nous ne cherchons pas à évaluer le fonctionnement des clients utilisés.

Un client TASE.2 fait face aux serveurs : l'un s'exécutant sur le noyau pSOS+, l'autre sur Windows NT. Le scénario de test est lancé et sollicite le serveur TASE.2 sur le PC Windows NT. Le scénario de test est lancé à nouveau et sollicite le serveur TASE.2 sur le PPC MBX sous pSOS+. Le client génère des traces (log), elles sont examinées et servent à la validation des tests. Le serveur génère lui aussi des traces qui sont aussi examinées. Les traces peuvent être comparées aussi au niveau du client, elles doivent être identiques (excepté les identifications d'entités et les dates).

Pour les tests, s'il s'avérait qu'un service ne fonctionnait pas entre le client et le serveur de référence sur le PC Windows NT, le test de ce service serait abandonné sur la cible pSOS+.

4.6.4 Tests fonctionnels

4.6.4.1 Tests Bloc 1 et Bloc 2

Le tableau ci-dessous décrit les services du bloc 1 et du bloc 2 testés car ce sont les mêmes services qui sont utilisés dans les deux blocs fonctionnels. La distinction des fonctions supportées est relative à l'échange de données : périodique pour le bloc 1, événementiel/changement d'état pour le bloc 2. Cet aspect touche l'objet DS Condition Monitoring.

VCC, Association et Table Bilatérale			
Services TASE.2		**Service côté Client(s)**	**Service côté Serveur**
Initiate (à l'initiative du Client)		Démarrage d'une association	Lancement du serveur (ouverture de connexion TCP passive). Connexion TCP ouverte, Association établie.
Conclude (Arrêt d'une association)	A l'initiative du Client	Observation (traces) de l'arrêt de l'association par la rupture de connexion TCP.	Observation (traces) de l'arrêt de l'association par la rupture de connexion TCP.

VCC, Association et Table Bilatérale *(suite)*			
Services TASE.2		**Service côté Client(s)**	**Service côté Serveur**
	A l'initiative du Serveur		Conclude à l'initiative du serveur pas testable (pas accessible localement)
Abort (association pré-établie)		**Abort Client** Arrêt du client par CTRL-ALT-SUPPR, relancement ,nouvelle association établie avec le serveur.	**Abort Serveur** Rupture de connexion TCP, et donc d'association : **le serveur doit se remettre en attente.** Le serveur continue les échanges (nvelle association).
Plusieurs Associations (deux clients)		Deux clients accèdent à deux domaines différents sur le serveur	Constat de l'établissement de deux associations avec le serveur
		Deux clients accèdent à deux domaines différents sur le Serveur et Abort d'un client.	Constat de l'établissement de deux associations avec le Serveur. Après Abort, une seule association fonctionne.
Supported Features		Obtenir la liste des blocs fonctionnels supportés par le serveur. Code attendu : OxC8 (Blocs 1,2 et 5). Affichage à l'utilisateur	Envoi des blocs supportés. Affichage local blocs (trace écran).
TASE.2 Version		Obtenir la version de TASE.2 du serveur. Affichage (traces). Numéro attendu : 6.1 (ou 1996, 8). Obtenue et affichée à l'utilisateur.	Envoi de la version TASE.2. Affichage local (trace écran moniteur)
Table Bilatérale et Contrôle d'Accès		Obtenir la Version de la table bilatérale	Le serveur envoie la version de sa Table Bilatérale.

VCC, Association et Table Bilatérale *(suite)*		
Services TASE.2	**Service côté Client(s)**	**Service côté Serveur**
	Mise en relation de table bilatérale avec un client autorisé Obtenir l'identificateur de table bilatérale associée au client sur le serveur. Celui-ci est affiché dans les traces activées sur le client. Il est obtenu par le logiciel client et affiché à l'utilisateur	Envoi de l'identificateur de table bilatérale au client.
	Mise en relation de table bilatérale avec un client non-autorisé [2] Un client essaie d'obtenir une table bilatérale pour laquelle il n'est pas autorisé à l'aide d'un "client MMS léger"	Refus par le serveur. Refus par le serveur de fournir la table bilatérale au client léger (pas nécessairement dit dans la norme TASE.2, puisqu'on accède au niveau MMS) [3]

TAB. 4.1: Tests des services liés aux VCC, Association et Table bilatérale.

[2]Un client est dit légal s'il est autorisé à accéder à une information.

[3]Idéalement, le test à mener est l'usurpation d'identité, c'est à dire qu'un client A demande à accéder à un domaine prévu pour un client B. Mais comme il semble que MMSd ne fait pas le tri sur les demandes d'association, on ne peut pas faire le test. Il faudrait regarder ce que dit la norme sur le contrôle d'accès au niveau TASE.2. Il ne me semble pas qu'elle impose au serveur de contrôler l'identité du client qui fait une tentative d'association. C'est plutôt au client de comparer l'identificateur de table bilatérale qu'il récupère avec celle qui est configurée en interne, et de conclure l'association en cas de divergence. C'est à examiner plus en détail dans le cadre de tests de conformité.

Data Value		
Services TASE.2	**Service côté Client(s)**	**Service côté Serveur**
Get Data Value Names	Pour chaque variable, accès à son nom Observation au niveau des traces et au niveau de l'utilisateur.	Le serveur rend à chaque fois le nom des variables. Observation au niveau des traces console.
Get Data Value Type	Pour chaque variable, accès à son type Observation au niveau des traces et au niveau de l'utilisateur.	Le serveur rend à chaque fois le type des variables. Observation au niveau des traces console.
Get/Set Data Value	Pour chaque variable, préalablement configurée avec un droit d'accès en lecture et écriture, accès à son contenu par l'enchaînement suivant : Lecture du contenu des variables Modification du contenu de ces variables, Lecture du contenu de ces variables à nouveau, et observation de la modification Observation au niveau des traces et au niveau de l'utilisateur.	Les variables ont toutes une valeur initiale de A. Le serveur retourne cette valeur au premier Get. Le Set sur les entiers et les réels fait +B, pour les états on passe à l'état complémentaire. Le Get suivant retourne la nouvelle valeur ou le nouvel état. Observation au niveau des traces console.
Droit d'Accès (Write Access, Read Access)	Essai d'écriture d'une variable en lecture seulement Observation au niveau des traces et au niveau de l'utilisateur.	Refus dans tous les cas Observation au niveau des traces console.
	Essai de lecture d'une variable en écriture seulement Observation au niveau des traces et au niveau de l'utilisateur.	Refus dans tous les cas Observation au niveau des traces console.
Data Value de portée VCC	Définition d'une variable entière simple discrète en accès lecture dans le VCC hors d'un domaine, Accès avec Get(A) Observation au niveau des traces et au niveau de l'utilisateur.	Le serveur retourne A. Observation au niveau des traces console.

Data Value *(suite)*		
Services TASE.2	**Service côté Client(s)**	**Service côté Serveur**
	Définition d'une variable entière simple discrète en accès lecture et écriture dans le VCC hors d'un domaine, Accès avec Get(A) puis Set(A+B) puis Get(A+B). Observation au niveau des traces et au niveau de l'utilisateur.	Le serveur retourne A, puis modifie la variable à A+B, puis retourne A+B. Observation au niveau des traces console.

Data Value *(suite)*		
Services TASE.2	**Service côté Client(s)**	**Service côté Serveur**
Partage de variables réelles par des variables TASE.2 de domaines différents (portée VCC).	Accès Get à une variable entière simple en accès lecture/écriture par un premier client. Ce Client fait Set et met la valeur A dans la variable TASE.2 de son domaine. Le second client fait Get sur une variable entière simple en accès lecture/écriture dans un second domaine. Le second client fait Set et met la valeur B dans la variable TASE.2 de son domaine. Accès Get à la variable entière simple en accès lecture/écriture par le premier client. Le get observe la valeur B aussi. Observation au niveau des traces et au niveau de l'utilisateur.	Le serveur a défini un même gestionnaire de variable réelle pour les deux variables TASE.2 différentes (domaines différents) accédées par les deux clients. Chaque Set ou chaque Get sur les variables TASE.2 a pour effet de lire/écrire la même variable réelle. Le serveur retourne la valeur initiale : 0. Le serveur dépose la valeur A dans la variable réelle via le handler connecté par le serveur à la variable TASE.2 dans le domaine du client 1. Le serveur lit une variable TASE.2 différente mais une variable réelle identique et retourne la valeur A au second client. Le serveur dépose la valeur B dans la variable réelle via le handler connecté par MMSd à la variable TASE.2 dans le domaine du client 2. Identique à la variable réelle vue sous un autre nom TASE.2 par le premier client. Le serveur retourne la valeur B au client 1. Observation au niveau des traces console et du moniteur.

TAB. 4.2: Tests des services liés aux Data Value

Data Set [4]		
Services TASE.2	**Service côté Client(s)**	**Service côté Serveur**
Create Data Set	Création d'un Data Set, spécification de ses paramètres pour initialiser les attributs sur le serveur Observation au niveau des traces et au niveau de l'utilisateur	Un Data Set et ses paramètres sont créés sur le Serveur, mais aucun échange n'est généré, il faut attendre le Start Transfer sur le Transfer Set associé Observation au niveau des traces console.
Delete Data Set	Destruction d'un Data Set Observation au niveau des traces et au niveau de l'utilisateur	Le Data Set Spécifié est détruit. Observation au niveau des traces console.
Get Data Set Element Values	Lire la valeur de chacun des éléments d'un Data Set Observation au niveau des traces et au niveau de l'utilisateur	Le serveur retourne la valeur des éléments du Data Set, un code retour indique le succès ou l'échec de la lecture pour chaque variable du Data Set Observation au niveau des traces console.
Set Data Set Element Values (accès en écriture positionné)	Écrire la valeur de chacun des éléments d'un Data Set Observation au niveau des traces et au niveau de l'utilisateur	Le serveur modifie la valeur des éléments du Data Set spécifié, un code retour indique le succès ou l'échec de l'écriture de chaque variable du Data Set Observation au niveau des traces console.
Get Data Set Names	Obtenir le nom des Data Set du VCC Observation au niveau des traces et au niveau de l'utilisateur	Le serveur retourne la liste des Data Set du VCC Observation au niveau des traces console.
Get Data Set Element Names	Obtenir le nom des Data Value du Data Set Observation au niveau des traces et au niveau de l'utilisateur	Le serveur retourne la liste des Data Value du Data Set Observation au niveau des traces console.

TAB. 4.3: Tests des services liés aux Data Set.

Data Set Transfer Set		
Services TASE.2	**Service côté Client(s)**	**Service côté Serveur**
Get Next DSTransfert Set Value	Demande d'un Transfer Set disponible au serveur	Allocation d'un Transfer Set disponible pour le client
Start Transfer	Avec le Transfer Set alloué, demande de démarrage de l'échange périodique, tous les attributs du Transfer Set sont spécifiés Observation au niveau des traces et au niveau de l'utilisateur.	Le Transfer Set est ENABLE. Le serveur génère des Transfer Report périodiquement compte tenu des paramètres spécifiés lors du Create Data Set Observation sur la trace console
Stop Transfer	Indication d'Arrêt Observation au niveau des traces et au niveau de l'utilisateur.	Le Transfer Set est DISABLE. Le Serveur ne génère plus de Transfer Report Observation sur la trace console
Data Set Condition Monitoring	Dépendant du bloc fonctionnel cf. 4.6.4.2	
Test de Robustesse : Tenue au nombre de variables (Essai à X variables)	Définition d'un domaine avec X Variables, toutes les variables dans un même Data Set, et donc un seul Transfert Set associé, affichage périodique sur Y heures.	Le serveur envoie périodiquement la valeur des données du Transfer Set sans défaillir pendant Y heures.

TAB. 4.4: Tests des services liés aux Data Set Transfer Set

Tests des Services Communs aux Blocs 1 et 2

4.6.4.2 Tests Data Set Condition Monitoring pour le Bloc1

Il y a une corrélation entre certains attributs des abstractions Data Set, Transfer Set et Transfer Report. Les conditions de déclenchement de compte-rendu sont dans le Transfer Set, les paramètres de génération de compte-rendu sont des attributs du Data Set. Le Transfer Report contient les Data Value du Data Set.

Par rapport au tableau précédent la position des colonnes Serveur et Client sont inversées.

Les variables concernées ici sont des variables simples, on prend par exemple une variable entière et une variable réelle.

Data Set Transfer Set (bloc 1)			
Attributs		**Serveur**	**Client**
EventCodeRequested / OtherExternalEvent		L'attribut indique quel événement externe est apparu. Observation sur la trace console	Indication d'événement externe (périodique). Observation dans la trace
DSTransmissionPars (figure comme attribut d'un Transfert Set) (hypothèse de comportement périodique due au bloc 1 suppose que RBE vaut faux)	StartTime	La valeur de Start Time a été spécifiée lors d'un Start Transfer : Valeur 0, Date. Observation sur la trace console	Le client commence à recevoir des Transfer Report, immédiatement ou à partir de la date spécifiée. Observation au niveau des traces et au niveau de l'utilisateur.
	Interval	Émission périodique d'un Transfer Report, la période est celle spécifiée dans le Start Transfer, tous les champs sont renseignés pour chaque période Les variables sont modifiées via le Moniteur. Observation sur la trace console.	Réception périodique de Transfer Report Observation au niveau des traces et au niveau de l'utilisateur.
		Produire plusieurs changements successifs à l'aide pendant le délai Interval, seul le dernier est envoyé dans le Transfer Report. Le délai Interval est de plusieurs minutes pour laisser le temps de faire les changements successifs. Observation sur la trace console de la dernière valeur produite.	Réception périodique de la dernière valeur produite pendant l'intervalle. Observation au niveau des traces et au niveau de l'utilisateur de la dernière valeur produite.

Data Set Transfer Set (bloc 1) *(suite)*				
Attributs			**Serveur**	**Client**
	DS-Condition Requested (figure comme attribut dans un Data Set)	Interval-TimeOut	L'attribut est positionné avec le Start Transfer Observation moniteur	Résultat correct sur réception de Transfer Report
		Operator Request	L'attribut est positionné avec le Start Transfer Une interaction opérateur génère un Transfer Report Observation moniteur	Résultat correct sur réception de Transfer Report
		Other External Event	L'attribut est positionné avec le Start Transfer Sur occurrence d'un événement externe un Transfer Report est généré. Observation moniteur	Observable dans la trace

TAB. 4.5: Tests de la génération des envois sur condition (bloc 1)

4.6.4.3 Tests Data Set Condition Monitoring pour le Bloc 2

D'après notre compréhension de la norme dans la partie conformité p.113 [IEC96a], la gestion des attributs RBE, Integrity Check, TLE, Buffer Time, et Critical n'est pas faite dans le Bloc 1. On peut supposer que la mise en œuvre des fonctions du bloc 2 implique leur prise en compte aussi dans l'émission de Transfer Report périodique.

Les variables concernées sont de tout type : simple, avec attribut de qualité, avec estampille temporelle, avec compteur COV (nombre de changements de valeur). A titre d'exemple, on prend une variable entière et une variable réelle pour chacune des possibilités enoncées ci-dessus. Il y a quatre DataSet, un par type (simple, avec qualité, avec estampille, avec COV) et autant de Transfer Set.

Data Set Transfer Set (bloc 2)			
Attributs		**Serveur**	**Client**
EventCodeRequested		L'attribut indique quel événement externe est apparu. Observation sur la trace console.	Indication d'événement externe (à l'apparition) Observable dans la trace
DSTransmission-Pars (figure comme attribut d'un Transfert Set) (hypothèse de comportement événementiel due au bloc 2 suppose que RBE vaut vrai)	RBE	RBE vaut vrai En fin de délai (Interval, IntegrityCheck ou Buffer Time), le Transfer Report ne contient que les variables ayant subies une modification. Chaque changement est fait à l'aide du moniteur. Observation sur la trace console	RBE a été positionné lors du Start Transfer Set Observable dans la trace, et au niveau utilisateur La réception de Transfer Report sur changement se fait en fonction des différentes temporisation ci-dessous. Le client ne doit constater aucune émission périodique.
	IntegrityCheck (corrélé avec IntegrityTimeOut (RBE vrai))	Integrity Check vaut U L'attribut Integrity Check permet de générer un Transfer Report à la fin de l'intervalle U (rappelons que ce mécanisme permet de mettre en place une synchronisation des données partagées entre un client et un serveur et répliquées chez chaque partenaire indépendamment des changements qui ont pu se produire). Un changement est opéré à l'aide du moniteur Observation à l'aide du moniteur	Paramètre pour le calcul de l'intégrité d'une valeur, durée de valeur U, a été positionné lors du Start Transfer du Data Set Transfer Set associé Observable dans la trace, et au niveau utilisateur.

Data Set Transfer Set (bloc 2) *(suite)*			
Attributs		**Serveur**	**Client**
	TLE	TLE vaut V unités de temps. L'attribut indique le délai disponible pour envoyer un transfer report. Observation à l'aide du moniteur	TLE a été positionné lors du Start Transfer du Data Set Transfer Set associé La valeur de l'attribut TLE est observable dans la trace
		Il faudrait retarder l'envoi d'un transfer report pour voir le comportement du serveur. Le comportement attendu étant que le serveur élimine le report fautif et que le report suivant soit envoyé sans conditions. NON TESTABLE EN TESTS BOITE NOIRE OU BLANCHE COMPTE TENU DES OUTILS DISPONIBLES.	
	BufferTime (corrélé avec Object Change, RBE vrai)	Buffer Time vaut W. L'attribut indique le délai de tamponnement d'un changement d'objet avant d'envoyer un Transfer Report. Instrumentation et Observation à l'aide du moniteur	Le changement est reçu par le client dans un Transfer Report La valeur de l'attribut Buffer Time est observable dans la trace, l'utilisateur reçoit le changement de l'objet
		Produire plusieurs changements successifs, seul le dernier est envoyé dans le transfer report. Instrumentation et observation avec le moniteur.	Le dernier changement est reçu par le client dans un Transfer Report. La valeur de l'attribut Buffer Time est observable dans la trace, l'utilisateur reçoit le dernier changement de l'objet

Data Set Transfer Set (bloc 2) *(suite)*				
Attributs			**Serveur**	**Client**
			La valeur de Buffer time est mise à un temps long (> une minute), il faut alors faire une succession de modifications sur le serveur avec le moniteur et vérifier que ça n'est qu'au bout du Buffer Time que le client reçoit un transfer report avec la dernière valeur mémorisée. Le compteur COV compte le nombre de modifications faites avec le moniteur.	
	Critical		L'attribut Critical indique si un client doit accuser réception d'un transfer report. Le serveur reçoit un acquittement. Observation dans la trace console.	Critical a été positionné lors du Start Transfer Set A la réception du Transfer Report correspondant, le client envoie un acquittement La valeur de l'attribut Critical est observable dans la trace
	DSCondition-Requested	Object Change	L'attribut est positionné avec Start Transfer. Observation avec le moniteur	Réception de Transfer Report sur Variable avec Attribut COV
		Integrity Time Out	L'attribut est positionné avec Start Transfer. Observation avec le moniteur	Réception de Transfer Report sur Variable avec Quality Flag
		Other External Event	L'attribut est positionné avec Start Transfer Observation avec le moniteur	Réception de Transfer Report sur Variable liée à un événement applicatif

TAB. 4.6: Tests de la génération des envois sur condition (bloc 2)

4.6.4.4 Tests Bloc 5

Le tableau ci-dessous décrit les tests effectués sur le bloc 5. Les équipements sont de deux types, exclusif ou non exclusif, le Tag peut varier ainsi que l'état des différents attributs du device. Nous avons regroupé les cas qui n'avaient pas d'intérêt. Ainsi tester l'opération de sélection sur un device non SBO pour toutes les valeurs du Tag possible n'est pas pertinent puisque cette opération renvoie un échec dans la norme de toute façon.

Device (bloc 5)			
Services/Fonctions TASE.2		**Service lancé côté Client(s) et attendus**	**Service lancé côté Serveur et attendus**
Select [5]	Device non sélectionné (IDLE)	Le client demande à sélectionner un device en mode exclusif (SBO device), libre d'accès (état IDLE). On pourra vérifier cet état grâce au moniteur.	Le serveur doit retourner un succès. Il doit positionner l'état du device à ARMED. Il doit également retourner au client le CheckBackID mentionné dans la table bilatérale. Vérifier le nouvel état avec le moniteur.
	Device sélectionné (ARMED)	Le client demande à sélectionner un device en mode exclusif qui a déjà été sélectionné par un autre client (état ARMED). Le moniteur pourra servir à positionner directement ce champ.	Le serveur doit retourner un échec au client. Il ne doit pas modifier l'état du device. La norme précise que le CheckBackID doit être renvoyé lors d'un échec. Le vérifier sur la trace.
	Tag à OPEN-CLOSE-INHIBIT	Le client demande à sélectionner un device dont l'accès est restreint en ouverture/fermeture. Par ailleurs l'état est IDLE.	Le serveur doit retourner un échec.
	Tag à CLOSE-ONLY-INHIBIT	Le client demande à sélectionner un device dont l'accès est restreint en fermeture seulement.	Le serveur doit retourner un succès.

[5]Le nom du device est passé en paramètre pour l'identifier. Le vérifier sur la trace

Device (bloc 5) *(suite)*				
Services/Fonctions TASE.2			**Service lancé côté Client(s) et attendus**	**Service lancé côté Serveur et attendus**
	Device non SBO		Le client demande à sélectionner un device qui n'est pas en mode exclusif.	Le serveur doit retourner un échec. Cette opération n'a sémantiquement pas de sens.
Operate [6]	Device SBO & IDLE		Le client essaye de faire une opération sur un device en mode exclusif qui doit être sélectionné auparavant (ARMED).	Le serveur doit renvoyer un échec.
	Device SBO ARMED by other		Le client essaye de faire une opération sur un device qui a été sélectionné par un autre client.	Le serveur doit renvoyer un échec.
	Device non SBO or ARMED	Tag NO-TAG	Le client a d'abord sélectionné le device si besoin, il lance une opération.	Le Tag n'est pas positionné, il n'y a aucune restriction sur les opérations permises. Le serveur doit toujours renvoyer un succès. Après l'opération l'état du device doit repasser à IDLE. Le vérifier avec le moniteur.
		Tag OPEN-CLOSE	Le client a d'abord sélectionné le device si besoin, il lance une opération.	Le Tag interdit des opérations sur le device. Le serveur doit renvoyer un échec obligatoirement. Notons que ce test implique obligatoirement un device non SBO, un SBO n'aurait pas pu sélectionner le device préalablement (cf. select).

[6] Le checkBackID est passé en paramètre pour identifier le device. (Vérifier sur la trace). Par défaut, la valeur de l'opération est écrite dans un attribut du device. Lire cette valeur grâce au moniteur permettra de vérifier l'exécution de l'opération.

Device (bloc 5) *(suite)*				
Services/Fonctions TASE.2			**Service lancé côté Client(s) et attendus**	**Service lancé côté Serveur et attendus**
		Tag CLOSE-ONLY	Le client a d'abord sélectionné le device si besoin, il lance une opération.	Le Tag limite les opérations permises. Le serveur doit renvoyer un succès sauf si l'opération est Raise ou Close.
Set Tag			Fonctionnalité visiblement non supportée dans MMSd, les Tag sont fixés à la configuration, donc à la compilation et ne sont pas modifiables.	
Get Tag				

4.7 Bilan de l'Étude

Les propriétés de sécurité du serveur sont inexistantes (et donc non testées lors de cette étude) et mériteraient d'être repensées dans une logique d'utilisation à travers Internet (réseaux privés virtuels par exemple). Ce défaut n'est pas le fait du fournisseur, mais prend son origine dans la norme TASE.2 qui laisse toute liberté à l'implanteur, et, qui n'envisage aucune fonction de sécurité sérieuse (pas de véritable contrôle d'accès, pas d'authentification, pas de confidentialité, pas d'intégrité ni de non-répudiation).

Dans la mesure, où les produits que nous avons utilisés sont fondés sur des normes ISO, et qu'il est nécessaire de les qualifier, il nous semble important de pousser cette logique jusqu'au bout, et d'utiliser des produits certifiés pour leur conformité à TASE.2, et leur interopérabilité. Cette propriété nous aurait fait gagner du temps car lors de certaines erreurs ou lors de certains tests, certaines investigations auraient pu être éliminées. Nous aurions évité certains tests et parfois nous n'aurions pas douté de la bonne mise en œuvre de la norme par la souche TASE.2... nous aurions remis en question notre portage de la norme. Cette exigence a une contrepartie, pour être définitif, notre travail devrait alors faire aussi l'objet de tests de conformité et d'interopérabilité.

Notre étude n'aborde pas les problèmes de performances. Ces tests permettraient de vérifier que la souche TASE.2 sur pSOS+/PowerPC/MBX donne des performances au moins identiques à celles annoncées par le fournisseur sur la version de référence. Par ailleurs, la norme définit un comportement temporel d'un serveur TASE.2 qui tient plus de la ponctualité que de la rapidité, il faudrait pouvoir valider exactement cette temporalité et non pas simplement en fonction

des résultats reçus côté client.

La souche TASE.2, en version de référence comme en version pSOS+/PowerPC/MBX, doit être encore éprouvée avec des tests applicatifs liés aux besoins spécifiques d'EDF R&D :

- Intégration à une centrale en parallèle des solutions actuelles,
- Interopérabilité entre centres négociateurs-fournisseurs- transporteurs-consommateurs d'énergie nationaux ou européens avec des souches issues de fournisseurs différents et avec des systèmes d'exploitation différents,
- Tenue aux contraintes de qualité de service : délais de tranfert, volume en nombre d'objets, promptitude et péremption des valeurs, tenue à la charge ...
- Résistance aux attaques de sécurité

Ces perspectives suivent les recommandations générales de tests d'intégration qu'on retrouve classiquement dans les applications d'informatique industrielle.

4.8 Conclusion

Nous nous proposons donc de répondre aux attentes d'EDF R&D en proposant une solution plus adaptée aux besoins d'ingénierie logicielle moderne et qui est fonctionnellement équivalente au protocole TASE.2. Cette solution doit permettre de pallier les défauts du protocole (manque de sécurité, passage par messages, implantations propriétaires) et y ajouter des réponses en ce qui concerne les nouvelles problématiques d'ouverture, d'homogénéité, etc ... déjà énoncées.

C'est dans cette optique que nous développons un prototype basé sur un bus logiciel CORBA. En l'état des connaissances au moment de ce travail, plusieurs technologies répondaient aux besoins : CORBA, Les Web Services, .Net, J2EE. Nous avons choisi CORBA pour plusieurs raisons :

- C'est une technologie ouverte et normalisée,
- Des implémentations existent qui s'exécutent sur des systèmes temps réel (pSOS+,vxWorks ...) Et ce, contrairement à toutes les autres technologies au moment de l'étude,
- C'est une technologie souple puisqu'elle permet de choisir les langages d'implémentation (contrairement à J2EE) et les systèmes d'exploitation (contrairement à .Net),
- Des notions de temps réel existent même si elles sont à évaluer, elles semblaient, au moment du choix être un axe d'évolution intéressant,
- CORBA introduit dans sa dernière version, la notion de composant, que nous avons pu utiliser pour une évolution de notre prototype.

Nous voulons cependant faire remarquer qu'une grande partie du travail fait, à savoir en particulier la spécification en termes d'objets, puis de composants et d'interfaces et d'événements, est transposable directement sur toute architecture supportant ces paradigmes.

Cette solution, ses avantages, en quoi elle répond à la problématique de départ, et comment elle a été réalisée, sont présentées dans le chapitre suivant.

Chapitre 5

TASE.2 Objet

Sommaire

5.1 Introduction

L'étude de plusieurs applications industrielles d'EDF visant la supervision/médiation et le contrôle-commande nous a permis de remarquer que beaucoup de mécanismes similaires sont mis en jeu mais avec des technologies différentes : utilisation de modes messages, modes connectés, nécessité de modes de diffusion...

D'où l'idée d'intercaler entre les différents équipements et les applications, une couche d'homogénéisation avec des fonctions et des abstractions de programmation identiques. Le besoin de l'approche objets distribués se justifie pour plusieurs raisons :

- ➡ Structuration en objets du modèle de conception (UML), utilisation naturelle de langages de programmation orientés objet (point de vue ingénierie)
- ➡ Possibilité d'utiliser le méta-modèle RM-ODP et son extension ReTINA [4] pour modéliser les contraintes de QoS.
- ➡ Bénéficier de tous les services de distribution proposés par le middleware et non disponibles avec le modèle ISO. Par exemple s'affranchir de la localisation des objets et de leur adressage en utilisant le service de noms disponible voire un courtier.

Dans cette optique, trois grands modèles d'architecture émergent : OMG/CORBA [33], MS/DCOM [17] (les Web Services à base de SOAP/XML et .NET n'étaient pas disponibles au moment du choix) et JAVA-RMI.

Dans ce nouvel environnement résultant de l'approche répartie appliquée aux systèmes et applications d'informatique industrielle, il est alors nécessaire de réévaluer la nature des contraintes temps réel. L'ordre de grandeur de ces contraintes est de l'ordre de cinq cents millisecondes dans la centrale à plusieurs secondes pour la médiation et elles sont de type déterministe.

Dans le contexte des applications visées, il pourrait être utile de permettre l'acheminement des données multimédia (voix, vidéo...) en même temps que les données plus classiques de contrôle commande pour faire de la télé-surveillance par exemple. Ainsi, dans certaines applications, le son émis par un matériel (pale de turbine par exemple) doit être analysé par un personnel qualifié qui peut en déduire des données essentielles [1] (usure, mauvais fonctionnement, détérioration...), ce son pourrait être transporté dans le système réparti. Il existe également des besoins vidéo, pour localiser les pannes géographiquement entre autres.

On se rend compte que certaines des contraintes précédemment évoquées, comme le délai borné de bout en bout, correspondent au concept de qualité de service (QoS) telle que l'on peut la rencontrer dans des réseaux ATM. On peut voir une spécification de contraintes de qualité de service comme un contrat à respecter. Une bonne gestion de la qualité de service nécessite deux choses :

- ➡ Une formalisation qui permette aux applications d'exprimer cette QoS.
- ➡ Des outils qui permettent au réseau de mettre en oeuvre la QoS requise.

On peut alors se demander si l'expression de ces contraintes sous la forme de QoS dans notre cadre applicatif n'aiderait pas à formaliser mieux ces contraintes,

[1] Citons ici le cas des "oreilles d'or", qui fait référence aux rondiers sur site qui à force d'habitude et de métier étaient capables de repérer une panne potentielle en fonction du bruit d'un équipement.

et donc ensuite à mieux mixer, déclarer, utiliser ces contraintes. La différence est que l'on parle de QoS au niveau du bus logiciel et non plus au niveau réseau de communication et protocole. De plus on pourra évaluer la faisabilité des contraintes temps réel exprimées dans ce même cadre de Qualité de Service. Cependant, il nous est apparu que les bus logiciels (même CORBA "Temps Réel", voir l'étude menée sur ce sujet [83]) n'offraient pas les mécanismes nécessaires à la prise en compte de la QoS. Différentes extensions de CORBA pour pouvoir gérer la QoS sont discutés dans les travaux de J. Rodriguez [44] [43].

De plus, d'un point de vue fonctionnel, notre expérience sur TASE.2 et sur les produits existant conforme à la norme (cf. 4) nous ont amené à réfléchir à une implantation objet du protocole. En effet, TASE.2 repose sur MMS qui manipule des concepts objets. Il est donc naturel de songer à la technologie objet afin de profiter de tous ses apports reconnus en termes d'ingénierie logicielle.

Le passage à un ORB nous permet d'utiliser le modèle RM-ODP/ReTINA [4] pour faire notre modélisation afin d'être indépendant du choix ultérieur qui sera fait quant aux technologies d'implantation. Le modèle RM-ODP est un méta-modèle qui manipule des concepts d'objets distribués, ce qui correspond tout à fait à la modélisation d'un protocole d'échange de données en objet ! De plus, il permet de spécifier au niveau du modèle des contraintes de Qualité de Service, et donc notre modèle actuel pourra être étendu de ce point de vue. Le choix de CORBA est lié en partie à ce choix puisque CORBA est une instanciation du modèle RM-ODP selon le point de vue de l'ingénierie.

5.2 RM-ODP

5.2.1 Introduction

L'évolution rapide des applications distribuées a entraîné le besoin d'une standardisation d'un méta-modèle pour faire du calcul distribué (Open Distributed Processing). En conséquence, un modèle de référence (RM-ODP) a été défini conjointement par l'ISO et l'ITU-T. ODP décrit les systèmes qui supportent des applications distribuées hétérogènes dans et entre organisations, via l'utilisation d'un modèle d'interaction commun.

Le but de RM-ODP est d'arriver à :

- la portabilité des applications entre des plateformes hétérogènes,
- l'interoperabilité entre différents systèmes ODP (i.e. échange d'informations et de services au travers du système distribué),
- la transparence à la distribution (i.e. "cacher" les conséquences de la distribution aux développeurs et aux utilisateurs).

Le modèle de référence fournit les bases qui permettent d'organiser les pièces d'un système distribué en un tout cohérent. Il n'a pas pour but de standardi-

ser les composants d'un tel système (comme CORBA) ni d'influencer les choix technologiques.

5.2.2 Principes généraux

Le modèle RM-ODP/ReTINA [4] est basé sur les principes généraux suivants :

- Un **Objet** est une entité contenant (encapsulant) de l'information et offrant des services. Les objets sont d'une granularité quelconque, allant d'un octet à, par exemple, un réseau téléphonique complet).
- Un objet peut interagir uniquement au moyen de ses **interfaces**. Les interfaces d'un objet peuvent être vues comme ses points d'accès, ce qui signifie que toutes les interactions d'un objet avec son environnement doivent survenir à une de ses interfaces. Ces interfaces sont manipulées par référence.
- Deux objets, appelons les O1 et O2, peuvent interagir de deux façons différentes : soit l'objet O1 invoque directement une opération sur O2 (s'ils sont dans le même espace d'adressage) ,soit il invoque la même opération sur un **objet de liaison** dont le rôle est de transmettre cette invocation à O2 et retourner le résultat si nécessaire.

5.2.3 Points de vue

Un des problèmes majeurs dans les systèmes distribués, c'est le spectre des aspects couverts et la complexité inhérente du domaine. C'est mis en évidence par la grande quantité d'informations nécessaires pour la spécification d'un tel système. Cette spécification doit s'occuper, entre autres, du modèle d'informations, de la mise en oeuvre, des choix techniques ... RM-ODP tente de simplifier cette complexité en partitionnant le problème via le concept de point de vue.

Chaque point de vue est une définition complète et autonome du système distribué à destination d'une audience particulière. Le langage de description est alors adapté à cette audience.

RM-ODP définit cinq points de vue :

① **Entreprise** : objectifs et périmètre du système

② **Information** : modèles d'informations véhiculées dans le système

③ **Traitements** : décomposition fonctionnelle du système

④ **Ingénierie** : Infrastructure pour la distribution du système

⑤ **Technologie** : Choix technologiques pour l'implantation du système

Nous nous situons à deux points de vue du modèle RM-ODP dans la suite de ce chapitre : le point de vue traitements quand nous détaillons la décomposition fonctionnelle de notre prototype, le point de vue technologie quand nous nous intéressons aux technologies utilisées pour son implantation. Le point de vue

ingénierie a quant à lui été abordé quand nous avons fait le choix de CORBA comme couche de distribution.

5.2.4 Abstraction des moyens de communication

La caractéristique première de l'architecture RM-ODP est de proposer une abstraction explicite pour les moyens de communication. Les objets de liaison sont fondamentaux dans beaucoup d'environnements applicatifs pour au moins deux raisons :

① Ils offrent une abstraction naturelle et adaptée pour représenter la sémantique des communications et leurs propriétés comme le multicast, les échéances ou bien encore la sécurité (point de vue des traitements).

② Ils encapsulent les mécanismes utilisés à la fois dans les exécutions locales et distantes pour les communications : par exemple, les algorithmes d'encodages/décodages, la gestion des caches ou des thread, les politiques d'ordonnancement, etc ... Les objets de liaison sont en général des objets composites, distribués sur plusieurs espaces d'adressage différents (point de vue ingénierie).

Cette architecture rend possible l'insertion de formes arbitraires de liaison entre les objets, au delà du modèle implicite de liaison pour les modèles client / serveur, utilisé par des architectures standard comme CORBA ou Java-RMI.

Un exemple d'objet de liaison, instancié pour TASE.2 est donné dans la figure 5.2.

5.2.5 Indépendance du nommage et de la localisation

La seconde caractéristique de l'architecture RM-ODP est que le nommage des interfaces ne dépend pas de la faculté à pouvoir les accéder. Cette propriété rend l'architecture très flexible puisqu'elle permet :

➡ la transmission de la référence d'une interface via une liaison donnée même si cette référence ne peut pas participer à des liaisons du même type

➡ la participation d'une interface donnée à plusieurs liaisons de différents types

➡ la désignation d'une interface même quand celle-ci n'est pas accessible (panne, mobilité, ...)

Le modèle RM-ODP définit donc un noyau minimal dont le rôle est de fournir un environnement générique qui peut être utilisé par toute fabrique d'objets de liaison pour créer et gérer des liaisons de tout type. Avec cette architecture, il est possible d'introduire de nouvelles fabriques de liaisons écrites par les développeurs applicatifs et potentiellement découvertes et installées à l'exécution.

Il fournit également une abstraction explicite pour les références d'interface, qui peuvent être accédées et modifiées par les fabriques de liaisons. Différentes fabriques de liaisons coexistent dans un même espace d'adressage, apporte les fonctions nécessaires à chaque type de liaison, et manipulent les mêmes références d'interfaces de manière sécurisée. Les fabriques de liaison fournissent principalement deux types de méthodes :

① des méthodes d'export, qui permettent à des interfaces d'être enregistrées par une fabrique de liaisons cible, pour qu'elle puisse créer une liaison pour l'accéder,

② des méthodes de liaison, qui vont établir une liaison entre un ensemble d'interfaces exportées

Dans cette approche, un bus logiciel CORBA ou RMI peut être construit comme une personnalité (un ensemble d'APIs), cela permet de découpler les spécificités des APIs CORBA ou RMI de l'interface du noyau d'un bus logiciel générique, indépendante des personnalités. Cela simplifie les portages d'un bus logiciel à l'autre.

5.2.6 Equations de QoS

Lorsque des contraintes temps réel sont nécessaires aux applications, les objets de liaison les gèrent par l'intermédiaire d'équations de qualité de service. Selon le modèle défini dans RM-ODP [4], la liaison est alors vue comme un flux de données et est associée à une suite d'invocations de méthodes. Chaque invocation (cf fig. 5.1) se décompose en événements d'émission de l'invocation (EI), de réception de l'invocation (RI), d'émission de la réponse (ER) et de réception de la réponse (RR). Les équations de QoS sont exprimées en logique temporelle QL (Qos Language) (défini dans RM-ODP [4]) et traduisent des contraintes temporelles entre les événements précédents.

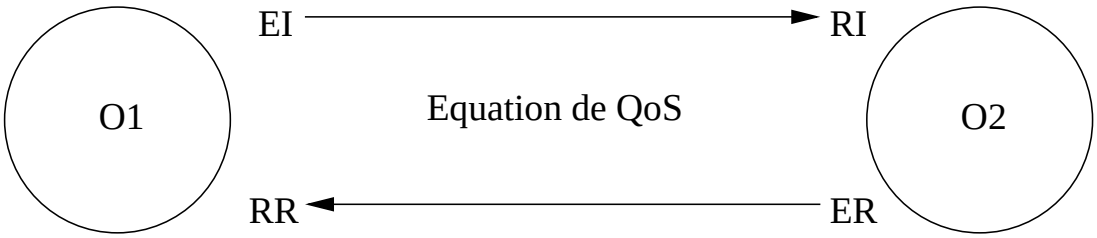

FIG. 5.1:

Les équations sont par exemple, de la forme suivante :

$$d_1 \leq t(e, n+k) - t(e', n) \leq d_2$$

où e et e' sont deux événements, $t(evt, n)$ la date de la n-ème occurrence de l'événement evt et $d1$ et $d2$ des durées. Une telle relation stipule que la n-ème occurrence de l'événement e' doit être de distante de la $(n + k)$-ème occurrence de l'événement e, d'au moins $d1$ et d'au plus $d2$ unités de temps.

5.3 Modélisation

5.3.1 L'objet de liaison TASE.2

Comme déjà dit précédemment, les interactions entre les clients et les serveurs (VCC) TASE.2 sont assez complexes : tous les échanges de données utilisent nécessairement une association explicite créée avant utilisation en réponse à une requête d'un client. Ces échanges de données correspondent soit à des opérations (initiées par le client), soit à des actions (initiées par le serveur, le VCC). En conséquence, toutes les entités TASE.2 sont à la fois client et serveur, comme définis dans le modèle client/serveur.

Le VCC déclare une interface dite **Gestion d'Association**, utilisée pour demander la création et la destruction d'une association.

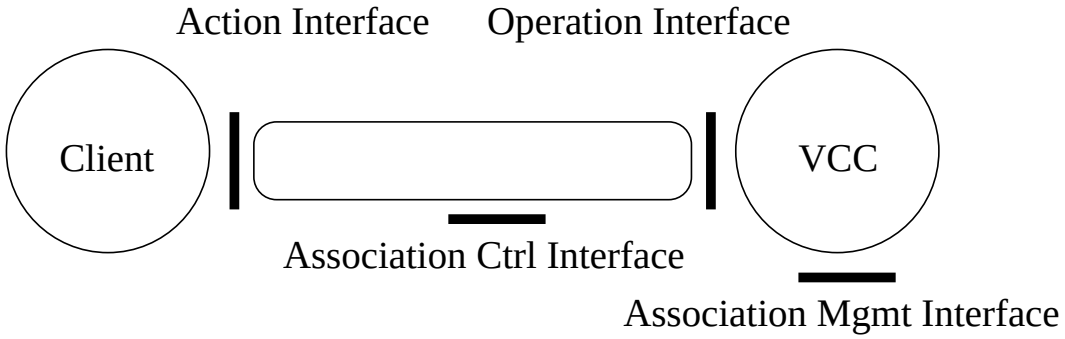

FIG. 5.2: L'objet de liaison TASE.2

La création d'une association entraine la création de trois nouvelles interfaces :

① Une interface **Opération** du côté serveur

② Une interface **Contrôle de l'association** du côté serveur

③ Une interface **Action** du côté du client

④ L'interface de gestion de l'association existe toujours

5.3.2 Spécification des interfaces (IDL)

Une des difficultés de la technologie des applications objet distribuées, correspond aux règles de conversion des appels de méthodes en messages échangés sur

le réseau (et inversement). Les données échangées et les interfaces sont décrites dans un fichier IDL [2], qui est un langage indépendant des matériels, des systèmes et des langages de programmation et donc nous assure d'une bonne cohérence même dans le cas d'un milieu très hétérogène. Cela nous permet aussi une grande latitude quant au choix des langages d'implémentation (par exemple C++ pour le serveur et Java pour un client graphique). Le compilateur d'IDL prend ensuite ces fichiers pour convertir ces appels en échange de messages en utilisant les règles évoquées ci-avant.

Nous nous proposons donc de spécifier complètement un service générique de messagerie temporisée (calqué sur TASE.2) dans le langage IDL de CORBA (un des plus répandus). Le résultat en est, au-delà de la base d'une implémentation possible, une spécification du standard TASE.2 non ambigüe et indépendante d'une technologie particulière, sous forme d'interfaces et de types de données.

La figure 5.3 présente une courte partie des spécifications IDL. Elle concerne l'interface de gestion de données.

```
typedef string DataNameType;
typedef sequence <DataNameType> DataNameSeqType;

exception UnknownDataNameType {};

interface DataValueManagement
{
  DataType getDataValue(in DataNameType name)
      raises (UnknownDataNameType);
  void setDataValue(in DataNameType name, in DataType value)
      raises (UnknownDataNameType);
  DataNameSeqType getDataValueNames();
  DataType getValueType(in DataNameType name)
      raises (UnknownDataNameType);
}
```

FIG. 5.3: L'interface de gestion de données (IDL)

5.3.3 Services de messagerie temporelle orientés objet

Suite à de précédents travaux de prototypage, nous avons appris à modéliser des services de messagerie en utilisant les concepts objet au-dessus de CORBA

[2] **I**nterface **D**efinition **L**anguage

[15] [16]. Les mêmes méthodes peuvent être réutilisées ici. Chaque client et chaque serveur TASE.2 contiennent un serveur CORBA (pour pouvoir faire à la fois des actions et des opérations). Ils sont multi-tâches si possible.

Dans une présentation à l'OMG [69], nous évoquons plusieurs manières d'implémenter les services de TASE.2 au-dessus de CORBA. Nos services sont ici directement construits au-dessus du bus logiciel.

Le serveur CORBA du côté du serveur TASE.2, supporte l'interface du VCC décrite auparavant, c'est-à-dire qu'il implémente les opérations TASE.2. Les méthodes correspondantes sont des invocations de méthodes classiques, retournant des résultats. Les autres objets TASE.2 sont supportés par le VCC en tant qu'objets implémentés dans le langage de programmation choisi, C++ dans notre cas. L'interface du VCC hérite de toutes les interfaces des objets de base. Les communications locales entre le VCC et les ressources physiques utilisent un adaptateur spécifique. Pour les besoins expérimentaux, c'est le bus logiciel qui supporte les interactions locales (pour des raisons pratiques).

Le serveur CORBA du côté du client TASE.2, supporte les services de Rapport de Transfert. Les méthodes correspondantes sont des classiques invocations de méthodes, mais sans valeur de retour.

La figure 5.4 décrit la modélisation de notre prototype.

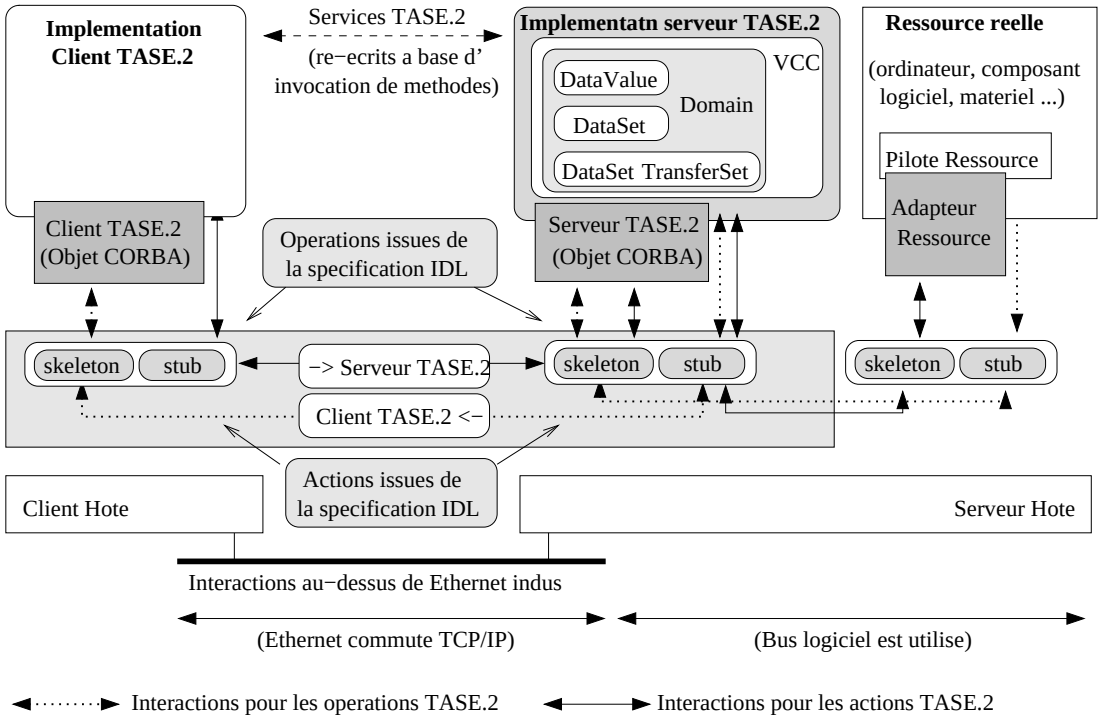

FIG. 5.4: Modélisation du prototype TASE.2 orienté objet

En ce qui concerne les aspects QoS temporels, nous pouvons utiliser la spé-

cification RM-ODP/ReTINA (cf. 5.2.6) pour donner un exemple de spécification lié à notre prototype. Cet exemple est donné dans la figure 5.5, on spécifie des contraintes pour des envois périodiques toutes les 400ms, l'envoi périodique commençant 500ms après l'invocation du client.

Les précédentes caractéristiques temporelles des services TASE.2 auraient pu nous mener à parler de services temps réel. Nous sommes capables, avec notre prototype, d'adresser des contraintes de ce type d'environnement, cependant nous préférons introduire le terme "temporel". C'est plus réaliste au regard des technologies utilisées. Les raisons principales qui nous poussent à faire cette distinction sont :

- L'utilisation d'Ethernet d'abord. Même la plus rapide de toutes les versions d'Ethernet (10 Gb/s) n'apporte pas de ponctualité. La commutation peut apporter des garanties au niveau de la bande passante mais le déterminisme dépend du commutateur et de sa stratégie. 802.1Q/p améliore les choses via le support de priorités de base, cependant, Ethernet n'est toujours pas aussi déterministe que peut l'être FDDI ou ATM.

- IP est un protocole orienté mode datagramme, c'est une approche "mieux possible" rien à voir avec du déterminisme temporel ! Les services intégrés, basé sur le protocole de Réservation de ressources (RSVP) [13] ou les services différenciés basés sur les priorités et des files d'attente gérées de manière efficace dans les routeurs [5] devraient aider à atteindre des services réseau temps réel. Le futur des réseaux temps réel est lié à la Qualité de Service (QoS) et à la Classification des Services (CoS).

- TCP a des mécanismes de base qui vont à l'encontre des besoins du temps réel. Généralement, les acquittements et le contrôle de flux (fenêtre glissante) ne rendent pas les choses faciles. De la même manière, le démarrage lent qui permet d'éviter la congestion mènent à une gigue non déterministe. La proposition TCP Vegas [59] apporte un mieux mais reste un choix insuffisant pour adresser des communications avec des contraintes temps réel.

- L'ORB que nous utilisons, MICO [41], est écrit en C++ et nous garantit des performances acceptables. Cependant, il n'intègre aucun mécanisme temps réel, en particulier dans le traitement des requêtes. Il faudrait le remplacer par une implémentation conforme à la spécification CORBA Temps Réel [31], mais là encore, sans un système d'exploitation sous-jacent temps réel et une forte connexion entre les deux couches, il est vain de vouloir adresser des contraintes temps réel de façon efficace et déterministe comme le montre J. Rodriguez dans sa thèse [42].

```
// TASE.2 server object
interface <operational> DataSetTransferSetMgt
{
 void startDataSetTransferSet (...);
 ...
}

// TASE.2 client object
interface <stream> TransferReportMgt
{
 void transferReport (...);
 ...
}

Provided clause
/* L'\'echange p\'eriodique commence 500 ms
   apr\'es l'\'emission de l'ordre "start" */
time(TransferReportMgt.transferReport.SendEvent,1) -
  time(DataSetTransferSetMgt.startDataSetTransferSet.SendEvent,1)
    = 500 ms

and

/*la p\'eriode est de 400 ms */
for all n, time(TransferReportMgt.transferReport.SendEvent,n+1) -
  time(TransferReportMgt.transferReport.SendEvent,n) = 400 ms

Required clause
/* Ceci n'est possible que si le temps
   de transmission est inf\'erieur à 250 ms */

time(DataSetTransferSetMgt.startDataSetTransferSet.ReceiveEvent,1) -
  time(DataSetTransferSetMgt.startDataSetTransferSet.SendEvent,1)
    < 250 ms
```

FIG. 5.5: Spécification de QoS pour le prototype TASE.2

5.4 Implémentation

Un prototype nommé OpenTAZ a été développé, opérationnel et qui a fait l'objet de présentations [64][71].

5.4.1 MICO (Mico Is COrba)

MICO est une implémentation du standard CORBA. Le projet a démarré en décembre 1996. Depuis, de grandes entités, qu'elles soient publiques (commission européenne, Facultad de Informatica Universidad Politecnica de Madrid ...) ou privées (AT&T, Deutsche Telekom, Alcatel ...), ont supporté cet effort qui a pour but de disposer d'une implémentation gratuite et open source la plus complète possible de la norme CORBA. MICO a été déclaré "Conforme CORBA" par l'OpenGroup, démontrant sa qualité et la possibilité réelle de l'utiliser en production.

A l'heure actuelle, MICO est devenu un produit mature, disposant d'un support commercial, d'un support gratuit via les intervenants sur le projet, et auquel déjà beaucoup de grands projets à la fois publics et commerciaux doivent leur succès. A titre personnel, je l'utilise également dans mon travail en entreprise, sur un projet de recherche qui fait intervenir une cinquantaine de personnes minimum.

A l'origine, MICO a été développé par l'Université de Franckfort. Il est écrit en C++, et propose un compilateur d'IDL pour ce langage (ce qui est une des raisons également du choix de MICO), il utilise l'API standard UNIX et des librairies non propriétaires.

La version 2.3.11 de MICO inclue les fonctionnalités suivantes :

- Mapping IDL vers C++
- Invocation Dynamique d'Interface (Dynamic Invocation Interface)
- Invocation Statique d'Interface (Dynamic Skeleton Interface)
- Outil graphique de gestion du Répertoire d'Interfaces
- Répertoires d'Interfaces (Interface Repository)
- Protocole IIOP (prêt pour le support multi-protocoles)
- Adaptateur d'Objets Portable (Portable Object Adapter)
- Passage des objets par valeur (Object By Value)
- Composants CORBA (Corba Component Model)
- Liaison possible avec couche graphiques (Xt,Qt,Gtk,Tcl/Tk)
- Gestion dynamique des Any (Dynamic Any)
- Intercepteurs Portables (Portable Interceptors)
- Support SSL
- Support d'invocation imbriquées

- Multi-tâches
- Nombreux Services CORBA :
  - Service de Nommage (Interoperable Naming service)
  - Service d'Echange (Trading service)
  - Service d'Evénements (Event service)
  - Service de Relations (Relationship service)
  - Service de Propriété (Property service)
  - Service de Temps (Time service)
  - Service de Sécurité (Security service)

5.4.2 Serveur

L'implémentation est faite en C++ pour la partie serveur. Une implémentation des blocs 1,2 et 5 de TASE.2 est actuellement disponible. Les parties adhérentes au protocole de communication ont été soigneusement isolées au sein de classes en masquant les spécificités. Ceci facilitera le portage vers d'autres implémentations de CORBA ou même vers d'autres bus logiciels à objets.

5.4.3 Client

L'implémentation est faite en C++ pour la partie client. Le client se voit adjoindre une interface graphique afin de pouvoir faire des démonstrations (voir figure 5.6). Ce prototype a donné lieu à une présentation très concluante à des chercheurs faisant partie du RTE (Réseau de Transport d'Electricité).

Pour le développement, la librairie Fltk (Fast Light Tool Kit) a été utilisée. C'est une librairie écrite en C++ et qui a le triple avantage d'être compacte, open source et gratuite, et portable.

Nous pouvons voir sur la figure 5.6 les différents éléments du protocole TASE.2, organisés de manière hiérarchique sur la gauche de la fenêtre. Le client peut gérer un nombre quelconque d'associations avec un même serveur ou des serveurs différents. Dans chaque association, on retrouve les variables nommées, les ensembles de transfert ...

5.4.4 Disponibilité

Les sources documentés sont disponibles sur Savannah [40] pour utilisation gratuite et non limitée.

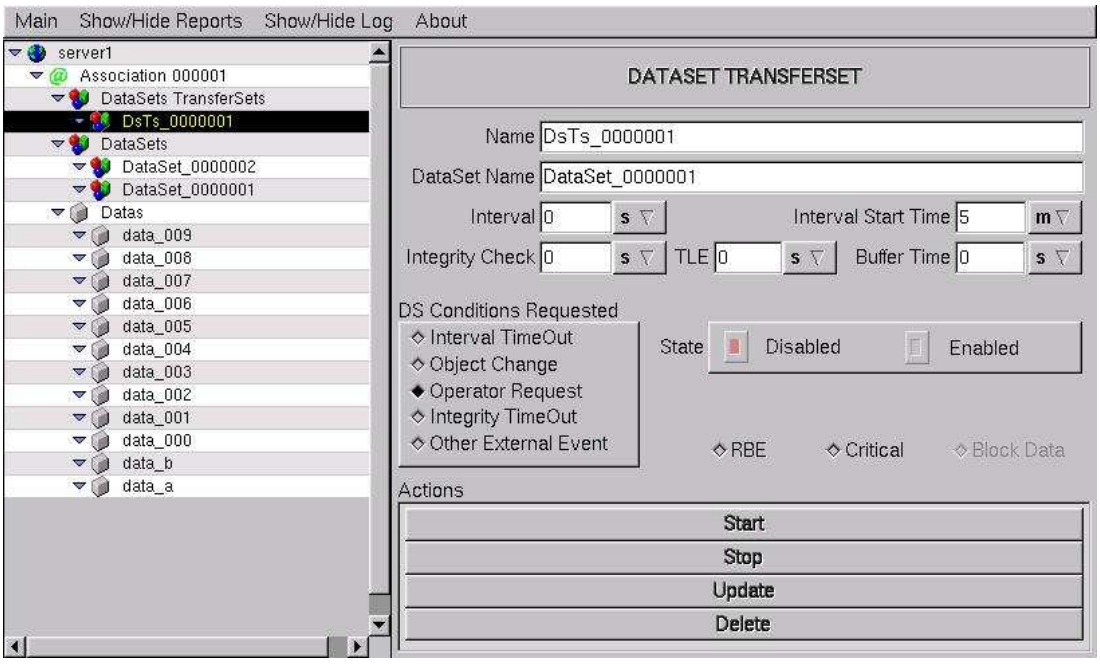

FIG. 5.6: Client TASE.2

5.5 Conclusion

Ce prototype permet de valider notre approche, à savoir que les applications de contrôle commande dans la production et la distribution d'énergie et les besoins en termes de médiation entre centres de production peuvent utiliser des solutions bâties à base d'objets distribués. L'implémentation est facilement transposable à un autre ORB objet. Une partie limitée de tests fonctionnels a permis de valider que les fonctions apportées sont strictement équivalentes à celles du protocole TASE.2, ce qui est normal puisque nous avons reproduit les mêmes interfaces et les mêmes structures de données.

De plus, la solution proposée est implémentable et donc utilisable sur une vaste étendue d'architectures, et en particulier à tous les niveaux du modèle CIM (cf. 2.3.2) grâce aux supports des bus logiciels CORBA par les systèmes d'exploitation temps réel, et grâce aux fonctions réseau bâties au-dessus de CORBA. L"apport de mécanismes de sécurité proposés par CORBA en standard en plus des services de transfert de données permet également d'atteindre les besoins des architectures de médiation. Nous avons donc une solution qui peut se déployer verticalement de bas en haut et qui permettrait de simplifier grandement la remontée des informations vers les niveaux hauts de l'architecture et entre les différents intervenants (producteurs, distributeurs, médiateurs).

De manière plus secondaire, il est intéressant de noter que le développement d'une solution complètement opérationnelle en quelques mois permet d'argumen-

ter que les coûts d'une telle solution seraient probablement inférieurs à ceux d'une solution propriétaire, tout en bénéficiant des avantages inhérents au monde du logiciel libre. Ceci est bien entendu lié au choix de l'ORB MICO.

Chapitre 6

TASE.2 Composants

Sommaire

6.1 Introduction

Pour évaluer les avantages et inconvénients d'une architecture orientée composants par rapport à une architecture plus classique orientée client/serveur, nous

avons développé un prototype basé sur les composants CORBA issus de la spécification CORBA 3.0 [35].

Le prototype offre les mêmes fonctionnalités que celles de notre précédent prototype basé sur CORBA 2.x : échange de données périodique (Bloc fonctionnel 1 de TASE.2), échange de données sur condition (Bloc fonctionnel 2 de TASE.2) et contrôle des DEVICE (Bloc fonctionnel 5 de TASE.2). De plus, nous avons utilisé la même implémentation de CORBA,MICO [41], de façon à pouvoir faire des comparaisons de performances entre les deux prototypes.

6.2 Notion de Composant

Plusieurs définitions peuvent être trouvées pour le composant, nous pouvons ici en citer quelques-unes :

- ➡ Un composant logiciel est une unité de composition dotée d'interfaces spécifiées. Un composant logiciel peut être déployé indépendamment et sujet à une composition par une tierce entité .
- ➡ Un composant est un ensemble de composants atomiques déployés. Un composant atomique est un module et un ensemble de ressources. Un module est un ensemble de classes. Une ressource est une collection d'éléments typés.
- ➡ Les objets forment les composants et les composants forment les applications.

De ces définitions, nous retenons qu'un composant est l'élément de base dans la construction d'applications, c'est sa finalité et il a été défini en tant que tel. C'est donc une brique logicielle autonome configurable appelée à êêtre associée à d'autres afin d'élaborer une application "par assemblage". Le composant offre pour ce faire des interfaces spécifiées pour s'interconnecter et pour se configurer, il est facilement composable, configurable et "déployable".

Un composant vérifie donc un certain nombre de propriétés : il doit êêtre possible de le configurer et d'accéder facilement à son état public, il doit permettre l'introspection. Son cycle de vie doit êêtre géré de façon transparente pour ses utilisateurs, en particulier les notions de création/activation/désactivation et destruction sont prises en charge par l'infrastructure. Il peut envoyer/recevoir des évènements qui permettent de réagir à des stimuli extérieurs et d'en générer. Il n'est pas nécessaire de spécifier toutes les propriétés non fonctionnelles du composant (sécurité, transaction, persistance ...) car elles sont prises en charge par l'infrastructure directement et offertes aux concepteurs de composants. Des outils graphiques sont disponibles pour faire de l'assemblage de composants.

Un composant, de par sa grande réutilisabilité, va donc permettre :

① de simplifier la conception des applications distribuées,

② de concevoir par assemblage, dans des outils graphiques ergonomiques,

③ donc d'améliorer l'efficacité du cycle de développement logiciel,

④ de masquer la complexité sous-jacente de certains concepts,

⑤ d'améliorer la qualité du logiciel produit,

⑥ d'augmenter dhrastiquement la réutilisabilité logicielle (voir figure 6.1).

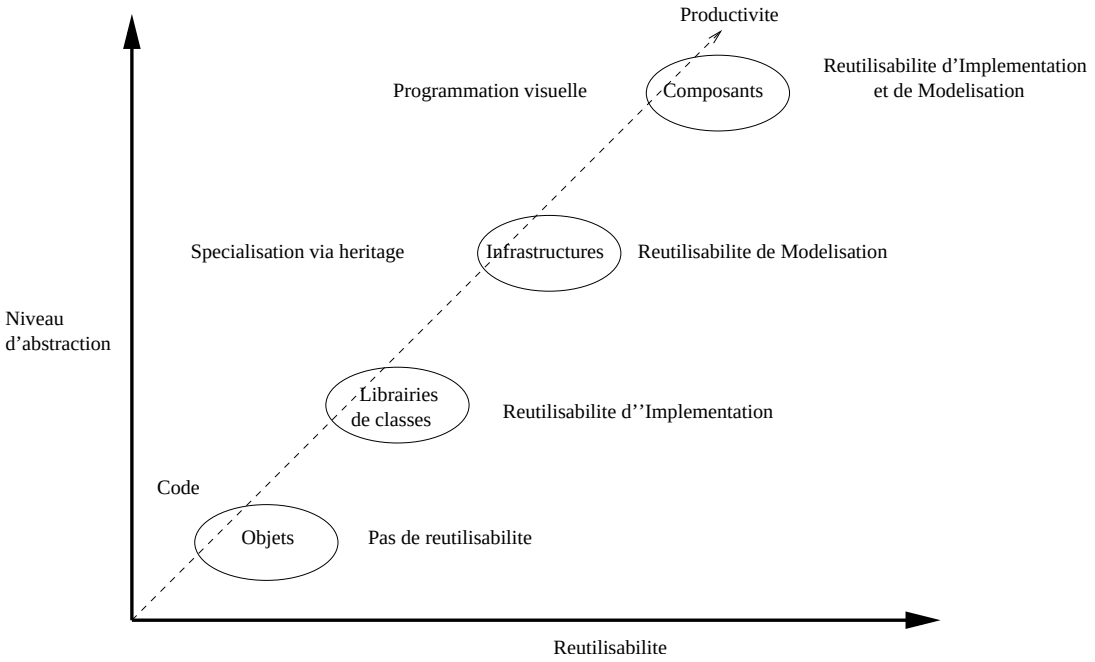

FIG. 6.1: La réutilisabilité des composants

Les CCM (Corba Component Model), les EJB (Enterprise Java Beans), COM/DCOM et dot Net de Microsoft sont des exemples d'architectures à composants.

6.3 Le modèle de Composant CORBA (CCM)

6.3.1 Introduction

Cette présentation des composants CORBA est basée sur un rapport technique de F. Pilhofer [37] et sur un didacticiel émanant de l'OMG [18]. Le modèle de composants CORBA [34] est la nouveauté majeure de la spécification 3.0 du standard CORBA [35]. Les composants CORBA sont un moyen d'arriver à une réelle intégration aisée et automatique de parties logicielles. Cette nouvelle vision des choses affecte à la fois le côté client (comment les composants sont utilisés) et le côté serveur (comment les composants sont développés). Deux idées principales ont conduit au modèle de composant CORBA : le modèle de composant lui-même et le modèle de conteneur de composants. Le standard se concentre sur la façon

dont les composants interagissent entre eux plutôt que de décrire les composants eux-mêmes.

6.3.2 Le Composant CCM

Un composant dans CORBA, est essentiellement défini via les interactions qu'il a avec les autres composants. Par exemple, un composant base de données peut être décrit comme un fournisseur d'une interface de lecture/écriture et comme un producteur d'événements Écriture pour prévenir les composants intéressés qu'une donnée vient d'être modifiée. Le composant Base de Données lui-même peut utiliser les opérations lecture/écriture d'un composant Système de fichiers sous-jacent. De cette manière, il est très facile de faire évoluer un composant à partir du moment où les interfaces fournies/utilisées et les événements produits/consommés ne sont pas modifiés).

6.3.3 Les Connections du Composant

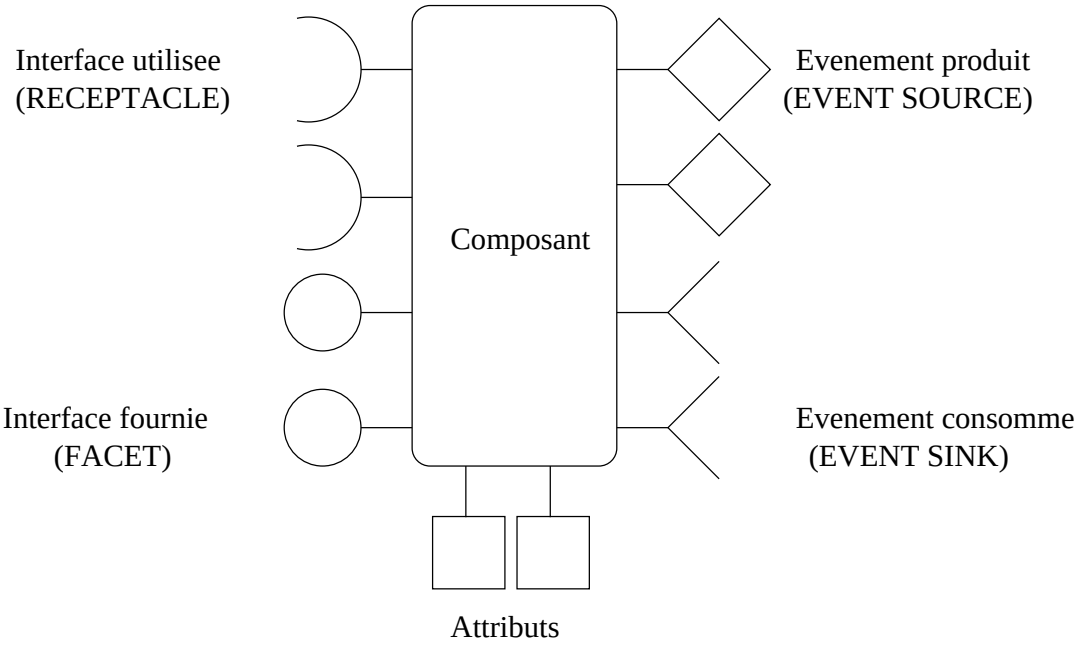

FIG. 6.2: Le composant CORBA

La figure 6.2 montre la vue externe d'un composant CORBA, i.e. ce qu'il exporte/requiert vers/de l'extérieur. Un composant peut avoir des attributs, supporter des interfaces (non montré sur la figure) et avoir des connexions. Les attributs et le support d'interfaces sont directement reliées au modèle CORBA standard 2.x. Les connexions, elles, décrivent complètement le connectivité d'un

composant, à savoir la manière dont il va s'interconnecter avec d'autres composants. L'étape suivante dans le processus de réalisation d'un logiciel est d'utiliser une interface utilisateur conviviale pour aider à cette interconnexion de composants.

Nous listons maintenant les différents types de connexions (aussi appelées ports) disponibles dans la norme CORBA 3.0 et les mots clés associés en IDL3 (IDL3 dénote l'évolution de la spécification de ce langage dans le standard CORBA 3.0) :

- **Facet** correspond à une interface fournie (mot-clé IDL3 : *provides*),
- **Receptacle** correspond à une interface utilisée (mot-clé IDL3 :*uses* et *uses multiple*), [1]
- **Event Source** correspond à un événement produit (mot-clé IDL3 : *publishes* et *emits*) [2]
- **Event Sink** correspond à un événement consommé (mot-clé IDL3 : *consumes*)

Il est important de noter que toutes ces nouvelles fonctionnalités offertes dans CORBA 3.0 sont complètement construites au dessus de l'architecture CORBA standard (2.x). L'avantage principal de ceci étant de pouvoir facilement intégrer des composants dans des architectures CORBA client/ serveur classiques et vice-versa.

6.3.4 La Configuration du Composant

Le modèle de composant CORBA fournit des mécanismes pour gérer le concept de configurabilité du composant. L'expérience a montré que construire des composants réutilisables implique des compromis entre le niveau de précision des fonctionnalités apportées et la souplesse nécessaire pour atteindre cette réutilisabilité entre plusieurs applications variées.

Les contraintes quant au packaging du composant excluent la possibilité de configurer le comportement du composant en changeant son implémentation ou (dans la plupart des cas) par dérivation. Le modèle se concentre donc sur l'extension et la personnalisation en utilisant la délégation (par exemple, via des dépendances envers d'autres composants exprimées avec le mot-clé *use*) et la configuration.

Le modèle de configuration a été conçu pour apporter les capacités suivantes :

- La capacité de définir des attributs dans le type du Composant, qui permettent d'établir la configuration d'une instance d'un Composant. Les attributs des composants sont supposés être utilisés durant la phase d'initia-

[1] le mot clé *multiple* est utilisé si plus d'une instance de cette interface doit êêtre connectée.

[2] Une distinction est faite si un événement est à destination d'un et un seul consommateur (*emits*) ou au contraire de plusieurs (*publishes*)

lisation de l'instance du composant afin de spécialiser son comportement. Le modèle de composant ne contraint pas la visibilité et l'utilisation de ces attributs. Cependant, il est généralement admis qu'ils ne sont pas utilisés par les utilisateurs du composant après la phase d'initialisation. Leur usage est plutôt réservé aux fabriques de composants et aux outils de déploiement.

- La capacité de définir une configuration dans un environnement autre que celui du déploiement (par exemple un outil d'assemblage) et de stocker cette configuration dans le package du composant ou de l'assemblage des composants pour êêtre utilisée au moment du déploiement.
- La capacité de définir une telle configuration sans avoir à instancier le composant.
- La capacité d'associer une configuration prédéfinie avec une fabrique de composants, de façon à ce que les composants créés par cette fabrique soient initialisés avec cette configuration.
- La capacité à permettre le fonctionnement d'outils visuels interactifs pour définir ces configurations (un exemple d'un tel outil, celui fourni avec MICO, est donné dans la figure 6.3). En particulier, les développeurs de composants sont autorisés à fournir un gestionnaire de configurations associé à l'implémentation du composant. Le gestionnaire de configurations expose une interface pour aider à la saisie, vérification ... des configurations.

6.3.5 Le Conteneur de Composants

Le conteneur est l'environnement d'exécution du composant du côté serveur. Dans les applications CORBA 2.x, de larges portions de code devaient être réécrites à chaque fois : gestion des références d'objets ou enregistrement auprès du service de nommage sont des exemples de ces tâches récurrentes. Le conteneur se charge de ces tâches automatiquement et de manière transparente. En conséquence, un composant CORBA et son implémentation ne sont jamais en contact avec l'ORB. En fait, le besoin pour le développeur d'un composant de connaître les détails du bus logiciel CORBA tend à être nul. Le but final étant d'intégrer au niveau du conteneur de plus en plus de services tels que la sécurité, les transactions ou bien encore la persistance.

6.3.6 La Fabrique du Composant

Et finalement, chaque type de composant peut être géré par une ou plusieurs fabriques (mais une fabrique ne peut gérer qu'un type de composant). La fabrique est le nom du container en CORBA (on peut trouver des concepts similaires dans le monde des Java Beans).

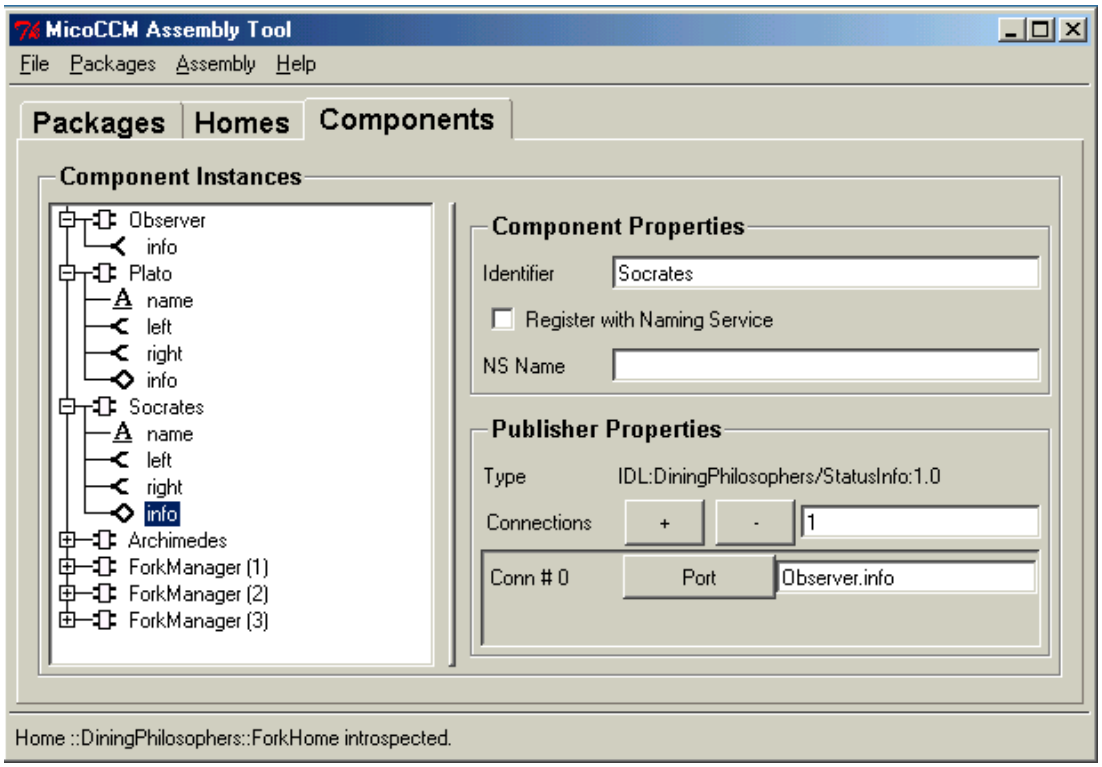

FIG. 6.3: L'outil d'assemblage et de configuration de MICO

Une fabrique est utilisée pour gérer les instances des composants. Par exemple, la fabrique est habituellement enregistrée auprès du service de nommage (annuaire CORBA) et fournit aux autres composants des services tels que création, activation, passivation et destruction du type de composant qu'elle gère. Du point de vue du composant lui-même, la Fabrique fournit toutes les opérations nécessaires à la communication avec le monde extérieur (i.e. autres composants), comme l'envoi et la réception d'événements, l'accès aux interfaces etc ...

6.3.7 Packaging & Déploiement du Composant

Les implémentations de Composants peuvent être packagées et déployées. Un package de Composant CORBA contient une ou plusieurs implémentations d'un composant (e.g., plusieurs versions). Il peut être installé seul sur un ordinateur ou groupé avec d'autres composants pour former un assemblage. Un assemblage de Composants est un groupe de composants interconnectés et représenté par un package d'assemblage.

En général, un package est formé de un ou plusieurs descripteurs et d'un ensemble de fichiers. Les descripteurs servent á noter les différentes caractéristiques du package et pointent vers ses différents fichiers. Les fichiers formant un package, y compris les descripteurs, peuvent être groupés dans une archive ou stockés séparément. Quand ils sont stockés séparément, le descripteur contient des pointeurs vers chacun des fichiers.

Le package de composant CORBA est une spécialisation d'un package logiciel au sens général. Le modèle utilisé dans CORBA a été inspiré à l'origine par la "Description de Logiciel Ouvert (Open Software Description ou OSD) du W3C. La description OSD est un vocabulaire XML pour décrire les packages logiciels et leurs dépendances. Ce modèle a été étendu légèrement, sans en perdre la généralité, pour supporter le packaging des Composants.

Le descripteur d'un assemblage de composants spécifie les composants qu'il contient, les contraintes de partitionnement et les connexions. Une connexion peut être établie entre un composant qui fournit une interface I et un autre qui utilise cette interface, et de la même manière entre un composant qui émet un événement et un autre qui le consomme.

Les packages de composants et d'assemblages de composants sont fournis en entrée d'un outil de déploiement. Un outil de déploiement déploie des composants isolés ainsi que des assemblages de composants sur un site d'installation, habituellement un ensemble de machines sur un réseau. L'utilisateur de cet outil de déploiement spécifie où va être installé chaque composant. Les composants d'un même assemblage peuvent être installés sur une même machine ou dispersés sur un réseau. A partir des informations du descripteur d'assemblage et des entrées de l'utilisateur, l'outil de déploiement installe et active les fabriques de composants et les instances. Il configure les propriétés des composants et les interconnecte, via les interfaces et les événements, comme indiqué dans le descripteur d'assemblage.

6.4 OpenTAZ-CCM : Un prototype TASE.2 à base de Composants CORBA

6.4.1 Modélisation

OpenTAZ-CCM est notre nouveau prototype, proposant les mêmes services que le protocole TASE.2 original mais utilisant les composants CORBA. Les interfaces entre le client et le serveur sont semblables mais les interactions et la modélisation sont assez différentes. La figure 6.4 donne un aperçu de l'architecture d'OpenTAZ-CCM, qui est détaillée par la suite.

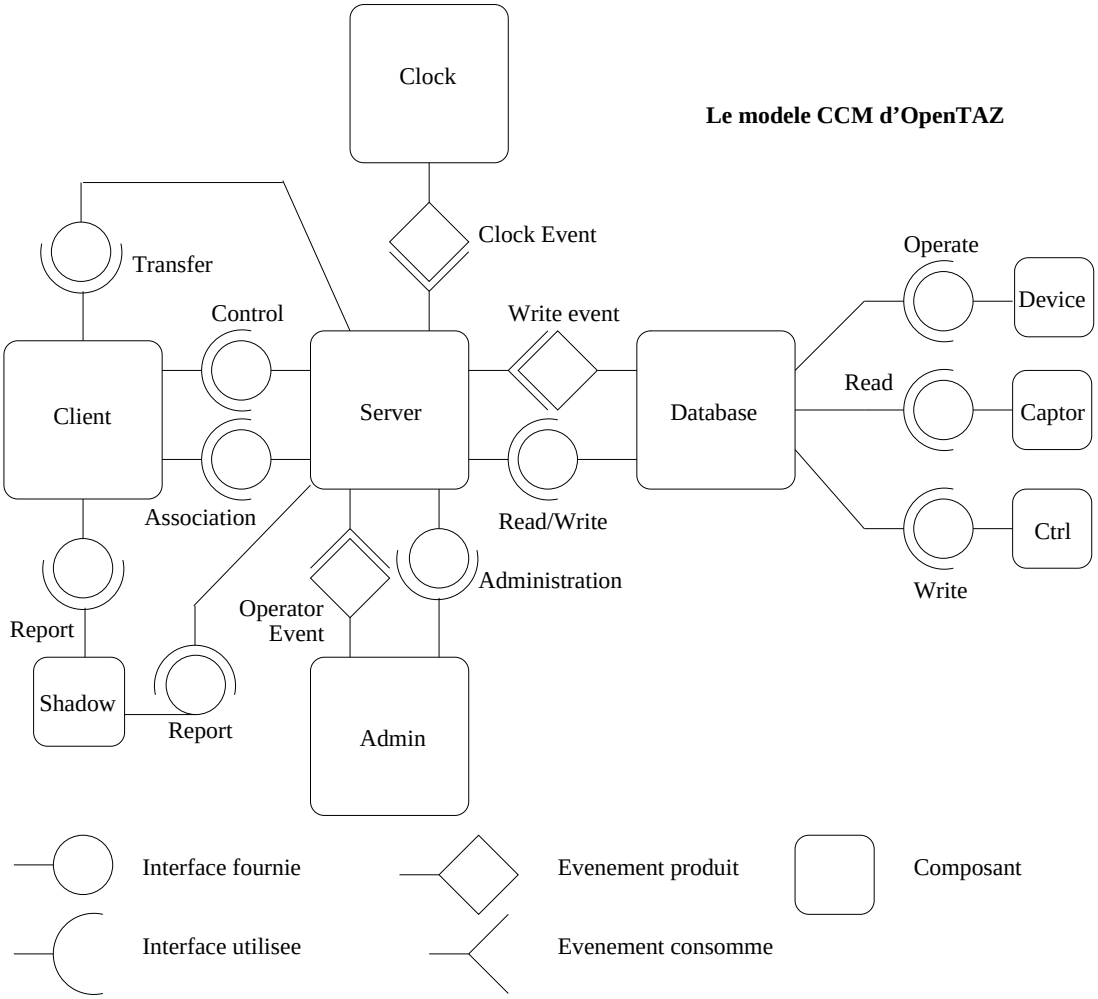

FIG. 6.4: L'architecture de composants d'OpenTAZ-CCM

Cette modélisation a été dictée par deux grandes idées :

➡ La modélisation doit directement être issue des spécificités fonctionnelles du protocole TASE.2. Nous nous somme attachés à modéliser aussi fidèlement

que possible les différents objets décrits dans la norme TASE.2, que ce soit au niveau des composants ou au niveau intra-composants, au niveau objet. Ceci nous permet de pouvoir définir facilement les interfaces directement à partir des messages définis dans TASE.2.

➡ La modélisation doit permettre la réutilisabilité, évidemment puisque c'est un des attendus de l'utilisation des composants. Mais plus que cela, notre modélisation doit permettre l'utilisation de notre prototype dans le cadre de TASE.2 mais aussi dans le cadre de la supervision du transport d'énergie. Et ce, bien entendu, avec le moins de transformations possibles.

La première idée amène une grande partie de la modélisation et est assez aisée à mettre en oeuvre car même si le protocole TASE.2 (comme MMS) est orienté message, les objets mis en jeu sont clairement identifiés.

La deuxième idée nous amène à découpler le plus possible les aspects "couche basse" des aspects fonctionnels de l'application. Ainsi, les éléments bas niveau identifiés dans le protocole TASE.2 et pilotés/surveillés, à savoir les contrôleurs, les capteurs et les actionneurs, sont modélisés sous forme de composants à part entière et séparés des fonctions d'archivage/gestion des avalanches/remontée des alarmes qui sont prises en charge par le composant base de données. Le composant base de données encapsule alors tous les mécanismes dont on veut pouvoir se resservir au niveau transport d'énergie.

De plus, les interfaces de ces contrôleurs, les capteurs et actionneurs sont identifiées et rendues plus générales pour pouvoir être exportées par d'autres composants dans d'autres contextes.

De la même manière, le composant Horloge, non identifié comme tel dans le protocole TASE.2 est ici isolé alors que l'on aurait pu l'intégrer dans le Serveur. Cependant, dans le contexte du transport d'énergie, les sites sont distants de dizaines voire de centaines de kilomètres et il est souhaitable de se référer à une horloge commune (via GPS par exemple). Ici il suffit de remplacer notre composant simple qui se contente d'envoyer l'heure système par un composant gérant un composant matériel amenant la fonction adéquate.

6.4.2 Rôle des composants

Nous décrivons maintenant chacun des composants de cette architecture, en terme de connectivité (ports) et de rôle dans l'atteinte des fonctionnalités de TASE.2.

➡ **Clock** (Horloge) est un composant très simple, avec un seul port qui est la source des événements de type ClockEvent. Ce composant envoie des événements à une cadence donnée (choisie à la création du composant). Ce composant est utilisé pour donner une référence de temps au serveur mais peut-être aussi à d'autres composants dans le futur.

- **Device, Captor, Controller** (Outil, Capteur, Controlleur) sont des composants directement issus des périphériques matériels. Ils sont très utiles car ils permettent de masquer la complexité et le détail de ces périphériques et n'exporter que les interfaces utiles dans notre cas. De cette manière, le composant Base de Données (cf. ci-après) peut gérer différents types de périphériques sans être modifiés, pourvus qu'ils exportent l'interface adéquate.
- **Database** (Base de Données) est un composant base de données basique avec des receptacles pour lui connecter des équipements à gérer (capteurs etc ...). Il peut aussi gérer des données en mémoire. Il exporte une simple interface qui permet aux autres composants (ici le serveur) de faire des lectures et/ou écritures sur les données gérées. De plus ce composant produit un événement de type Write à chaque modification d'une donnée, soit par un composant, soit car la donnée correspond par exemple à un processus physique contrôlé. Les composants intéressés par les modifications de telle ou telle donnée n'ont plus qu'à s'abonner à cet évènement. L'évènement Write a été introduit afin de pouvoir reproduire parfaitement une des fonctionnalités TASE.2 qui permet l'envoi d'un rapport sur modification de valeur. De plus, cela nous permet d'uniformiser l'envoi des rapports puisqu'ils le seront toujours sur réception d'événement, que ce soit un événement Write (modification), Clock (périodique) ou Operator (opérateur).
- **Admin** (Administration) est un composant qui permet une reconfiguration dynamique du serveur (hors TASE.2) telle que : ajout d'un client, modification des permissions des clients existant, arrêt du serveur. C'est aussi via ce composant que l'opérateur peut forcer l'envoi d'un rapport de données, en générant l'événement Operator qui est consommé par le Serveur. Cette dernière possibilité provient directement de la spécification TASE.2.
- **Server** (Serveur) est principalement le moyen de contrôler l'accès aux données et de générer les rapports. Il gère les associations avec les clients, vérifie les droits d'accès de chacun avant l'ouverture d'une association. Il peut gérer plusieurs associations avec un même client (peut-être avec une qualité de service différente par exemple). C'est lui qui génère les rapports de données et les envoie aux clients en fonction de leurs demandes et des événements qu'il reçoit.
- **Client** (Client) est un composant qui utilise l'interface du Serveur pour ouvrir/fermer une association et les autres interfaces pour pouvoir effectuer les opérations définies dans le standard TASE.2. Il exporte une interface pour pouvoir recevoir les rapports de données envoyés par le Serveur et qu'il a demandé à recevoir.
- **Shadow** (Cache) est complètement nouveau et n'a pas d'équivalent dans notre précédent prototype basé sur CORBA 2.x. Il a été ajouté pour mettre en valeur l'avantage prototype basé sur les composants. Ce composant est

très simple en terme de connexions puisqu'il exporte une seule interface, la même que celle qu'il utilise. Il peut être vu comme un proxy mais il ne se contente évidemment pas de renvoyer les rapports de données aux clients. Il est possible d'y inclure des politiques plus ou moins complexes pour gérer les rapports de données. Par exemple, il est possible d'implémenter un protocole de cohérence des données (strict, causal, temporel etc ...) qui est assuré par les composants Cache qui interagissent entre eux. Il est aussi possible d'utiliser ce composant pour implanter une politique de sécurité avec intégrité, authentification etc...).

6.4.3 Prototype

Un prototype a été développé, opérationnel et qui a fait l'objet de présentation [68].

6.4.3.1 MICO-CCM

Le projet MICO-CCM, dont le but est d'ajouter à MICO le support des composants, a démarré en octobre 2000. Sont but est de fournir, à l'instar de MICO pour la norme CORBA 2.3, une implémentation de référence du modèle de composant CORBA, la plus grande nouveauté de la version 3.0 de CORBA.

Le projet MICO-CCM supporte les fonctionnalités suivantes (et respect des fonctionnalités définies comme objectifs, à savoir le support complet des composants dits service et session) :

- Support du langage IDL 3 (mots-clés *component*, *home* et autres) ont été ajoutés au compilateur d'IDL de MICO et au Répertoires d'Interfaces (Interface Repository).
- Support de tous les types de connexions : facettes (mot-clé *provide*), receptacles d'interfaces simple et multiple (mots-clés *uses* et textCodeuses multiple, sources d'événements (mots-clés *emit* et *publishes*) et consommateurs d'événements (mot-clé *consumes*).
- Support des conteneurs Service et Session.
- Support des implémentations monolithiques et basée sur les locateurs.
- Les composants peuvent être construits en tant qu'éxecutable autonome ou dans des librairies partagées.
- Des outils pour installer, contenir, charger et gérer les implémentations des composants.
- Un outil graphique pour l'assemblage et le déploiement des assemblages de composants (les archives contenant de multiples composants et les informations de déploiement peuvent être déployés en une seule étape).

Toutes ces fonctionnalités sont conformes aux spécifications officielles de l'OMG.

6.4.3.2 Implémentation

Le prototype OpenTAZ-CCM est directement une implémentation des composants décrits en 6.4.1. Tous les composants sont implémentés hormis les contrôleurs, capteurs et actionneurs. Les composants de Cache et d'Administration sont basiques et ne fournissent pour l'instant pas de fonctionnalités complexes telles que la gestion de la cohérence des données.

La figure 6.5 est la description complète du composant Serveur dans le langage de spécification IDL3, tel que défini dans la spécification CORBA 3.0 CCM.

```
component Server
{
  provides AssociationInterface iAssociation;
  provides ControlInterface iControl;
  provides TransferInterface iTransfer;
  provides AdministrationInterface iAdministration;

  uses DBInterface iDB;
  uses ReportInterface iReport;

  consumes ClockEvent eClock ;
  consumes OperatorEvent eOperator ;
  consumes WriteEvent eWrite ;
};
```

FIG. 6.5: La description IDL3 du composant Serveur.

Ce prototype a été implémenté en utilisant MICO [41], une des rares implémentations des composants CORBA, et de plus en OpenSource. Il y en a d'autres tels que OpenCCM [30] par exemple mais MICO fournit la liaison C++ dont nous avions besoin afin de réutiliser au maximum le code de notre précédent prototype. De plus, l'utilisation du même ORB facilitera les comparaisons de ces deux prototypes en termes de performances.

6.5 Tests de Performance

6.5.1 Introduction

Nous testons les performances relatives des deux prototypes, OpenTAZ version objet, présenté au chapitre précédent et la version composant, décrite juste auparavant dans ce chapitre.

Les tests effectués correspondent à des mesures brutes des invocations de temps aller/retour pour chaque fonctionnalité. Pour chaque méthode disponible côté client, nous calculons une valeur moyenne. Les tests sont conduits dans les mêmes conditions matérielles et logicielles pour les deux prototypes. Dans la prochaine partie, dédiée à l'analyse des résultats, nous nous focalisons sur les

différences pouvant apparaître pour un même appel avec la même sémantique. Les prototypes utilisent la même version de MICO, à savoir la 2.3.8.

Deux séries de mesures sont présentées. La première correspond à des tests locaux, le serveur et le client (et tous les composants dans le cas de la version CCM) sont sur une seule et même machine. La deuxième correspond à des tests en réseau, avec le client communiquant avec le serveur via un réseau local (LAN). Dans ce dernier cas, les tests sont effectués de manière à minimiser l'impact du trafic réseau.

La plateforme de tests est composée de deux machines à base de processeurs Intel-2.2 Ghz, avec le système d'exploitation LinuX RedHat 7.3 (Kernel 2.2.18) et connectés via un switch 100mbs, relié lui-même à internet via un proxy.

6.5.2 Résultats

Les résultats sont résumés dans deux tableaux, un pour les tests en local (voir figure 6.6) et l'autre pour les tests en réseau (voir figure 6.7). Chaque test est réalisé avec les deux prototypes. Les tests pour les méthodes Get/Set (récupération et positionnement d'une donnée, ou bien encore lecture/écriture) sont de deux sortes, les premiers mettent en jeu des données simples (S) et les deuxièmes des données étendues (X). Les données étendues ajoutent aux données simples des informations complémentaires comme une estampille temporelle ou bien encore un drapeau de qualité de la donnée.

6.5.2.1 Local

On peut tout d'abord noter que beaucoup de résultats sont identiques pour les deux prototypes. Ceci est du au fait que beaucoup des interactions dans TASE.2 n'impliquent que le client et le serveur (Récupération des noms des données (GetDataNames), Operations sur les Ensembles de données (xxxDataSet) et Ensembles de transfert (xxxTransferSet), Fermeture d'une association (Conclude)). En conséquence, chacun de ces tests conduit logiquement au même résultat.

Au contraire, les résultats des quatre tests mettant en jeu le composant Base de Données (Lecture/Ecriture de données (Get/Set DataValue)) montrent des différences notables. Ces tests font en effet intervenir un objet CORBA de plus dans la version CCM, les résultats y sont donc plus grands logiquement.

Les résultats devraient être identiques pour l'opération d'établissement d'une association (Associate). Nous pouvons seulement supposer que l'implémentation fournie par les Fabriques aux composants de services tels que enregistrement, construction des interfaces, publication et autres sont meilleures que les nôtres. Cela plaide en faveur de l'utilisation de plus en plus de services fournis par les Fabriques et donc de tirer bénéfice d'un code très utilisé donc mieux testé, optimisé etc. . ..

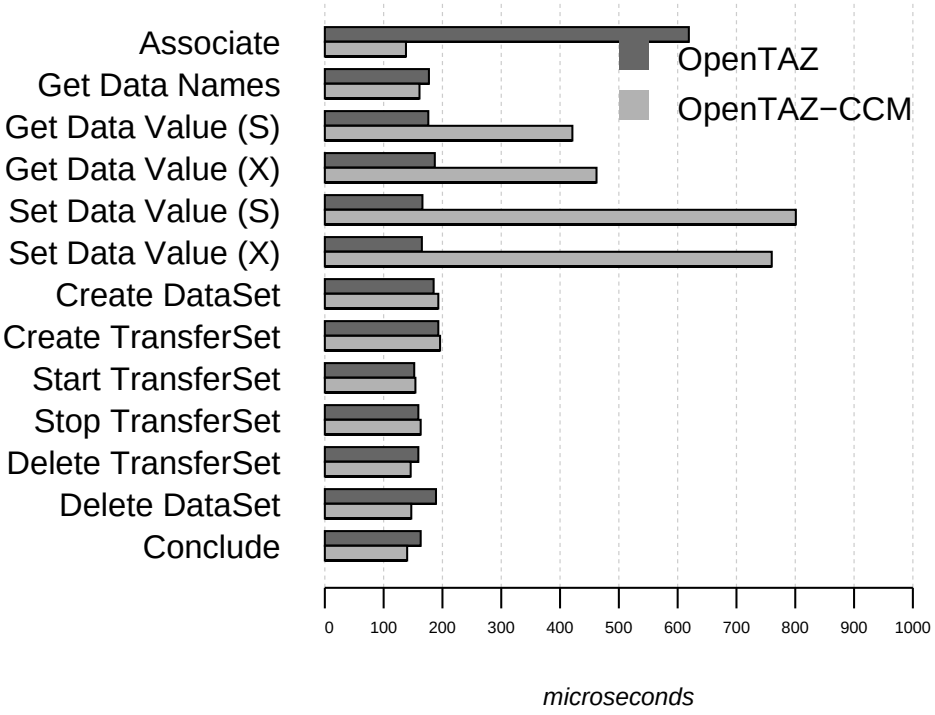

FIG. 6.6: Résultats des tests OpenTAZ en local

6.5.2.2 Réseau

La seule nouveauté ici, comparé aux résultats de tests en local, est que la différence entre les deux prototypes est légèrement moindre. C'est intéressant car le but d'une telle architecture est évidemment d'être massivement distribuée. Il faudrait faire des tests à plus grande échelle pour le valider, mais on peut déjà voir ici que les baisses de performances ne sont pas trop importantes par rapport à une version client/serveur plus classique.

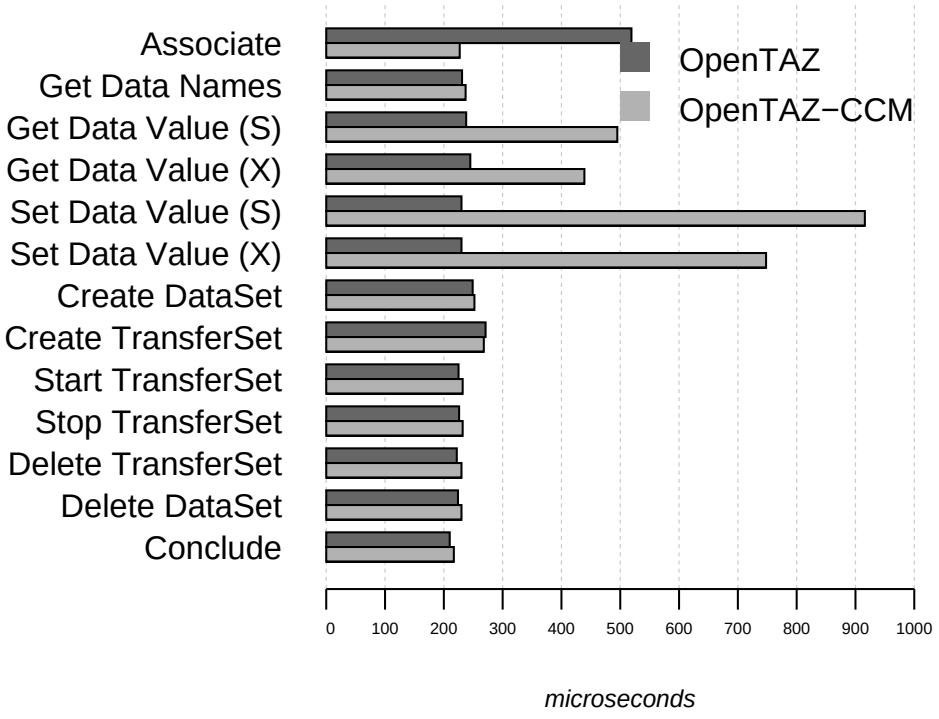

FIG. 6.7: Résultats des tests OpenTAZ en réseau

6.6 Conclusion

Ce nouveau prototype a permis de démontrer le réel apport des architectures à composants pour notre problématique, par exemple la mise en évidence du composant Shadow qui nous permet d'implanter des politiques de cache variables très facilement sans toucher aux autres composants. Dans le bilan, nous revenons sur les avantages et les inconvénients de cette solution. Dans la partie suivante, nous essayons d'ouvrir la réflexion vers des problématiques plus temps réels et dans quelle mesure les technologies existantes permettent d'y répondre.

6.6.1 Bilan

Les composants ont rendu la modélisation de notre nouveau prototype beaucoup plus simple. Deux prototypes ayant les mêmes fonctionnalités ont été développés, l'un avec, et l'autre sans les composants, on peut comparer et dire avec certitude que le développement avec les composants est plus rapide et plus efficace que dans un monde client/serveur. De plus, pour ce second prototype, une connaissance poussée de CORBA devient totalement inutile. Beaucoup de mécanismes tels que l'enregistrement auprès du service de nommage, le démarrage des serveurs, l'envoi et la réception d'évènements sont très faciles à utiliser, grâce à la Fabrique. Elle masque au développeur du composant toute la complexité CORBA, et de fait, il se retrouve en train de programmer un objet simple. Enfin, rappelons-nous que de plus en plus de services seront ajoutés au niveau de la fabrique (persistance, transactions, sécurité etc ...) et pourront donc être utilisés tels quels pour les composants.

Notre nouveau prototype bénéficie des avantages de l'approche orientée composants. La réutilisabilité n'est plus un vain mot, et surtout, elle est facile à mettre en oeuvre. Ainsi, notre composant Base De Données est générique, aucune supposition n'a été faite dans son implémentation quant à ses utilisateurs potentiels, il se contente de fournir des méthodes de lecture/écriture de données, et de fournir un service d'événements lorsqu'une donnée est modifiée. De même les composants gérés par la base de données ne sont en rien fixés et peuvent évoluer. C'est la base d'un composant Base de Données Temps Réel simple mais fonctionnel qui peut être utilisé sans aucune modification, nous le pensons, dans beaucoup d'autres environnements (jeux répartis par exemple). Notre composant basique Horloge a la même propriété.

De plus, les résultats des tests de performance montrent que les bénéfices liés aux composants ne sont pas trop pénalisés par les temps d'accès pour des objets distants. En ce qui concerne les tests locaux, il faudrait relancer les tests en tirant parti du fait que plusieurs composants peuvent s'exécuter au sein d'un même processus afin d'améliorer les performances. Toutes les communications inter-composants étant alors locales au processus, nous pensons que les performances devraient être meilleures qu'avec un client/serveur classique qui doit traverser la pile IP.

Finalement, la modélisation de notre nouveau prototype a décomposé notre premier serveur monolithique en plusieurs composants. L'implémentation résultante amène plus de flexibilité et, nous l'espérons, plus de réutilisabilité. On peut noter également que le déploiement des composants peut être facilité par l'utilisation d'une interface graphique.

Le seul réel inconvénient des composants CORBA est que cette technologie est encore récente, et peu supportée par les fournisseurs d'ORB commerciaux (par exemple la dernière mouture de Java qui intègre une couche CORBA fonctionnelle, ne gère pas les CCM). De plus les rares implémentations existantes

sont peu utilisées, donc peu testées et sans support ou presque. La jeunesse des composants CORBA fait aussi que l'un des intérêts majeurs des composants, à savoir la réutilisabilité, est nulle puisqu'aucun composant n'est disponible. On est loin des larges bibliothèques de composants COM disponibles un peu partout !

6.6.2 Les composants et le temps réel

Notre solution, basée sur CORBA, ne peut pas prétendre adresser des problématiques temps réel et les applications visées se contentent de contraintes temps réel mou qui sont accessibles avec des réseaux physiques sous-jacents adéquats. Il est intéressant de se demander si des technologies à base de composants qui peuvent répondre à ces besoins existent.

Puisque nous avons choisi la technologie CORBA pour nos prototypes, il est naturel de regarder si l'OMG prévoit une solution "légère" de CORBA adéquate. En effet, l'OMG a spécifié une version de CORBA qui peut être déployée sur des systèmes temps réel embarqués, Lightweight CCM [36]. Mais cette spécification se contente de diminuer la taille d'une implémentation de CORBA en définissant les parties à enlever dans un contexte embarqué. Le seul gain est un gain en terme d'espace mémoire occupé en comparaison d'une version CORBA standard.

En ce qui concerne les autres technologies à composants du marché, les documentations disponibles, composants Java EJB [60] et composants .Net[8], ne mentionnent pas de capacités temps réel.

Chapitre 7

Conclusion & Perspectives

Sommaire

7.1 Conclusion

7.1.1 Rappel de la problématique scientifique

Le but de ce travail de thèse est l'objectification et le découpage en composants d'une messageris industrielle. Ce travail prend sa source dans l'étude de la problématique des applications de contrôle-commande d'EDF R&D d'une part, et de la supervision de la production d'autre part pour aboutir au constat qu'il existe un besoin pour une messagerie industrielle à contraintes temporelles construite à partir de composants du marché, c'est-à-dire une solution non propriétaire.

Nous avons mis en évidence le besoin d'une solution qui pourrait réconcilier les applications de contrôle-commande dans la production d'énergie (traditionnellement basées sur des solutions propriétaires à base de protocoles industriels) et les applications liées à la distribution de l'énergie (et à la supervision de la production à grande échelle). En effet, ces deux types d'application ont beaucoup en commun, comme montré auparavant, et les différences, en particulier des besoins de diffusion dans la centrale ou bien des besoins de sécuritée dans la supervision peuvent être adressées par une même solution.

7.1.2 Démarche

Pour répondre à ces besoins, nous avons d'abord évalué les contraintes de ces deux types d'applications :

- ➡ via des questionnaires adressés à des personnels EDF R&D ou des intégrateurs et une étude des architectures temps réel pour les applications de contrôle-commande pour la production,
- ➡ via l'étude fonctionnelle/étude de conformité d'un protocole de communication spécifique au domaine (TASE.2).

Il en est ressorti des besoins proches que peut adresser une solution basée sur des technologies standardisées.

7.1.3 Une première solution : prototype à objets

Un premier prototype a été développé qui a permis de démontrer la faisabilité de notre approche, il a reçu un bon accueil à la fois auprès de personnels EDF R&D travaillant dans le domaine de la production, et à la fois auprès de personnels liés au réseau de transport (RTE). Une étude de performances (qu'il faudrait certes approfondir avec des cas plus réalistes) a permis de montrer que les objectifs sont atteignables.

Finalement, ce prototype, de par ses fonctions et de par son temps de développement a permis aussi de montrer l'intérêt de notre approche du point de vue de l'ingénierie logicielle.

7.1.4 L'évolution vers les composants

Une fois l'aspect fonctionnel validé avec ce premier prototype, nous avons voulu mettre en valeur les avantages de l'approche par composants en développant un prototype fonctionnellement équivalent au premier.

Les composants nous ont fait gagner du temps dans le processus de développement global. De plus, nous avons montré qu'ils permettent une meilleure réutilisabilité et une meilleure evolutivité. Les gains en terme de déploiement sont aussi importants dans notre contexte industriel.

7.2 Perspectives

Les perspectives d'évolution pour ces travaux sont essentiellement, à mon avis, liées aux deux prototypes développés. Quatre grands axes d'évolution peuvent être identifiés :

① **Évolution des fonctionnalités du logiciel de messagerie :** Pour l'instant seule une partie des fonctionnalités du protocole TASE.2 est implémentée (transferts de données à la demande et sur événement essentiellement),

on pourrait rajouter les fonctionnalités manquantes telles que le chargement de programmes dans les capteurs/actionneurs ... On peut également envisager l'ajout de fonctionnalités autres telles que des protocoles de cohérence de données afin d'avoir une mémoire répartie partagée (ceci étant rendu possible facilement par l'architecture à base de composants). Ce dernier point nous amène à la troisième forme d'évolution : puisque nous avons de nouvelles fonctionnalités, pourquoi ne pas adresser d'autres problématiques ?

② **Évolution de l'architecture du logiciel de messagerie :** De la même façon que nous sommes passés d'un prototype à base d'objets à un prototype à base de composants, il est possible de réfléchir sur des évolutions possibles des prototypes. Une intéressante approche est par exemple la programmation par aspects qui a déjà été empruntée [73] mais on peut aussi citer une approche à base de serveur d'applications avec déploiement automatique.

③ **Évolution du domaine des applications visées :** Lorsque l'on regarde les fonctionnalités de base des prototypes déjà existant, on s'aperçoit que ces services sont finalement très génériques et que peu adhérents au domaine de la production/distribution d'énergie. Toutes les applications distribuées ayant des besoins de synchronisation, d'échanges de données sur événement, de contrôle à distance de processus ... sont de bonnes candidates à l'utilisation de notre messagerie "industrielle". On peut par exemple déjà citer l'orchestre virtuel réparti ou notre prototype à d'ores et déjà prouvé sa versatilité : il s'agit d'un orchestre distribué et contrôlé à distance avec la messagerie [71]. Nous pouvons également citer les applications de travail collaboratif (que ce soit de la CAO, du développement, de la PAO...), de jeux en réseau (envoi des états des différents joueurs à intervalles réguliers, envoi de messages particuliers quand un joueur meurt ...), de grilles de calcul ...

Finalement, le quatrième axe d'évolution n'est pas au même niveau que les autres en ce sens qu'il impacte les trois aspects déja cités. Il s'agit de migrer notre architecture vers une technologie à base de Web Services et de s'interfacer au dessus de la spécification DPWS [1] [46] qui sera intégrée dans le prochain Windows Vista (sortie prévue en 2007). Cette norme offre des fonctions intéressantes d'auto-configuration et est pressentie pour être utilisée aussi bien dans le monde informatique que domotique (voir le projet SIRENA de Schneider Electric [12]).

Dans ce contexte, notre prototype est un bon candidat de par ses fonctions, pour en faire un protocole qui permettrait de faire communiquer à la fois l'installation du particulier vers l'extérieur en termes de besoins en consommation électrique, et également les équipements entre eux dans la maison. Chaque équipement présenterait donc une interface de service et pourrait être accédé par les autres équipements : équipements audio-video, électro-ménager ...

[1] **D**evices **P**rofile for **W**eb **S**ervices

7.3 Bilan

D'un point de vue personnel, cette thèse m'a beaucoup apporté, que ce soit humainement, techniquement ou bien encore en termes de méthodologie.

Humainement d'abord, le fait de mener à bien un travail qui s'inscrit dans un cadre plus général, avec des élèves ingénieur par exemple qui permettent d'étendre un point précis, avec l'utilisation des prototypes développés ici dans des projets connexes, avec la poursuite de ce travail dans d'autres axes et par d'autres personnes, permet d'apprendre à travailler avec les autres et pour les autres. Diriger le travail de quelqu'un est plus ardu qu'on ne pourrait le penser.

Techniquement, ces années de thèse furent très riches car beaucoup de domaines nouveaux pour moi ont été abordés. Le temps réel tout d'abord, que j'ai pu voir assez en profondeur au travers de la comparaison des noyaux temps réel et aussi lors de conférences scientifiques relatives à ce sujet. Les domaines des bus logiciels et des composants, et en particulier CORBA que j'utilise encore aujourd'hui dans mon travail au quotidien. Unix, WindowsNT, les micro-noyaux et Linux enfin, qui m'ont permis d'avoir une assez bonne vue d'ensemble, je crois, des différents modèles de systèmes d'exploitation existants.

La méthodologie n'est peut être pas assez reconnue, pourtant lors d'un travail de longue haleine comme celui-ci, il faut acquérir des qualités d'organisation et des capacités à se projeter et à visualiser un but à long terme.

Index Général

Index Noms

P

R

S

T

U

V

W

Glossaire

– **A** –

ACE	Adaptative Communication Environment
ACL	Access Control List
AMI	Asynchronous Messaging Interface
ANA	ANAlogique (e.g. capteur)
API	Application Programming Interface
ATM	Asynchronous Transfer Mode

– **B** –

BOA	Basic Object Adapter
BSP	Board Support Package

– **C** –

CCM	CORBA Component Model
CIM	Computer Integrated Manufacturing
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
CoS	Classification of Services
COTS	Component Off The Shelf
COV	Change Of Value
CVT	Custom Value Type

– **D** –

DCE	Data Communications Equipment
DCOM	Distributed Component Object Model
DII	Dynamic Invocation Interface
DNS	Domain Name Server
DSI	Dynamic Skeleton Interface

– **E** –

EDF	Electricité De France, Earliest Deadline First
EJB	Enterprise Java Beans
ESIOP	Environment-Specific Inter-ORB Protocol

– **F** –

FDDI	Fiber Distributed Data Interface

FIFO First In First Out
FTP File Transfer Protocol

– G –

GIOP General Inter-ORB Protocol

– I –

ICC Inter Control Center
ICCP Inter Control Center Protocol
IDL Interface Definition Language
IFR InterFace Repository
IIOP Internet Inter-ORB Protocol
IMAP Internet Message Access Protocol
INS Interoperable Name Service
INT Intercepteur
IOR Interoperable Object Reference
IP Internet Protocol
IPC Inter-Process Communications
IR Interface Repository
ISO International Standards Organization
ISR Interrupt Service Routine

– L –

LAP Local Access Point
LDAP Lightweight Directory Access Protocol

– M –

MfgDTF Manufacturing Domain Task Force
MMS Manufacturing Message Specification
MS Microsoft

– N –

NDS Novell Directory Services
NFS Network File System

– O –

OBV Object By Value
ODL Open Document Language
ODMG Object Data Management Group
OLE Object Linking and Embedding
OMG Object Management Group
OQL Object Query Language
ORB Object Request Broker
OS Operating System
OSD Open Software Description

OSF	Open Software Foundation
	– P –
PCP	Priority Ceiling Protocol
PICS	Protocol Implementation Conformance Statement
PIP	Priority Inheritance Protocol
POA	Portable Object Adapter
POSIX	Portable Open System Interface eXchange
PPC	PowerPC
	– Q –
QoS	Quality of Service
	– R –
RBE	Report By Exception
RFP	Request For Proposals
RM-ODP	Reference Model for Open Distributed Processing
RMI	Remote Method Invocation
RMS	Rate Monotonic Scheduling
RPC	Remote Procedure Call
RT	Real-Time
RTP	Real-time Transport Protocol
	– S –
SCADA	Supervisory Control And Data Acquisition
SNMP	Simple Network Management Protocol
SQL	Structured Query Language
SSL	Secure Sockets Layer
	– T –
TAL	Time Allowed to Live
TAO	The ACE ORB
TASE2	Telecontrol Application Service Element 2
TCAP	Transaction Capabilities Application Part
TCP	Transmission Control Protocol
TLE	Time Limit for Execution
TLI	Transport Layer Interface
TOR	Tout Ou Rien
TR	Temps Réel
	– U –
UDP	User Datagram Protocol
URL	Uniform Resource Locator
	– V –
VCC	Virtual Control Center

VMD	Virtual Manufacturing Device
VME	VersaModule Eurocard bus

– **X** –

XML	eXtensible Markup Language
XTI	X/Open Transport Interface

Bibliographie Générale

[1] Laurent Bacon and Eric Gressier-Soudan. Etude des contraintes utilisateurs pour les systèmes d'exploitation temps réel - questionnaire. Technical Report HP-32/98/095/A, EDF R&D, September 1998.

[2] Laurent Bacon and Eric Gressier-Soudan. Setr : Systèmes d'exploitation temps réel - etude des contraintes utilisateurs - synthèse entretiens. Technical Report HP-32/98/079/A, EDF R&D, 1999.

[3] M. Bayart and CIAME. *Réseaux de Terrain, Description et Critères de choix*. Number ISBN 2-86601-724-2. Hermès, 1999. 192 Pages.

[4] G. Blair and J-B. Stefani. *Open Distributed and Multimedia*. Addison-Wesley, 1997.

[5] S. Blake and al. An architecture for differentiated services. Technical report, IETF DiffServ Working Group, December 1998. RFC 2475.

[6] J. R. Bunch, J. J. Dongarra, C. B. Moler, and G. B. Stewart. *LINPACK User's Guide*. Society for Industrial and Applied Mathematics Publications, Philadelphia, 1979.

[7] H. J. Curnow and B. A. Wichmann. A synthetic benchmark (referred to as the whetstone benchmark). *The Computer Journal*, 19(1) :43–49, 1976.

[8] D.Lantim. *.Net*. Number ISBN : 2-212-11200-9. Eyrolles, 1ère edition edition, Octobre 2003.

[9] EPRI. Draft input for the utility communications architecture, version 2.0. Technical Report 0.8, Electric Power Research Institute, March 1997. Prepared under the Auspices ofthe Profile Working Group of the MMS Forum.

[10] EPRI. Utility communications architecture, version 2.0 : Common applications service models (casm) and mapping to mms. Technical report, Electric Power Research Institute, USA, March 1997. Prepared under the Auspices ofthe Profile Working Group of the MMS Forum.

[11] EPRI. Utility communications architecture version 2.0 : Generic objects models for substation & feeder equipment (gomsfe). Technical report, Electric Power Research Institute, USA, 1997.

[12] Harm Smit François Jammes, Antoine Mensch. Service-oriented device communications using the devices profile for web services. In *6th International Middleware Conference - 3rd International Workshop on Middleware for*

Pervasive and Ad-Hoc Computing (MPAC05) Proceedings, Grenoble, France, December 2005. ACM / IFIP / USENIX.

[13] G. Gaines and M. Festa. A survey of rsvp/qos implementations. Technical report, RSVP Working Group, July 1998. update 2.

[14] B.O. Gallmeister. *POSIX.4 : Programming for the Real World.* O'Reilly & Associates Inc., first edition edition, January 1995.

[15] E. Gressier-Soudan, M. Epivent, A. Laurent, R. Boissier, D. Razafindramary, and M. Raddadi. Component oriented control architecture, the COCA project. *Special Issue on Manufacturing, Microprocessors and Microsystems Journal*, 23(2) :95–102, September 1999. Elsevier Science.

[16] Eric Gressier-Soudan. Prototyping a corba based mms -industrial communications with corba. In *OMG Technical Meeting*, Burlingame, California USA, September 2000. OMG. ftp ://ftp.omg.org/pub/doc/mfg/00-09-16.pdf.

[17] R. Grimes. *Professionnal DCOM Programming, A guide to creating practical applications with Microsoft's Distributed Component Object Model.* Wrox Press, 1997.

[18] OMG CCM Implementers Group. CORBA component model tutorial. In OMG, editor, *OMG Meeting*, Yokohama, Japan, April 2002. OMG.

[19] IETF. Ietf diffserv working group. http ://www.ietf.org/html.charters/diffserv-charter.html, May 2001. URL.

[20] IETF. Ietf intserv working group. http ://www.ietf.org/html.charters/intserv-charter.html, May 2001. URL.

[21] W-H. Kwon K-J. Myoung, D-S. Kim. Implementation of virtual factory using mms companion standard. In IEEE Control Systems Society, editor, *IEEE International Workshop on Factory Communication Systems (WFCS'02)*, Västeräs. Sweden, August 2002.

[22] Rabindra P. Kar. Implementing the rhealstone real-time benchmark (where a proposal's rubber meets the real-time road). *Dr. Dobb's*, April 1990.

[23] Rabindra P. Kar and Kent Porter. Rhealstone : A real-time benchmarking proposal (an independantly verifiable metric for complex multitaskers). *Dr. Dobb's*, February 1989.

[24] KEMA-ECC. *ICCP User Guide.* Mineapolis, USA., Final Draft edition, October 1996.

[25] Christophe Lizzi. Une architecture de traitement temps réel réparti sur réseau atm pour le système chorus. In *7th Real-Time Systems Conference (RTS'99)*, Paris, France, February 1999.

[26] Francisco Rodriguez Rubio Maria Teresa Ariza Gomez. Running distributed application using mms-corba. In IEEE Control Systems Society, editor, *3rd IEEE International Workshop on Factory Communication Systems (WFCS'00)*, pages 47–50, Porto, Portugal, 2000.

[27] Francisco Rodriguez Rubio Maria Teresa Ariza Gomez, F.J. Fernández. Implementing a virtual manufacturing device for mms/corba. In IEEE Control Systems Society, editor, *8th IEEE Interantional Conference on Emerging Thechnologies and Factory Automation (ETFA' 01)*, number ISBN : 0-88986-239-7, pages 100–103, Antibes, Francia, 2001.

[28] L. McVoy and C. Staelin. Lmbench : Portable tools for performance analysis. In *Proc. Winter 1996 Usenix*, pages 279–284, San Diego, CA, USA, January 1996. USENIX.

[29] H. Nussbaumer. *Téléinformatique*, volume 4. Presses Polytechniques et Universitaires Romandes, 1991.

[30] ObjectWeb. OpenCCM - homepage. http ://www.objectweb.org/openccm-/index.html, April 2003. URL.

[31] OMG. Real-time corba 1.0. Request for proposal, Object Management Group, January 1998.

[32] OMG. Real-time corba. Joined revised submission, Object Management Group, March 1999.

[33] OMG. The common object request broker : Architecture and specification. revision 2.4. Technical report, Object Management Group, October 2000.

[34] OMG. The ccm specification. Technical report, Object Management Group, June 2002.

[35] OMG. The common object request broker : Architecture and specification. revision 3.0. Technical report, Object Management Group, December 2002.

[36] OMG. Lightweight corba component model. Technical Report ptc/2004-06-10, Object Management Group, May 2004.

[37] Frank Pilhofer. Writing and using CORBA components. Technical report, Alcatel and FPX, April 2002.

[38] Patrick Pleinevaux. Des réseaux à tout faire ou presque .... Technical Report 6, DI-LIT, June 1994. Flash Informatique.

[39] Y. Pouffary and A. Young. Rfc 2126 : Iso transport service on top of tcp (itot). Technical report, ISO, March 1997.

[40] The GnU Project. Savannah web site. web, May 2005.

[41] A. Puder and K. Romer. *Mico Is Corba : A CORBA 2.2 Compliant Implementation.* Morgan Kaufmann Publishers, April 1999.

[42] José Rodriguez. *Architectures pour le support d ?applications temps réel distribuées avec CORBA.* Thèse de doctorat, Université Paul Sabatier, Toulouse, June 2005.

[43] Jose Rodriguez, Zoubir Mammeri, and Pascal Lorenz. Corba extensions to support qos-aware distributed systems. In *ICOIN*, pages 535–542, 2003.

[44] Jose Rodriguez, Zoubir Mammeri, and Pascal Lorenz. Corba notification service and rsvp-based architecture to provide qos guarantees. *Telecommunication Systems*, 24(2-4) :363–383, 2003.

[45] M.T. Rose and D.E. Cass. Rfc 1006 : Iso transport services on top of the tcp. version 3. Technical report, ISO, May 1987.

[46] T. Kuehnel A. Regnier B. Roe D. Sather J. Schlimmer H. Sekine D. Walter J. Weast D. Whitehead D. Wright S. Chan, C. Kaler. Devices profile for web services. Technical report, Microsoft, May 2005. Microsoft Developers Network Library.

[47] L. Sha, T. Rajkumar, and J.P. Lehoczky. Priority inheritance protocols : An approach to real-time synchronization. *IEEE Transactions on Computers*, pages 1175–1185, September 1990.

[48] B. Shirazi, L. Welch, B. Ravindran, C. Cavanaugh, B. Yanamula, R. Brucks, and E. Huh. Dynbench : a dynamic benchmark suite for distributed real-time systems. In J. Rolim and al. eds., editors, *Parallel and Distributed Processing : IPPS/SPDP Workshops*, pages 1335–1349, Springer, Berlin, April 1999.

[49] M. Spuri. *Earliest Deadline Scheduling in Real-Time Systems.* PhD thesis, Scuola Superiore S.Anna, Pisa, Italy, 1995.

[50] W.R. Stevens. *Unix Network Programming.* Software Series. Prentice Hall, 1990.

[51] A. Tang and S. Scoggins. *Open Networking with OSI.* Prentice Hall, 1992.

[52] Christian Toinard, Gérard Florin, and Christian Carrez. A formal method to prove ordering properties of multicast systems. *ACM Operating Systems Review*, 33(4) :75–89, 1999.

[53] H. Tokuda and T. Nakajima. Real-time scheduling and synchronization in real-time mach. In *11^th ACM Symposium on Operating Systems Principles.* ACM, 1991.

[54] UCS. TASE.2 object models. version 1996-08. iec870-6-802. iccp inter-control centre communications protocol version 6.1. Technical Report IEC 870-6-802, Utility Communications Specification Working Group, August 1996. Version 1996-08.

[55] UCS. TASE.2 services and protocol. version 1996-08. iec870-6-503. iccp inter-control centre communications protocol version 6.1. Technical Report IEC 870-6-503, Utility Communications Specification Working Group, August 1996. Version 1996-08.

[56] UCS. TASE.2 profiles. version 1996-11. iec870-6-702. iccp inter-control centre communications protocol version 6.1. Technical Report IEC 870-6-702, Utility Communications Specification Working Group, February 1997. Version 1996-11.

[57] E. Tordera V. Sempere, J. Mataix. Video transmission on industrial processes on map networks using mms. In IEEE Control Systems Society, editor, *7th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'99)*, Barcelona, Spain, October 1999.

[58] A. Valenzano, C. Demartini, and L. Ciminiera. *MAP and TOP Communications*. Number ISBN 0-201-41665-4. Addison-Wesley, Wokingham, England, 1992.

[59] VEGAS. Tcp vegas home page. http ://www.cs.arizona.edu/protocols/, August 2001. URL.

[60] J. Weaver, K. Mukhar, and J. Crume. *J2EE 1.4*. Number ISBN : 2-212-11484-2. Eyrolles, 1ère edition edition, Septembre 2004.

[61] N. Weiderman. Hartstone : Synthetic benchmark requirements for hard real-time applications. Technical report, Software Engineering Institute, Carneggie Mellon University, June 1989.

[62] Pascal Lorenz Zoubir Mammeri, Jose Rodriguez. Framework for corba extensions to support real-time object-oriented applications. *Telecommunication Systems*, 19(3-4) :361–376, 2002.

Bibliographie Personnelle : Conférences Internationales

[63] Laurent Bacon, Erwan Becquet, Eric Gressier-Soudan, Christophe Lizzi, Christophe Logé, and Laurent Réveilleau. Provisioning qos in real-time distributed object architectures for power plant control applications. In *2^{n}d IEEE International Symposium on Distributed Objects and Applications (DOA'2000)*, Antwerp, Belgium, September 2000. IEEE.

[64] Erwan Becquet, Mazen Abdallah, Eric Gressier-Soudan, François Horn, and Laurent Bacon. Object oriented timed messaging service for industrial ethernet : a fieldbus like architecture for power plant control and factory automation. In *Fieldbus Technology (FeT'2001)*, Nancy, France, November 2001. IFAC.

[65] Erwan Becquet, Philippe Decogné, Eric Gressier-Soudan, and Laurent Bacon. Evaluation de micro-noyaux temps réel pour les applications d informatique industrielle d'edf. In *Real Time Systems (RTS'2001)*, Paris, France, March 2001.

[66] Erwan Becquet, Eric Gressier-Soudan, Erwan Legrand, and Georg Hellack. Industrial messaging service for a power utility : Tase-2 over the real-time microkernel psos+. In *Emerging Technologies and Factory Automation (ETFA'2001)*, Antibes, France, October 2001. IEEE.

[67] Erwan Becquet, Erwan Legrand, Eric Gressier-Soudan, Georg Hellack, and Fayçal Boudghene-Stambouli. Mise en oeuvre d'une messagerie industrielle temps réel : Tase.2 sur le micro-noyau temps réel psos+. In *Real Time Systems (RTS'2001)*, Paris, France, March 2001.

[68] Erwan Becquet, Hans-Nikolas Locher, and Gressier-Soudan. Component-based industrial messaging service design for utilities. In *Emerging Technologies and Factory Automation (ETFA'2003)*, Lisbon, Portugal, September 2003. IEEE.

[69] Eric Gressier-Soudan and Erwan Becquet. Real-time CORBA-MMS for embedded systems. In *Workshop OMG*, San Diego, CA, USA, March 2001. OMG.

[70] C. Lizzi, L. Bacon, E. Becquet, and E. Gressier-Soudan. Prototyping qos based architecture for power plant control applications. In *IEEE Workshop on Factory Communication Systems (WFCS'2000)*, Porto, Portugal, October 2000. IEEE.

[71] Hans-Nikolas Locher, Erwan Becquet, and Gressier-Soudan. Monitoring the distributed virtual orchestra with a corba based objectoriented real-time data distribution service. In *International Symposium on Distributed Objects and Applications (DOA'2003)*, Catagna, Sicily (Italy), November 2003. FEDMA.

[72] Laurent Réveilleau, Erwan Becquet, Laurent Bacon, Jean-Michel Douin, Eric Gressier-Soudan, and François Horn. Towards a rt-java based embedded remote monitoring tool for small and medium power plant units. In *Emerging Technologies and Factory Automation (ETFA'2001)*, Antibes, France, October 2001. IEEE.

[73] Luc Teboul, Renaud Pawlak, Lionel Seinturier, Eric Gressier-Soudan, and Erwan Becquet. AspectTAZ : A new approach based on aspect-oriented programming for object-oriented industrial messaging services design. In *IEEE Workshop on Factory Communication Systems (WFCS'2002)*, Västerås, Sweden, August 2002. IEEE.

Bibliographie Personnelle : Conférences Nationales

[74] Laurent Bacon, Erwan Becquet, and Eric Gressier-Soudan. Etude comparative de micro-noyaux du commerce. In *7^eme Séminaire du Laboratoire d'Ingénierie de la Sureté du Fonctionnement*, Domaine de Villepreux, Bordeaux, France, March 1999.

[75] Erwan Becquet. Evaluation de systèmes temps réel du commerce. In *Réunion du groupe de travail Systèmes de Communication Industriels (SCI)*, Paris, France, January 2000.

[76] Erwan Becquet and Laurent Bacon. Evaluation de systèmes d'exploitation temps-réel pour les applications d'informatique industrielle edf. In *Jeunes Chercheurs en Systèmes (JCS'2000)*, Besançon, France, June 2000.

[77] Erwan Legrand, Eric Gressier-Soudan, , and Erwan Becquet. Setr : Systèmes d'exploitation temps réel - expérimentation t.a.s.e.2. Technical Report HP-32/00/XXX/A, EDF R&D, December 2000.

Bibliographie Personnelle : Rapports de Recherche

[78] Laurent Bacon and Erwan Becquet. Setr : Systèmes d'exploitation temps réel - psos+ - architecture. Technical Report HP-32/98/098/A, EDF R&D, December 1998.

[79] Laurent Bacon and Erwan Becquet. Setr : Systèmes d'exploitation temps réel - vxworks - architecture. Technical Report HP-32/98/100/A, EDF R&D, September 1998.

[80] Laurent Bacon and Erwan Becquet. Setr : Systèmes d'exploitation temps réel - description de edfbench. Technical Report HP-32/99/035/A, EDF R&D, July 1999.

[81] Laurent Bacon and Erwan Becquet. Setr : Systèmes d'exploitation temps réel - psos+ - tests. Technical Report HP-32/99/027/A, EDF R&D, March 1999.

[82] Laurent Bacon and Erwan Becquet. Setr : Systèmes d'exploitation temps réel - vxworks - tests. Technical Report HP-32/99/028/A, EDF R&D, March 1999.

[83] Laurent Bacon and Erwan Becquet. Bus logiciels temps réels - synthèse bibliographique. Technical Report HP-32/00/XXX/A, EDF R&D, December 2000.

[84] Laurent Bacon, Erwan Becquet, and Eric Gressier-Soudan. Setr : Systèmes d'exploitation temps réel - compte-rendu des présentations des fournisseurs. Technical Report HP-32/98/077/A, EDF R&D, 1998.

[85] Laurent Bacon, Erwan Becquet, and Eric Gressier-Soudan. Setr : Systèmes d'exploitation temps réel - compte rendu des présentations des systèmes d'exploitations temps réel du commerce pour les applications d'informatique industrielle d'edf. Technical Report HP-32/98/077/A, EDF R&D, 1998.

[86] Laurent Bacon, Erwan Becquet, and Eric Gressier-Soudan. Setr : Systèmes d'exploitation temps réel - etude des contraintes utilisateurs - critères d'évaluation. Technical Report HP-32/98/078/A, EDF R&D, 1998.

[87] Laurent Bacon and Philippe Decogné. Setr : Systèmes d'exploitation temps réel - lynxos - architecture. Technical Report HP-32/99/099/A, EDF R&D, July 1999.

[88] Laurent Bacon and Philippe Decogné. Setr : Systèmes d'exploitation temps réel - lynxos - tests. Technical Report HP-32/99/099/A, EDF R&D, July 1999.

[89] Erwan Becquet, Fayçal Boudghene-Stambouli, Georg Hellack, Erwan Legrand, Laurent Bacon, and Eric Gressier-Soudan. Experimenting tase.2 over psos+ for remote power plant control. Technical Report 00-06, CEDRIC, 2000.

[90] Eric Gressier-Soudan, Erwan Becquet, Erwan Legrand, Olivier Mallet, and Laurent Bacon. Setr : Systèmes d'exploitation temps réel - portage d'une souche tase.2 sur le micro-noyau temps réel psos+. Technical Report HP-32/00/057/A, EDF R&D, November 2000.

[91] Erwan Legrand, Eric Gressier-Soudan, , and Erwan Becquet. Setr : Systèmes d'exploitation temps réel - expérimentation t.a.s.e.2. Technical Report HP-32/00/XXX/A, EDF R&D, December 2000.

Annexe A

Systèmes d'Exploitation Temps Réel

A.1 Introduction

Les systèmes d'exploitation temps réel forment la deuxième couche de notre architecture au dessus du réseau de communication, c'est la première couche logicielle et donc la première à évaluer. En effet, Si le système d'exploitation n'offre pas certaines garanties temporelles, comment en avoir dans les couches supérieures ? Ce chapitre s'intéresse donc à une évaluation effectuée sur trois systèmes d'exploitation temps réel du commerce pour évaluer leurs performances et leur adéquation aux besoins d'une part, et pour évaluer la partie ingénierie d'autre part.

Si l'évaluation de performances est maintenant devenue une épreuve habituelle pour les systèmes d'exploitation (on verra cependant que tout n'est pas fait), l'évaluation de l'environnement d'ingénierie, (i.e. l'environnement de développement, l'aide, la facilité d'installation ...) est toujours oubliée à notre connaissance alors que dans une problématique industrielle, ces aspects sont prépondérants puisqu'influant beaucoup sur les coûts et les temps de développement logiciel.

A.2 Evaluation de Performances

A.2.1 Plate-forme

La plate-forme d'expérimentation est construite à partir des éléments suivants [1] :

- des processeurs Intel en réseau Ethernet, pour simuler les processeurs du niveau supervision (niveau 2 de la hiérarchie du contrôle-commande EDF,
- des processeurs PowerPC sur bus VME en réseau Ethernet, pour simuler les processeurs du niveau traitement des acquisitions et des commandes

OS-RT	Matériel		
	Intel 80x86	**PowerPC MVME**	**PowerPC MBX**
LynxOS	✓	✓	✗
pSOS+	✗	✓	✓
VxWorks	✗	✓	✓

✓ **Testé** ✗ **Non Testé**

TAB. A.1: Matrice de tests des systèmes

(niveau 1 de la hiérarchie du contrôle-commande EDF),

➡ d'un processeur PowerPC sur une carte MBX, pour simuler les capteurs et actionneurs intelligents.

La grille de tests d'évaluation en fonction des plates-formes matérielles sélectionnées est donnée dans le tableau A.1.

A.2.2 Bancs de tests existant

Peu de banc de tests pour l'évaluation de performances des systèmes temps réel existent. Les bancs de tests utilisés habituellement ne répondent pas à nos attentes. Certains ne sont pas dédiés spécifiquement au temps réel comme Lm-Bench [28] qui évalue les systèmes UNIX, ou bien Whetstone [7] qui s'intéresse plus à l'évaluation d'applications scientifiques et de calculs flottants ou bien encore Linpack [6] qui évalue les performances de résolution de systèmes d'équations linéaires.

D'autres sont plutôt centrés sur les applications distribuées, comme Dynbench [48] et Hartstone [61], et sont trop complexes pour les tests que nous envisageons qui, eux, sont au niveau des primitives système.

Rhealstone [23] [22] est le seul banc de tests qui corresponde à nos besoins mais l'ensemble des tests est insuffisant et surtout, l'implémentation est spécifique à un système. Le développement d'un banc de tests pour trois systèmes implique donc une réécriture complète que nous voulons éviter. Nous voulons un banc de tests portable et modulaire afin de tester les abstractions existant sur chacun des systèmes et afin de réduire les temps de portage sur un éventuel nouveau systèmes d'exploitation à tester.

A.2.3 Objectifs

Les tests de performances ont pour but d'évaluer les abstractions fournies par chacun des systèmes et leurs propriétés temps réel. Ces tests ont pour fonction

première de vérifier les propriétés temps réel des systèmes sélectionnés. Ils n'ont pas pour objectif une comparaison des performances des systèmes d'exploitation entre eux.

Cette mesure doit également nous permettre de vérifier le comportement des différentes abstractions des systèmes temps réel sous test. Si certaines abstractions sont communes aux quatre systèmes que nous devons étudier, des particularités émergent. L'étude de performance nous fournit les premiers éléments de distinction fonctionnelle entre les systèmes d'exploitation temps réel.

Les tests de performance mis en oeuvre sont de type boite noire. En effet, nous n'instrumentons pas le code noyau, nous nous plaçons au-dessus de l'exécutif, et nous n'utilisons que l'API offerte aux programmeurs. De plus, une approche portable a été retenue afin de pouvoir adapter rapidement le banc de tests à chacun des systèmes d'exploitation cible.

Les tests que nous décrivons forment un premier niveau d'évaluation. Un second niveau de tests devrait mettre en oeuvre une application type qui rassemblerait les fonctions qu'on trouve généralement dans les applications d'informatique industrielle d'EDF, l'application type fonctionnant seule sur la machine de test sans faire de communication réseau. Un troisième niveau de test devrait évaluer l'application précédente mais avec des perturbations de plusieurs ordres : entrées/sorties réseau, entrée/sorties disque, entrée/sorties bus processeur, et enfin des sorties écran.

Compte tenu du temps imparti pour cette étude, du nombre de systèmes d'exploitation, et du nombre de machines cibles à évaluer, seul le premier niveau de tests est effectué. Cependant, le banc de tests est réalisé en ayant pour objectif de faciliter ces deuxièmes et troisièmes niveaux de tests.

A.2.4 Méthode d'Evaluation des Primitives

A.2.4.1 Modèle des Tests en Boucle

Ne disposant pas d'une échelle de temps assez fine, pour les cas où la primitive système le permet [1], nous ferons exécuter cette opération un nombre de fois N (assez grand, par exemple 1 000, 10 000 ou 100 000 fois en fonction du système et des possibilités offertes) dans une boucle dénommée LOOP. En prenant la date système avant puis après la boucle, nous calculons le temps mis et divisons par N pour obtenir le temps moyen de la primitive. Ce modèle de tests que nous appelons "tests en boucle" est décrit dans la figure 1.

Cette méthode s'inspire de Lmbench [28], suite de tests éprouvée pour les systèmes UNIX.

[1] Impossible par exemple dans certains systèmes de créer 10 000 tâches pour évaluer la primitive de création de tâche.

Données : Nombre d'itérations, Primitive à évaluer
Résultat : Temps moyen d'exécution de la primitive

LOOP(N,`PrimitiveAEvaluer()`)
début
  $t_1 \leftarrow$ `PrendreDate()`
  pour $i \leftarrow 1$ **à** N **faire**
    `PrimitiveAEvaluer(`*paramètres*`)`
  fin
  $t_2 \leftarrow$ `PrendreDate()`
  $t_{result} \leftarrow \frac{t_2 - t_1}{N}$
fin

Algorithme 1: Modèle de tests en boucle

A.2.4.2 Modèle des Tests en Cascade

Le modèle en cascade est associé à l'évaluation de primitives qui bloquent le demandeur sur l'acquisition d'une ressource (sémaphore, verrou) détenue par une autre tâche ou sur l'attente d'un événement. Le modèle évalue simultanément la primitive d'acquisition ou d'attente, et son symétrique qui permet la libération de la ressource ou le déclenchement de l'événement.

Soit t_1 la date système prise avant le début du test . Nous effectuons ensuite une série d'appels à la primitive d'acquisition ou d'attente successivement par N tâches, un appel par tâche, à l'image d'une cascade d'appels. La dernière tâche stoque dans une variable t_2 la nouvelle date système, et déclenche la première libération qui va débloquer la tâche précédente qui elle-même va débloquer la précédente et ainsi de suite. La première tâche, après avoir été débloquée note la date système t_3. Nous pouvons alors obtenir simplement le temps d'acquisition de ressource ou d'attente d'un événement bloquant grâce à A.1. La formule A.2 permet d'obtenir le temps d'une libération de ressource ou de déclenchement d'un événement.

$$t_{acquisition} = \frac{t_2 - t_1}{N - 1} \tag{A.1}$$

$$t_{liberation} = \frac{t_3 - t_2}{N - 1} \tag{A.2}$$

Pour être précis, il faut soustraire du résultat obtenu la durée des actions nécessaires à la mise en oeuvre du test qui perturbe la mesure, en particulier la durée des changements de contexte de tâche, sauf si ce temps doit être comptabilisé parce que faisant partie de la mesure. La figure C.6 schématise le modèle de tests en cascade.

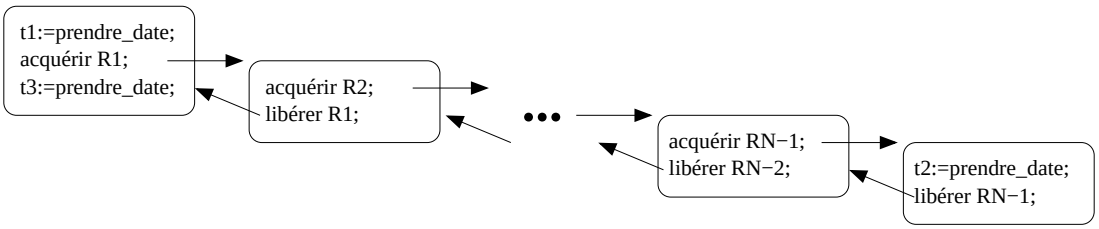

FIG. A.1: Modèle de tests en cascade

A.2.4.3 Campagne de tests

On appelle campagne de tests (CAMPAIN), l'action de mesurer plusieurs fois le temps d'une primitive (temps évalué grâce à un des deux modèles exposés en A.2.4.1 et A.2.4.2.

La notion de campagne permet de définir plusieurs temps de référence pour chaque primitive et ainsi de calculer des valeurs minimales, maximales, moyennes et un écart-type. Ces différentes valeurs permettent de quantifier la pertinence de certains résultats grâce en particulier à l'écart-type.

Le nombre d'itérations effectuées dans une campagne est choisi afin de pouvoir calculer des écart-types pertinents (20 dans notre cas).

A.2.4.4 Conditions de tests

Pour tous les OS, les tests s'effectueront avec des tâches en mode superviseur, certains exécutifs n'offrent pas de tâches en mode utilisateur. Les tests sont effectués en machine "vide". Aucun autre programme excepté ceux nécessaires aux tests ne s'exécuteront sur la machine pour éviter toute perturbation des résultats des tests.

En particulier, il n'y a pas de pile de communication (sauf pour les tests réseau), le noyau n'est pas généré en mode déverminage pour travailler avec la version la plus optimisée disponible.

A.2.4.5 Description formelle des primitives testées

Dans nos tests, les primitives de création ou de destruction d'abstractions (tâches, files de messages, sémaphores ...) ne sont pas évaluées. Dans une application temps réel, les contraintes de déterminisme temporel sont fortes, nous considérons que toutes les abstractions nécessaires à une application sont créées à l'initialisation. La destruction d'abstraction est synonyme d'arrêt ou de panne, ce qui correspond à une étape exceptionnelle en régime permanent de l'application.

Malgré ce choix d'évaluation, la signature des primitives de création et de destruction d'abstractions sera spécifiée dans chaque partie de ce chapitre sur les tests de performances.

A.2.5 Inversion de Priorité

Le phénomène d'inversion de priorité n'est pas simple à mettre en évidence. Nous proposons la détection du schéma d'inversion de priorité suivant les modèles évoqués dans les travaux de C. Lizzi pour proposer une architecture ATM [25] et qui fait référence à l'ordonnancement EDF dans les systèmes temps réel [49] et à l'ordonnancement dans le noyau Mach [53].

S'ils sont disponibles sur le système d'exploitation évalué et en fonction des fonctionnalités offertes, nous utilisons les outils de trace pour mettre en évidence l'occurrence du phénomène d'inversion de priorité pour les deux schémas précédents.

A.2.5.1 Modèle du Blocage Direct

Une tâche de faible priorité T_A prend un verrou nécessaire à une tâche T_B qui n'est pas prête, quand T_B devient prête, elle est bloquée par T_A qui ne change pas de priorité si l'inversion de priorité n'est pas pris en compte.

A.2.5.2 Modèle du Blocage Indirect

Une tâche de faible priorité T_A s'exécute (①sur fig. A.2) prend un verrou (②) nécessaire à une tâche T_C très prioritaire qui n'est pas prête, quand T_C devient prête (③) elle s'exécute et elle est bloquée par T_A (④). Une autre tâche, T_B, de plus forte priorité que T_A mais moins forte que T_C devient prête mais ne requiert pas la section critique. Elle prend le processeur (⑤). Cette nouvelle préemption par une tâche moins prioritaire retarde T_C. Ce cas de figure n'est pas légitime pour un système temps réel qui implante une stratégie qui évite l'inversion de priorité. Le chronogramme de la figure A.2 présente le problème du blocage indirect.

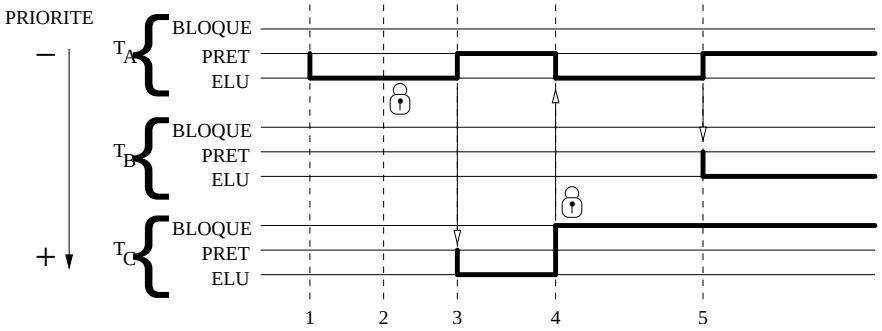

FIG. A.2: Inversion de priorité, modèle indirect

A.2.5.3 Modèle du Blocage Indirect avec Héritage de Priorité

Supposons que le scénario précédent se produit avec un système offrant l'héritage de priorité. Comme T_A bloque T_C pour l'acquisition du verrou de la section critique, T_A hérite de la priorité de T_C. Par ce mécanisme, T_B va être bloquée par T_A qui aura hérité de la forte priorité de T_C, or du point de vue applicatif, T_A est moins prioritaire que T_B. Le chronogramme présenté dans la figure A.3 décrit cette situation.

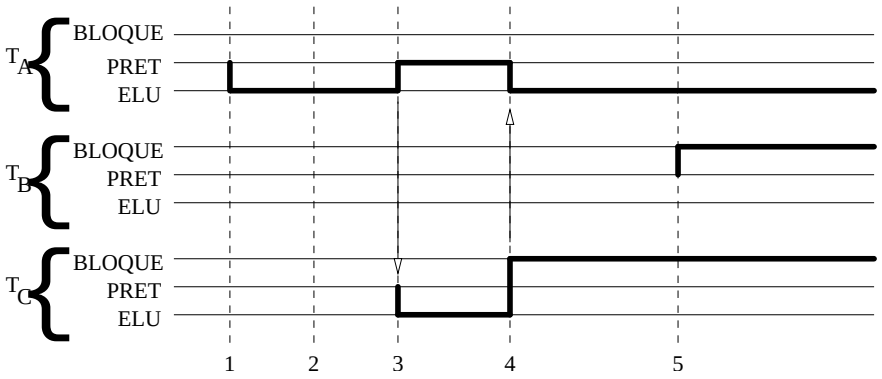

FIG. A.3: Héritage de priorité

A.2.6 Exécution et Ordonnancement

Le système doit permettre de développer des processus et/ou des tâches concurrentes. Il doit être possible de leur affecter des priorités numériques et d'observer que c'est toujours la tâche prête la plus prioritaire qui détient le processeur (pour les expérimentations, les machines sont monoprocesseur). Concernant les tâches, nous évaluerons les temps de changement de contexte. Les temps de création et de destruction ne seront pas évalués pour les raisons déjà explicitées. Ces temps, lorsqu'ils sont donnés, le sont pour information uniquement.

Si le système est capable de supporter plusieurs politiques d'ordonnancement, nous choisissons celle qui est la plus communément admise dans le domaine du temps réel : mécanisme FIFO par priorité avec préemptivité. Nous vérifions que la préemption s'effectue bien.

C'est dans cette catégorie de tests que nous vérifions si le système permet d'éviter l'inversion de priorité par héritage de priorité (PIP) ou autre politique (PCP par exemple)[47] quand cette facilité est annoncée par le fournisseur. Au moment de l'étude, aucun des systèmes d'exploitation temps réel ne fournissait de mécanisme basé sur PCP.

A.2.7 Déterminisme des accès mémoire

Les aspects relatifs à la gestion de la mémoire qui nous intéressent concernent l'allocation, et la désallocation de zones mémoire ainsi que la gestion de droits d'accès à ces zones mémoire. En fonction du système d'exploitation cible, deux types d'allocations peuvent être fournies à l'utilisateur : les allocations sans garantie temporelle et les allocations en temps borné. Si les deux mécanismes existent, c'est le deuxième qui sera évalué.

Pour tester les mécanismes d'allocation/désallocation, nous nous proposons d'effectuer deux types de tests. Un premier type consiste à évaluer le temps d'allocation et de désallocation d'une seule zone mémoire. L'autre type de tests consiste à observer l'effet de la fragmentation mémoire sur les temps d'allocation de tampons. Il faut déterminer si la gestion mémoire fournit une borne au temps d'allocation et au temps de désallocation.

Enfin, il est parfois nécessaire de projeter une zone mémoire d'une adresse à une autre. Si ce type d'opération existe, nous sommes amenés à l'évaluer également.

Enfin, nous évaluons la gestion de droits d'accès à une zone mémoire. C'est plutôt un test fonctionnel qu'un test de performance.

Nous n'abordons pas le déterminisme des accès mémoire d'une tâche. En effet, suivant le mode de gestion de la mémoire, l'accès à une référence mémoire est plus ou moins long. Si la donnée accédée est présente dans le cache, le temps d'accès est optimal, dans le cas contraire, il faut évincer une ligne du cache, charger la donnée manquante à sa place et le temps d'accès devient maximal. Une gestion mémoire temps réel devrait garantir le déterminisme des temps d'accès à la mémoire par une borne maximum. Ce type d'évaluation nécessite une approche boîte blanche qui n'a pas été envisagée.

A.2.8 Communication inter-processus

Les outils de communication inter-tâches ou inter-processus sont regroupés sous le terme IPC [2]. Ces outils peuvent être très différents. Ils matérialisent généralement deux mécanismes différents : la transmission du contrôle d'un processus à un autre, et la transmission de données. La mémoire partagée et les files de messages sont les deux types d'outils les plus fréquents. On trouve aussi des modes de communication très souples (l'IPC sur portes de ChorusOS est indépendant de leur localisation), et des implantations très efficaces (le RPC léger ou LAP de ChorusOS qui permet une communication inter-processus sans commutation de contexte puisque le traitement supporté par le demandé est exécuté dans le contexte d'exécution du demandeur).

Les communications inter-tâches se déclinent de plusieurs façons différentes : communication par mémoire partagée, communication par files de messages (tubes,

[2] **I**nter **P**rocess **C**ommunication

files ou boîtes aux lettres), communication par fichier projeté en mémoire. Bien que très pratique pour gérer la persistance de données en mémoire, ce dernier mécanisme n'est pas adapté aux environnements temps réel qui ont des contraintes temporelles et de déterminisme logique fort.

Le mode fichier projeté ne pénalise l'applicatif qu'au moment de l'initialisation. Nous n'envisageons pas pour l'instant d'évaluer respectivement le temps nécessaire à projeter une page, et le temps de mise à jour de l'image fichier. Certains environnements n'ont d'ailleurs pas de disque.

Le mode mémoire partagée est intéressant. La sémantique de communication par partage de variable est assez simple pour le programmeur bien qu'elle exige quelques précautions par l'utilisation de primitives de synchronisation. Toutefois, ce mécanisme de partage de variables n'est pas pénalisant en lui-même puisque la zone mémoire partagée entre deux tâches communiquantes est directement (physiquement) accessible dans l'espace de travail de chacune des tâches. Le programmeur utilisant la mémoire partagée est pénalisé au moment de la définition de la zone partagée, ce qui ne se produit qu'en phase d'initialisation, puis lors de l'accès aux primitives de synchronisation qui font l'objet du paragraphe suivant. Il n'est donc pas nécessaire d'évaluer les performances de ce mécanisme.

Nous évaluons donc uniquement les primitives qui concernent le mode d'échange de type file de messages.

A.2.9 Synchronisation Inter-Processus et Exclusion Mutuelle

La synchronisation peut être utilisée sous deux formes : pour faire de la communication inter-tâches sans transmission de données, et de la gestion de sections critiques. Dans la première catégorie, on trouve les sémaphores, les signaux et les événements, dans la seconde il faut classer les verrous d'exclusion mutuelle.

Parfois, il n'y a pas de verrous offerts par le système d'exploitation, seulement des sémaphores. Dans ce cas, notre étude se porte uniquement sur les sémaphores et les événements (mutex) qui sont traités dans le paragraphe suivant.

Pour les verrous d'exclusion mutuelle, on teste deux schémas d'utilisation : la prise de verrou suivie d'un relâchement de verrou (situation telle qu'il n'y ait pas de blocage de la tâche qui sollicite le verrou), la prise de verrou seule sur un verrou déjà pris. Le premier schéma est simple à évaluer et se prête bien au modèle de test général en boucle décrit dans le paragraphe A.2.4.1. Le second schéma, nécessite l'implantation de plusieurs tâches qui chacune à son tour prend le verrou et se bloque, à l'image d'une cascade d'acquisitions de verrou et de blocages comme décrit dans le paragraphe A.2.4.2. Le temps est évalué à l'aide du délai séparant la première prise de verrou par la première tâche de l'activation de la dernière tâche (qui suit l'avant-dernière prise de verrou de l'avant-dernière tâche). Un schéma symétrique est adopté pour évaluer le temps associé à une

libération de verrou.

Le mécanisme d'héritage de priorité est souvent appliqué aux verrous et non aux sémaphores. Ce test rentre dans le cadre du paragraphe A.2.6 car il touche à l'ordonnancement.

Le même modèle d'évaluation est adopté pour les sémaphores et les événements.

A.2.10 Datation et Temporisations

La gestion du temps et des temporisations est une partie capitale de l'API d'un système temps-réel. En effet, en temps réel, on est amené à gérer un grain d'horloge fin, à programmer des temporisations, à dater des actions ou des événements.

Cette partie fait l'objet d'une évaluation fonctionnelle et d'une évaluation de performances.

Il doit être possible de régler le grain des tics horloge, mais cet aspect est plutôt évalué dans le paragraphe sur la configurabilité A.4.2. Les possibilités offertes par cette fonctionnalité sont dépendantes de la plateforme matérielle.

Nous évaluons les délais induits par trois types de primitives : obtention du temps courant, obtention de la date courante, déclenchement d'une temporisation. La façon dont se décline ces types de primitives est souvent très spécifique à un noyau.

A.2.11 Interruptions

C'est une partie critique dans les tests de performance. Elle est spécifique à chaque architecture matérielle, et à chaque système d'exploitation, nous précisons ce point pour chaque système.

Ce point fait l'objet d'une évaluation fonctionnelle avant tout. L'évaluation de performance ne sera engagée que si elle ne nécessite pas d'instrumentation particulière du système d'exploitation et permet une évaluation boîte noire.

Les fournisseurs de système d'exploitation temps réel indiquent généralement le temps de latence d'une interruption. Cette valeur sera prise par défaut si nous ne sommes pas en mesure d'effectuer des tests boîte noire.

A.2.12 Communications Réseau

Certains fournisseurs proposent la pile TCP/IP pour les communications sur un bus VME. Pour les communications réseau, nous ne considérons pas le support VME, elles s'effectueront uniquement sur le réseau local Ethernet ou une liaison série si ce n'est pas possible.

Deux possibilités s'offrent à l'utilisateur : l'interface TLI/XTI [51], et l'interface sockets [50]. Les tests utilisent les sockets. Les sockets forment une abs-

traction communément admise pour la programmation de communications inter-tâches distantes. Deux modèles de communication sont proposés au-dessus de la famille des protocoles de l'Internet : le mode datagramme (non connecté) avec UDP,et le mode connecté avec TCP . A chaque fois, une relation client/serveur sert de schéma d'interaction pour l'évaluation. Pour le mode connecté, chaque message envoyé est précédé d'une ouverture de connexion, et suivi d'une fermeture de connexion.

Les conditions de test sont les suivantes : le noyau temps réel testé sur la cible ne dispose que des modules systèmes nécessaires aux communications réseau, la machine partenaire est une machine sous Windows NT de préférence, sinon sous Solaris 2.5.

En fait, les tests s'effectueront avec 4 longueurs de messages de données différentes[3] : un message court (6 octets de données), un message moyen (1460 octets), un message mi-long (8Ko), un message long (64 Ko). Les deux dernières tailles sont choisies afin de faire jouer la fragmentation dans les couches protocolaires.

Nous testons les performances sous les protocoles TCP et UDP, pour les rôles de client et de serveur, en testant toutes les combinaisons possibles pour ces différents paramètres, soit 16 tests au total. Le programme partenaire sur la machine hôte joue tantôt le rôle du client, tantôt le rôle du serveur. Le programme nécessaire à ce test est écrit sous la forme d'un client/serveur générique. Son lancement est paramétrable (protocole, port, taille de message, rôle).

Pour ce type de test, l'implantation de la pile de protocoles devrait faire la différence. Si le système cible le permet, nous évaluons l'effet d'une implantation multitâche, et d'un mode de gestion "zero copy" des tampons. Ces derniers aspects font l'objet d'une étude fonctionnelle d'après la documentation du fournisseur. Si l'étude fonctionnelle est favorable, les tests de communication sont lancés sans optimisation (zéro-copie ou multithread) d'abord si c'est possible, puis ils sont lancés à nouveau avec une ou plusieurs optimisations.

A.2.13 Conclusion

Les tests de performance que nous venons de décrire évaluent les propriétés temporelle des principales fonctions système pour mettre en oeuvre une application temps réel. Nous nous sommes alignés sur le plus petit dénominateur commun. Chaque système offre ses propres spécificités. Pour chaque système, nous nous attachons à évaluer les abstractions non standard qui nous semblent pertinentes pour la mise en oeuvre d'applications temps réel.

[3]Une trame Ethernet fait entre 46 et 1500 octets. Les couches IP et TCP ajoutent 40 octets d'entête. Ceci explique les valeur de 6 et 1460 octets de données utiles.

A.2.14 Résultats des tests

Les résultats présentés sont ceux obtenus pour les systèmes LynxOS 3.0.1 (Lynx Real-Time Systems), pSOS+ 2.1.1 (Integrated Systems Inc.) et VxWorks 5.3.1 (Wind River). Les temps retenus sont un sous-ensemble de ceux obtenus puisque le banc de tests complet fourni une centaine de temps différents par système, suivant les abstractions supportées.

Enfin, les temps présentés correspondent à la carte MBX (PPC 821 à 40 Mhz), au Pentium II (à 350 Mhz) et à la carte PowerPC MVME 2700 (PPC 750 à 266 Mhz) sachant que cette dernière est la seule plate-forme commune aux quatre systèmes de l'étude (cf. A.1). Les résultats complets pour chaque système sont disponibles dans [88], [81] et [82], respectivement pour LynxOS, pSOS+ et VxWorks. Des résultats partiels ont aussi été présentés [88][65] [76].

L'histogramme en figure A.4 synthétise les résultats obtenus avec pSOS+ à propos des sémaphores sur une carte MBX. Les résultats sont présentés normalisés. Les tests correspondent aux opérations classiques sur les sémaphores : création, prise de sémaphore (P), libération de sémaphore (V), libération entraînant le réveil d'une tâche bloquée (V ready), prise entraînant un blocage (P wait) et libération entraînant une préemption (V preempt). Le test "P/V Preempt" a servi à déduire "V Preempt". Les résultats présentés pour le 68040 sont issus de données constructeur.

La figure A.5 expose les temps (normalisés) obtenus sous LynxOS s ur un Pentium II, pour les files de messages. Les temps mesurés ici sont la création, la destruction, l'envoi et la réception avec les variantes quand ces opérations entraînent le réveil d'une tâche (ready), la préemption de la tâche appelante (preempt) ou son blocage (wait). Ces tests ont été de plus effectués avec une taille de message T, puis T*4, puis T*16, ceci afin de noter une influence éventuelle de la taille des messages sur les résultats.

Diagramme psos avec semaphores :

Les tests réseaux ont porté sur 4 tailles différentes de messages choisies de façon à mettre en évidence les problèmes liés à la fragmentation. De plus, la machine testée a joué successivement le rôle du serveur et celui du client, ceci en UDP et en TCP. Enfin nous avons testé le temps de création/destruction d'une socket. Nous avons aussi testé l'implantation "zero-copy" disponible sous VxWorks et pSOS+ (Tests suffixés par Z). Les résultats pour VxWorks et pSOS+ sont présentés sur la figure A.6. Les résultats de ces tests sont concluants en ce qui concerne les contraintes propres aux applications visées. On remarque particulièrement les gains appréciables en ce qui concerne l'implantation "zero-copy" des protocoles TCP-UDP/IP.

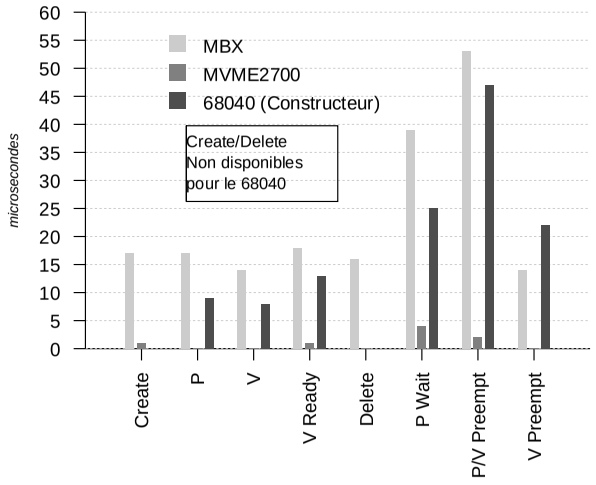

FIG. A.4: Résultats du banc de tests : Sémaphores sous pSOS+

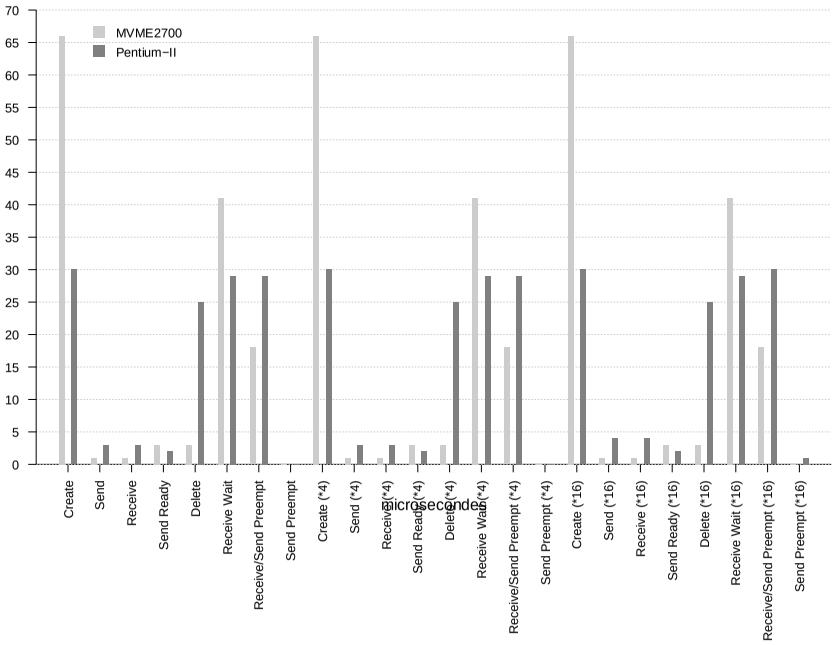

FIG. A.5: Résultats du banc de tests : Files de messages sous LynxOS

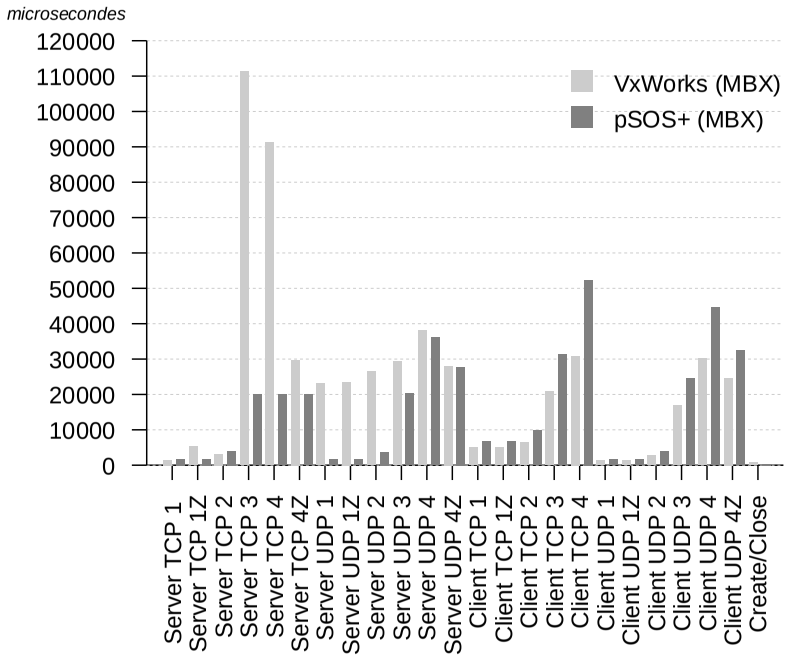

FIG. A.6: Résultats du banc de tests : Réseau sous VxWorks et pSOS+

A.2.15 Extensibilité du Benchmark

L'avantage de cette méthode de programmation pour l'écriture du banc de tests (utilisation massive de macros), c'est d'une part le portage simplifié sur un nouveau système temps réel, d'autre part l'extension des tests facilitée. En effet, pour appréhender la vitesse réelle du système, en utilisation plus "réaliste", on peut écrire des scenarii plus ou moins crédibles, à partie des macros du banc de tests. Une seule écriture suffit alors, il n'y aura plus qu'à recompiler sous tous les systèmes déjà portés pour avoir les résultats des tests.

Évidemment, l'extensibilité se situe aussi aux niveau de l'ajout d'un nouveau système, qui se fait assez rapidement. D'autant plus rapidement que le nouveau système intègre des API POSIX. En effet, il est préconisé lors des portages d'utiliser une API POSIX pour une abstraction si elle est disponible. Cela permettra de réutiliser la définition des macros lors d'un portage d'un nouveau système qui offre l'API POSIX. Ainsi, l'utilisation de l'API POSIX pour la gestion du temps dans LynxOS et VxWorks a permis de n'écrire qu'une fois le portage de cette API.

A.2.16 Bilan et Perspectives

Du point de vue des performances, les systèmes d'exploitation évalués tiennent les contraintes des applications d'EDF visées. En construisant une chaîne de traitement du capteur au niveau 0 jusqu'au poste opérateur en niveau 2, nous observons que les temps passés en gestion système (envois/réceptions en files, commutations de threads, communications réseaux avec passage par les sockets, utilisation de mémoire partagée et de sémaphores) quelle que soit la plate-forme ne sont pas prohibitifs. Ceci représente moins de 10% de la contrainte énoncée plus haut de 500ms de bout en bout.

A.3 Étude d'Ingénierie

A.3.1 Introduction

Pour tester les outils d'ingénierie des systèmes d'exploitation temps réel cibles, comme l'outil de déverminage s'il est disponible ou encore l'analyseur de performances, des procédures types sont mises en place, comme développer des sources contenant des erreurs caractéristiques et voir quels sont les moyens mis à notre disposition pour les résoudre.

Nous utilisons la suite de tests de performances pour effectuer ces tests en cours de développement, nous nous plaçons ainsi dans un cadre d'ingénierie réaliste et nous aurons une arborescence de fichiers sources identique dans tous les cas, sur tous les systèmes testés. Nous complétons les programmes de tests par des

applications de taille réduite si c'est nécessaire pour mettre en évidence d'éventuels problèmes spécifiques que l'utilisation de notre suite de tests n'aurait pas permis d'isoler.

A.3.2 Environnement de Développement

Les outils de développement regroupent les éditeurs, les navigateurs de classes, les outils de gestion de projet ... Il va falloir évaluer ces outils dans le cadre d'un développement réel, on peut par exemple comparer plusieurs opérations de base dans différents environnements de développement comme :

- ➡ Recherche de la définition d'une variable
- ➡ Modification d'un #define dont la localisation est inconnue
- ➡ Recherche de l'occurrence d'une variable dans le projet
- ➡ Imbrication des fichiers de définition (*.h)
- ➡ Génération automatique de Makefile
- ➡ ...

On peut définir le test suivant :

① Positionnement dans un source quelconque sur l'utilisation de la variable **Var**

② Recherche de la définition de cette variable dans le projet (i.e. tous les sources)

③ Retour à la position de départ après consultation de la définition

On note le nombre d'opérations à faire pour exécuter cette suite d'opérations dans l'environnement de développement à évaluer, le temps de recherche dans l'arborescence, la facilité d'utilisation (tout à la souris ? Raccourcis clavier ? ...). Par exemple dans un environnement UNIX standard il faut éditer le fichier source et lancer une recherche de chaîne de caractères sur le fichier, ou bien même lancer un grep[4] dans le répertoire du projet si on ne se souvient plus du nom du fichier dans lequel se trouve la définition. Cet environnement servira d'ailleurs souvent de point de référence car le développement sous UNIX ne fait appel, en général, à aucun outil intégré comme ceux que l'on pourra trouver dans les OS temps réel qui sont évalués ici.

A.3.3 Déverminage

Le déverminage est un point essentiel lors de la mise au point d'une application et sa mise en œuvre devient très complexe lorsque l'on se place dans le cadre d'applications multi-activités qui ont des contraintes temps réel. Il convient donc de

[4] **G**et **R**egular **Ex**P**r**ession : recherche de chaîne évoluée dans un ensemble de fichiers

vérifier que les fonctionnalités de l'outil de déverminage fourni avec les systèmes temps réel offrent toutes les fonctionnalités classiques de déverminage(points d'arrêts, visualisation de variables, de registres, de zones mémoire, pile des appels ...) mais aussi de fonctionnalités supplémentaires pour adresser ces problèmes nouveaux ; par exemple, peut-on avoir la liste des tâches s'exécutant à un moment donné ? Peut-on faire des suppositions sur les points d'arrêt pour que telle tâche s'y arrête mais pas telle autre ?

De plus nous regardons quel est le niveau d'intégration du dévermineur par rapport au système en particulier, c'est-à-dire quelles sont les commandes spécifiques dans le dévermineur qui nous permettent de suivre l'état des ressources du système :

- Liste des objets systèmes : tâches, partitions, files de messages, sémaphores ...
- Informations détaillées sur les tâches : priorité, attributs, état ...
- Informations sur le système : nombre maximum de tâches, composants installés ...
- ...

On prend soin d'évaluer également l'ergonomie du système, pour le passage des commandes et l'affichage des résultats. En effet dans un environnement de déverminage, il peut rapidement y avoir une surchargé due aux nombreuses informations et leur intégration est un point important qui contribue à l'efficacité du système.

Pour tester ces fonctionnalités, on peut se poser des problèmes précis et voir quels sont les moyens apportés par le dévermineur pour les résoudre. Dans le cadre du développement de notre suite de tests, nous sommes amenés à déverminer notre application, il est alors intéressant de noter si l'outil fourni avec les systèmes temps réel répond à nos attentes et permet un déverminage rapide et efficace. Au cas où notre application n'aurait pas besoin d'être déverminée ou peu, ou peut "simuler" des bogues et évaluer la capacité au dévermineur à fournir des informations pertinentes et utiles. Il est possible de générer les problèmes suivants :

- Variable non initialisée,
- Débordement de pile,
- Écrasement de valeurs,
- Problème de portée,
- ...

A.3.4 Trace

Dans un environnement temps réel et multitâche, la trace permet de vérifier d'abord un comportement logique (en particulier le bon ordonnancement des

tâches entre elles) et éventuellement des contraintes temporelles. Il faut vérifier que la trace peut être paramétrée de façon précise, commencer et finir sur des événements systèmes choisis ou sur des points spécifiés dans le code de l'application. On vérifie également si les résultats de la trace peuvent être sauvegardés, rechargés ou importés dans des formats de fichiers qui pourraient permettre une réutilisation dans d'autres buts (statistiques par exemple). Peut-on tracer tous les événements systèmes spécifiques comme les créations d'objets systèmesou les changements de contexte ? Quelle est le grain de cette trace au niveau temps ? Celle de l'horloge système (de 10ms en général dans un système UNIX) ou l'outil de trace utilise-t-il une autre horloge plus précise ?

Nous posons plusieurs problèmes types, comme préconisé en introduction, et voyons dans quelle mesure l'outil de trace fourni nous permet de les résoudre. Pour commencer, on peut développer un programme, composé de deux tâches T_1 et T_2, qui va mener à une situation d'interblocage, et dont le pseudo-code est donné en 2 pour T_1 et en 3 pour T_2.

Données : sem_1 et sem_2, deux sémaphores, Code de la Tâche T_2
`MAIN()`
début
  `CréationTâche(`T_2`)`
  `PrendreSémaphore(`sem_1`)`
  `DémarrerTâche(`T_2`)`
  `PrendreSémaphore(`sem_2`)`
fin

Algorithme 2: Code de la tâche T_1.

Données : sem_1 et sem_2, les deux même sémaphores que pour T_1
`MAIN()`
début
  `PrendreSémaphore(`sem_2`)`
  `PrendreSémaphore(`sem_1`)`
fin

Algorithme 3: Code de la tâche T_2.

Le programme précédent va se bloquer car on aura pris soin de donner à T_2 une priorité supérieure à celle de la tâche principale T_1. T_2 va donc prendre le sémaphore sem_2, se bloquer sur la prise de sem_1 puisque T_1 le détient et rendre la main. La tâche T_1 va alors se bloquer sur la prise du sémaphore sem_2, détenu par T_1. On trace donc cette exécution pour voir si l'on assiste bien à la situation d'inter blocage ainsi décrite.

De la même façon, on peut écrire d'autres programmes pour mettre en évidence des situations bien précises (inversion de priorité, problème producteur / consommateur, défaut d'échéance ...) et vérifier que l'on observe bien le phénomène via l'outil de trace fourni. On devrait pouvoir observer de visu les tâches bloquées sur des sémaphores, non exécutées par le système ...

Nous pouvons aussi tout simplement observer la trace de la suite de tests qui produit des situations intéressantes, comme dans le cas de la mesure du temps de changement de contexte (cf. A.2.6). Cette solution a en plus l'avantage d'être assez exhaustive par définition et nous pourrons ainsi tracer la plus grande partie des appels systèmes propres à chaque système.

A.3.5 Observation de performances & Dimensionnement

L'observation des performances d'une application permet de savoir quelles sont les parties critiques dans les différentes fonctions qui la composent, parties que l'on peut alors optimiser pour un gain maximal. On regarde si l'outil fourni avec chacun des systèmes, s'il existe, permet par exemple de savoir quel est le pourcentage du temps total d'exécution de l'application fonction par fonction, ou tâche par tâche. Nous pouvons également voir si le nombre de changements de contextes par application peut être tracé et ainsi détecter une mauvaise conception de l'application.

Ici, les performances sont des performances en termes de temps d'exécution essentiellement et les performances qualitatives ne sont pas visées.

Le dimensionnement est l'action de bien choisir les tailles des différents paramètres de l'application afin de suivre au plus juste les besoins. C'est surtout de l'utilisation mémoire qu'il va s'agir, on va déterminer les tailles des piles des différentes tâches, les tailles des files de messages, des différentes zones tampons de communication, etc... Cette phase nécessite un outil qui permette de voir durant l'avancement de l'application, l'utilisation de toutes ces ressources.

On note la granularité requise et la variété des paramètres qu'il est possible de superviser, ce sont à peu près les seuls points à évaluer ici. Il est aussi possible d'évaluer, comme pour d'autres outils, l'ergonomie de l'interface homme / machine et si elle fait ressortir nettement les informations capitales, si elle est paramétrable en fonction du but recherché.

A.3.6 Support technique

La qualité du support technique s'évalue en fonction du temps de réponse et de la pertinence de la réponse apportée. Dans un premier temps, on peut déjà évaluer ces paramètres au moment du développement de la suite de tests de performances et donc de l'utilisation du produit qui entraîne plusieurs questions.

Cependant pour être cohérent et pour pouvoir comparer les différents systèmes entre eux, il est souhaitable de mettre en place une démarche bien définie, c'est-

à-dire poser le même problème à chacun des supports techniques et comparer les réponses entre elles. Il s'agit de trouver une ou plusieurs questions qui vérifient les propriétés suivantes :

- ➡ assez ardue,
- ➡ pertinente par rapport à tous les OS,
- ➡ réponse introuvable dans la documentation fournie.

Cela dit, les questions qui se posent lors de la conception de la suite de tests constituent déjà une base solide et suffisante pour évaluer la qualité du support technique. De plus, les systèmes étant différents, il peut être difficile de trouver des questions qui se posent pour tous et ce peut d'ailleurs être un bon objectif que de dégager des tests types pour tous les systèmes dans ce domaine.

A.3.7 Documentation

L'évaluation de la documentation peut se décomposer en deux grandes parties, à savoir :

① Fonctionnalités d'utilisation

② Qualité intrinsèque de la documentation

Le premier point ne pose pas trop de problèmes, il s'agit de voir quels sont les outils fournis à l'utilisateur pour utiliser la documentation : index, format hypertexte bien exploité, recherche par mot clé, par mot clé approché (recherche mots se rapprochant de ...), impression, récupération, export ...

La qualité intrinsèque se juge à l'utilisation, la question est de savoir si la documentation permet en général de trouver les réponses cherchées, et quelle est alors la qualité de la réponse. Par exemple dans le cas des appels systèmes, un descriptif assez détaillé est-il fourni avec le comportement dans les cas limites, les valeurs d'erreur, les valeurs possibles des différents paramètres, la définition des structures utilisées... Tout ceci est évalué pendant la période de développement de la suite de tests de performances.

La documentation des outils fournis pour le développement est également évaluée. Les documentations sont-elles dans un format unique et cohérent, avec éventuellement des références croisées de l'une à l'autre ? Sont-elles au contraire totalement indépendantes et utilisant des règles propres pour chacune (dans le cas d'un outil fourni par une société tierce par exemple) ?

A.4 Tests d'Administration

A.4.1 Installation

L'installation est évaluée suivant deux points, on note la simplicité d'installation d'une part et le temps de l'installation d'autre part. La simplicité de l'ins-

tallation se note suivant les critères suivants : nombre d'informations à connaître (faut-il avoir une bonne connaissance du système hôte pour installer le système ?), nombre d'informations à saisir (numéros de série ...) ... Cette première phase consiste donc en l'installation sur le système hôte de tout l'environnement de développement.

Le bon fonctionnement de cet environnement est vérifié. Il permet bien de faire du développement, de gérer un projet, des makefile, que tous les éléments nécessaires à la compilation se trouvent bien à leur place (vérification faite en compilant un des programmes d'exemple fournis).

Dans le cas d' un environnement hôte / cible, il est nécessaire de vérifier que le téléchargement d'une application sur la machine cible (il s'agit ici d'une carte MVME2700 et d'une carte MBX) est possible. Est-ce simple ? Cela requière-t-il des connaissances particulières sur la cible (topologie, adressage de la mémoire, dévermineur fourni, méthodes de démarrage ...) ? Pour deux cartes différentes, deux noyaux différents sont nécessaires, est-il simple de générer des binaires différents pour une même application ? Est-il facile de passer d'une cible à l'autre au niveau du développement ?

A.4.2 Configuration

Deux niveaux de configuration sont distingués : configuration de l'environnement de développement d'une part, configuration du noyau lui-même d'autre part. Le premier va regrouper toutes les possibilités données au(x) développeur(s) pour adapter l'environnement à ses (leurs) préférences (éditeur, couleurs, indentation ...) alors que le deuxième va permettre de choisir les modules du noyau à inclure ainsi que toutes les variables systèmes (nombre de tâches max., nombre de sémaphores, méthode de boot ...).

Dans tous ces systèmes, plusieurs produits différents (éditeur, dévermineur, compilateur ...) cohabitent,y-a-t-il autant de niveaux de configuration différents que d'outils ? Et autant d'interfaces différentes ? Cette multiplication d'interfaces si elle existe (et donc le manque d'intégration au niveau configuration) n'est-elle pas trop lourde pour le développeur utilisateur ? Au contraire l'intégration des outils a-t-elle pris en compte ce problème de configuration des différents outils intégrés et offre-t-elle une interface unique ?

Le deuxième niveau de configuration permet au développeur de générer un noyau paramétré suivant les besoins de son application. Il faut vérifier si le choix des différents paramètres se fait d'une manière centralisée ou non. Il est également nécessaire de regarder si le paramétrage est assisté (explication des variables, différents choix possibles, répercussions sur le système, dépendances entre modules ...).

A.4.3 Extensibilité

L'extensibilité d'un tel produit se divise en deux grandes parties, les mêmes que celles déjà vues dans le cas de la configuration : environnement de développement et noyau.

Est-ce que l'environnement de développement permet de gérer plusieurs utilisateurs ? Si oui, ce groupe est-il statique ou dynamique ? De même, l'ajout et le retrait de nouvelles cibles pour le développement (nouvelles cartes éventuellement de nouveaux types) est-il possible ? Si ces évolutions sont possibles, sont elles pour autant faciles à mettre en œuvre (nombre de manipulations, redémarrage de l'environnement, changements "à chaud" ...) ?

Pour mesurer l'extensibilité du noyau, on regarde si la possibilité d'ajouter des modules dynamiquement (pendant l'exécution du noyau sur une cible) est offerte. Nous voyons également si nous pouvons étendre les fonctionnalités du noyau par un autre système (chargement de code objet sur la cible dynamiquement par exemple).

A.4.4 Résultats

Les résultats sont consignés dans les tableaux A.2 et A.3

Si les trois sytèmes évalués disposent globalement des mêmes fonctionnalités (LynxOS est un peu plus restreint mais sa philosophie POSIX devrait permettre de rajouter sans problèmes des outils GNU existant), la différence se fait essentiellement au niveau de l'intégration des différents outils.

A ce niveau, VxWorks sort incontestablement du lot avec une interface développée en Tcl/Tk et donc facilement paramétrable et extensible et une intégration excellente de tous les outils proposés (éditeur, gestion de projet, lanceur, dévermineur ...). De plus cet environnement est très souple et permet par exemple de remplacer l'éditeur proposé par défaut.

pSOS+ propose une pseudo-intégration puisqu'une barre d'outils permet de lancer tous les programmes disponibles (reposant sur CORBA). Mais ce système équivaut à lancer plusieurs outils très différents avec des arguments ad hoc. Même si ce système est utile et permet un gain de temps appréciable, on est encore loin de ce qu'on peut appeler une intégration.

LynxOS a une philosophie UNIX donc plus en ligne de commande, de plus la facilité d'utilisation des outils fournis est diverse. Certains, comme le dévermineur, offrent une interface graphique complète et très agréable alors que d'autres doivent se contenter du mode texte. Évidemment ce manque d'intégration permet d'un autre côté d'être complètement libre quant au choix de certains outils comme l'éditeur.

Le choix sera donc très lié aux besoins et aux habitudes de développement. Les coûts et les connaissances internes feront la différence.

Ingénierie (1/2)			
	LynxOS	**VxWorks**	**pSOS+**
Développement	✓environnement X-WindoW ✓coloration syntaxique et indentation automatique dans l'éditeur ✓outils GNU ✗manque d'intégration ✗pas de générateur de fichiers makefile ✗pas de gestionnaire de versions	✓concepts UNIX ✓outils GNU ✓coloration syntaxique dans l'éditeur ✓excellente intégration ✓interface graphique paramétrable ✗pas de générateur de fichiers makefile ✗pas de gestionnaire de versions	✓outils de recherche et de gestion des sources ✓arborescence d'utilisation des symboles ✓gestion des versions ✓menu d'exécution des cibles des makefiles ✓bonne intégration ✗pas de générateur de fichiers makefile
Déverminage	✓nombreuses fonctionnalités (basé sur GNU gdb) ✓informations sur les objets systèmes ✓interface graphique ✓facile à utiliser	✓nombreuses fonctionnalités (basé sur GNU gdb) ✓informations sur les objets systèmes ✓interface graphique paramétrable ✓facile à utiliser ✓excellente intégration (dans l'éditeur)	✓facilité de mise en œuvre ✓nombreuses fonctionnalités ✓informations sur les objets systèmes ✓interface graphique ✓bonne intégration ✗pas de recherche de motif en mémoire

Ingénierie (1/2) *(suite)*			
	LynxOS	**VxWorks**	**pSOS+**
Trace	✗pas d'outil de gestion de traces livré	✓trace les opérations réalisées sur la cible ✓trace les appels systèmes ✓trace les changements de contexte ✓positionnement d'événements déclencheurs ✓filtre paramétrable ✓très ergonomique, utilisation souple	✓trace les opérations réalisées sur la cible ✓trace les appels systèmes ✓trace les changements de contexte ✓positionnement d'événements déclencheurs ✓filtre paramétrable ✓très ergonomique, utilisation souple

TAB. A.2: Résultats d'Ingénierie : LynxOS, VxWorks et pSOS+ (1/2)

Ingénierie (2/2)			
	LynxOS	**VxWorks**	**pSOS+**
Observation des Performances & Dimensionnement	✓infos sur les processus ✓infos sur les tâches ✓infos sur l'utilisation mémoire ✓facilement paramétrable	✓infos sur les objets systèmes ✓taille des piles (allouée/utilisée) ✓contenu des registres	✓infos sur les tailles mémoires caractéristiques ✓paramétrage de la fréquence d'échantillonage et de représentation ✓possibilité d'utiliser l'outil de trace

Ingénierie (2/2) *(suite)*			
	LynxOS	**VxWorks**	**pSOS+**
Documentation	✓plusieurs formats (PDF, PostScript, Papier) ✓page manuel pour les API systèmes ✓documentation complète pour les outils GNU ✗pas de documentation pour les outils Lynx (Total/db, Process viewer) ✗pas de liste des abstractions ✗pas d'outil de recherche dans la documentation	✓complète et précise ✓page manuel pour les API systèmes ✓très bonne documentation papier ✗manque d'intégration ✗pas d'outil de consultation ✗format HTML	✓2 formats : HTML & PDF ✓complète et précise ✓page manuel pour les API systèmes ✓documentation automatique des programmes ✓outil de recherche ✗manque d'intégration
Support Technique	✓bon temps de réponse ($<$ 1 jour) ✓réponses de qualité ✓très bonne connaissance du système et des API	✓bon temps de réponse ($<$ 2 jours) ✓réponses de qualité ✓très bonne connaissance du système ✓fourniture de jeux de tests et d'exemples	✓bon temps de réponse ($<$ 1 jour) ✓réponses de qualité au niveau des API ✗mauvaise connaissance du système (interne)

TAB. A.3: Résultats d'Ingénierie : LynxOS, VxWorks et pSOS+ (2/2)

Annexe B

Evaluation des Performances des Systèmes d'Exploitation Temps Réel

B.1 Résultats

B.1.1 LynxOS

Test		MVME 2700					Pentium II				
		Min	**Max**	**Avg**	**Err**	**Sdv**	**Min**	**Max**	**Avg**	**Err**	**Sdv**
Tâches	Create	26	28	26	0	0	7	9	7	0	0
	Start	2	2	2	0	0	1	1	1	0	0
	Set Priority	3	5	3	0	0	2	2	2	0	0
	Suspend	1	2	1	0	0	1	1	1	0	0
	Resume	1	2	1	0	0	1	1	1	0	0
	Suspend/Resume	3	3	3	0	0	3	3	3	0	0
	Delete	5	5	5	0	0	5	5	5	0	0
	Context Switch	6	6	6	0	0	3	3	3	0	0
Mémoire	Get Buffer	1	12	1	0	0	0	2	0	0	0
	Return Buffer	0	3	0	0	0	0	0	0	0	0
	Get Buffer *4	1	26	1	0	0	0	2	0	0	0
	Return Buffer *4	1	1	1	0	0	0	0	0	0	0
	Get Buffer *16	1	76	1	10	0	1	2	1	0	0
	Return Buffer *16	1	1	1	0	0	0	0	0	0	0
Tps	Get	2	2	2	0	0	5	6	5	0	0
	Pause	0	0	0	0	0	0	0	0	0	0
	Set	25	26	25	0	0	35	35	35	0	0
Sém	Create	22	906	23	0	0	16	19	16	0	0
	P	0	0	0	0	0	0	0	0	0	0
	V	0	0	0	0	0	0	0	0	0	0
	V Ready	2	2	2	0	0	1	1	1	0	0
...											

Test		MVME 2700					Pentium II				
		Min	**Max**	**Avg**	**Err**	**Sdv**	**Min**	**Max**	**Avg**	**Err**	**Sdv**
	Delete	8	8	8	0	0	7	7	7	0	0
	P Wait	33	35	33	0	0	27	1252	28	0	0
	P/V Preempt	10	10	10	0	0	15	15	15	0	0
	V Preempt	0	0	0	0	0	0	0	0	0	0
Files Messages Var	Create	66	72	66	0	0	30	50	30	0	0
	Send	1	1	1	0	0	3	3	3	0	0
	Recv	1	1	1	0	0	3	3	3	0	0
	Send Ready	3	3	3	0	0	2	2	2	0	0
	Delete	3	3	3	0	0	25	25	25	0	0
	Recv Wait	41	75	41	0	0	29	639	29	0	0
	Recv/Send Preempt	18	18	18	0	0	29	29	29	0	0
	Send Preempt	0	0	0	0	0	0	0	0	0	0
	Create *4	66	67	66	0	0	30	30	30	0	0
	Send *4	1	1	1	0	0	3	4	3	0	0
	Recv *4	1	1	1	0	0	3	3	3	0	0
	Send Ready *4	3	3	3	0	0	2	2	2	0	0
	Delete *4	3	3	3	0	0	25	26	25	0	0
	Recv Wait *4	41	43	41	0	0	29	639	29	0	0
	Recv/Send Preempt *4	18	18	18	0	0	29	30	29	0	0
	Send Preempt *4	0	0	0	0	0	0	0	0	0	0
	Create *16	66	66	66	0	0	30	30	30	0	0
	Send *16	1	1	1	0	0	4	4	4	0	0
	Recv *16	1	1	1	0	0	4	4	4	0	0
	Send Ready *16	3	3	3	0	0	2	2	2	0	0
	Delete *16	3	3	3	0	0	25	25	25	0	0
...											

Test		MVME 2700					Pentium II				
		Min	**Max**	**Avg**	**Err**	**Sdv**	**Min**	**Max**	**Avg**	**Err**	**Sdv**
	Recv Wait *16	41	78	41	0	0	29	639	29	0	0
	Recv Send Preempt*16	18	18	18	0	0	30	30	30	0	0
	Send Preempt *16	0	0	0	0	0	1	1	1	0	0
Réseau	Create/Close	24	26	25	0	0	20	23	21	0	0
	Client TCP 1	1558	1760	1639	0	1879	2088	2110	2098	0	32
	Client TCP 2	3204	3332	3260	0	1185	4408	4426	4417	0	32
	Client TCP 3	17668	17807	17730	0	1439	24647	24680	24663	0	92
	Client TCP 4	30446	30633	30547	0	2344	42256	42298	42274	0	105
	Client UDP 1	396	652	398	0	17	495	660	496	0	0
	Client UDP 2	2101	2507	2103	0	6	3040	3044	3041	0	0
	Client UDP 3	16296	16815	16540	0	48268	18050	18055	18051	0	0
	Client UDP 4	30799	31314	30811	0	237	32459	32465	32461	0	1
	Server TCP 1	414	32202	26365	0	79843k	527	531	529	0	0
	Server TCP 2	199857	200477	200150	0	25339	199989	200140	200137	0	1
	Server TCP 3	69550	200109	87463	0	50011k	199039	200041	200036	0	217
	Server TCP 4	200106	221336	213530	0	31103k	200037	200539	200537	0	0
	Server UDP 1	420	28608	26327	0	34368k	504	520	507	0	2
	Server UDP 2	10568	47767	44244	0	1490k	3014	3120	3020	0	154
	Server UDP 3	26401	53489	49473	0	5048k	29773	29900	29786	0	705
	Server UDP 4	36037	49778	42030	0	3079k	55529	56043	55560	0	10628
Séms Bin	Create	434	760	437	0	224	770	850	773	0	143
	P	2	2	2	0	0	2	2	2	0	0
	V	2	2	2	0	0	2	2	2	0	0
	V Ready	2	2	2	0	0	1	1	1	0	0
	Delete	86	208	86	0	0	102	103	102	0	0
...											

Test		MVME 2700					Pentium II				
		Min	**Max**	**Avg**	**Err**	**Sdv**	**Min**	**Max**	**Avg**	**Err**	**Sdv**
	P Wait	33	36	33	0	0	27	39	27	0	0
	P/V Preempt	9	9	9	0	0	14	15	14	0	0
	V Preempt	0	0	0	0	0	0	0	0	0	0
Mutexes	Create	6	21	6	0	0	1	2	1	0	1
	P	0	0	0	0	0	0	0	0	0	0
	V	0	0	0	0	0	0	0	0	0	0
	V Ready	5	5	5	0	0	3	3	3	0	0
	Delete	60	61	60	0	257	2	53	39	0	359
	P Wait	35	41	35	0	0	18	1635	18	0	0
	Scenario	19	20	19	0	0	30	40	39	0	5
	V Preempt	0	0	0	0	0	0	0	0	0	0

B.1.2 pSOS+

Test		MBX 821					MVME 2700				
		Min	**Max**	**Avg**	**Err**	**Sdv**	**Min**	**Max**	**Avg**	**Err**	**Sdv**
Tâches	Create	152	157	156	4	2	9	14	12	4	6
	Start	49	54	51	4	6	4	9	5	4	5
	Set Priority	16	17	16	0	0	0	1	0	0	0
	Suspend	14	19	16	4	6	0	4	0	4	2
	Resume	19	19	19	4	0	0	4	1	4	4
	Suspend/Resume	35	36	35	0	0	1	2	1	0	0
	Delete	49	54	49	4	0	0	4	3	4	2
	Context Switch	7	8	7	0	0	1	2	1	0	0
Mémoire	Create	24	29	24	4	3	0	4	1	4	4
	Delete	14	19	16	4	6	0	4	1	4	4
	Get Buffer	17	18	17	1	0	1	2	1	1	0
	Return Buffer	21	23	22	1	1	1	2	1	1	0
	Create *4	24	29	28	4	2	0	4	1	4	4
	Get Buffer *4	17	18	17	1	0	1	2	1	1	0
	Return Buffer *4	21	23	22	1	1	1	2	1	1	0
	Delete *4	14	19	16	4	6	0	4	1	4	3
	Create *16	24	29	28	4	2	0	4	3	4	3
	Get Buffer *16	17	18	17	1	0	1	2	1	1	0
	Return Buffer *16	23	24	23	1	0	1	2	1	1	0
	Delete *16	14	19	17	4	5	0	4	1	4	3
Tps	Get	13	14	13	1	0	0	1	0	1	0
	Pause	28	29	29	1	0	1	2	2	1	0
	Set	30	31	30	5	0	0	1	0	1	0
...											

Test		MBX 821					MVME 2700				
		Min	**Max**	**Avg**	**Err**	**Sdv**	**Min**	**Max**	**Avg**	**Err**	**Sdv**
Evts	Send	15	16	15	1	0	0	1	0	1	0
	Send Ready	19	24	22	4	5	0	4	1	4	4
	Receive No Wait	14	15	14	1	0	0	1	0	1	0
	Send/Recv	27	28	27	1	0	1	2	1	1	0
	Recv	12	13	12	2	0	0	1	0	2	0
	Recv Wait	34	39	34	4	0	4	9	4	4	2
	Recv/Send Preempt	51	52	51	1	0	3	4	3	1	0
	Send Preempt	12	18	17	5	0	0	0	0	5	0
Sém	Create	14	19	17	4	5	0	4	1	4	3
	Delete	14	19	16	4	7	0	4	0	4	3
	P	17	18	17	0	0	0	1	0	0	0
	V	14	15	14	0	0	0	1	0	0	0
	V Ready	14	19	18	4	1	0	4	1	4	3
	P Wait	39	44	39	4	2	4	9	4	4	2
	P/V Preempt	49	54	53	4	2	0	4	2	4	4
	V Preempt	5	15	14	8	6	0	0	0	8	0
Files Messages	Create	19	24	20	4	6	0	4	1	4	4
	Delete	19	24	20	4	6	0	4	1	4	4
	Send	22	23	22	1	0	1	2	1	1	0
	Send Ready	24	29	24	4	0	0	4	1	4	4
	Recv	22	23	22	1	0	1	2	1	1	0
	Recv Wait	39	44	39	4	2	4	9	4	4	3
	Recv/Send Preempt	60	62	60	1	1	3	4	3	1	0
	Send Preempt	16	23	21	5	2	0	0	0	5	0
	Broadcast	14	15	14	1	0	0	1	0	1	0
...											

Test		MBX 821					MVME 2700				
		Min	**Max**	**Avg**	**Err**	**Sdv**	**Min**	**Max**	**Avg**	**Err**	**Sdv**
	Urgent	22	23	22	1	0	1	2	1	1	0
Files Messages Var	Create	58	58	58	4	0	0	4	2	4	4
	Delete	24	29	26	4	7	0	4	1	4	4
	Create *4	58	63	58	4	3	0	4	2	4	4
	Delete *4	24	29	28	4	2	0	4	1	4	4
	Create *16	58	58	58	4	0	0	4	3	4	2
	Delete *16	24	29	26	4	6	0	4	2	4	4
	Send	20	22	22	1	0	1	2	1	1	0
	Send Ready	19	19	19	4	0	0	4	1	4	4
	Recv	18	19	18	1	0	1	2	1	1	0
	Recv Wait	34	39	38	4	4	4	9	5	4	4
	Recv/Send Preempt	60	62	60	1	0	3	4	3	1	0
	Send Preempt	21	26	22	5	3	0	0	0	5	0
	Broadcast	13	14	13	1	1	0	1	0	1	0
	Urgent	19	19	19	1	0	1	2	1	1	0
	Send *4	22	23	22	1	0	1	2	1	1	0
	Recv *4	19	20	19	1	0	1	2	1	1	0
	Broadcast *4	13	14	13	1	0	1	1	1	1	0
	Urgent *4	19	20	19	1	0	1	2	1	1	0
	Send Ready *4	19	24	22	4	6	0	4	1	4	4
	Recv Wait *4	34	39	37	4	5	4	9	4	4	3
	Recv/Send Preempt *4	61	63	61	1	0	3	4	3	1	0
	Send Preempt *4	22	27	23	5	5	0	0	0	5	0
	Send *16	25	25	25	1	0	1	2	1	1	0
	Recv *16	23	23	23	1	0	1	2	1	1	0
...											

Test		MBX 821					MVME 2700				
		Min	**Max**	**Avg**	**Err**	**Sdv**	**Min**	**Max**	**Avg**	**Err**	**Sdv**
	Broadcast *16	13	13	13	1	0	0	1	0	1	0
	Urgent *16	24	24	24	1	0	1	2	1	1	0
	Send Ready *16	24	24	24	4	0	0	4	2	4	4
	Recv Wait *16	34	39	36	4	6	4	9	4	4	3
	Recv/Send Preempt *16	64	66	64	1	0	3	4	3	1	0
	Send Preempt *16	25	31	28	5	5	0	0	0	5	0
Réseau	Create/Close	343	344	343	1	0	✗	✗	✗	✗	✗
	Client TCP 1	6650	6700	6687	50	493	✗	✗	✗	✗	✗
	Client TCP 1Z	6650	6750	6697	50	125	✗	✗	✗	✗	✗
	Client TCP 2	9700	9800	9760	50	421	✗	✗	✗	✗	✗
	Client TCP 3	31200	31500	31262	50	2598	✗	✗	✗	✗	✗
	Client TCP 4	52300	52450	52350	50	1578	✗	✗	✗	✗	✗
	Client UDP 1	1500	1600	1560	50	421	✗	✗	✗	✗	✗
	Client UDP 1Z	1500	1600	1565	50	552	✗	✗	✗	✗	✗
	Client UDP 2	3750	3800	3780	50	631	✗	✗	✗	✗	✗
	Client UDP 3	24450	24550	24507	50	598	✗	✗	✗	✗	✗
	Client UDP 4	44400	44550	44542	50	335	✗	✗	✗	✗	✗
	Client UDP 4Z	32350	32500	32482	50	862	✗	✗	✗	✗	✗
	Server UDP 1	1600	1650	1647	50	125	✗	✗	✗	✗	✗
	Server UDP 1Z	1600	1650	1637	50	493	✗	✗	✗	✗	✗
	Server UDP 2	3700	3800	3755	50	236	✗	✗	✗	✗	✗
	Server UDP 3	20150	20200	20185	50	552	✗	✗	✗	✗	✗
	Server UDP 4	36050	36300	36075	50	657	✗	✗	✗	✗	✗
	Server UDP 4Z	27700	27800	27712	50	493	✗	✗	✗	✗	✗
	Server TCP 1	1750	1800	1775	50	657	✗	✗	✗	✗	✗
...											

Test	MBX 821					MVME 2700				
	Min	**Max**	**Avg**	**Err**	**Sdv**	**Min**	**Max**	**Avg**	**Err**	**Sdv**
Server TCP 1Z	1750	1800	1782	50	598	✗	✗	✗	✗	✗
Server TCP 2	4000	4050	4022	50	651	✗	✗	✗	✗	✗
Server TCP 3	199950	20045	20039	50	1012	✗	✗	✗	✗	✗
Server TCP 4	200100	20015	20012	50	657	✗	✗	✗	✗	✗
Server TCP 4Z	199200	20020	20017	50	651	✗	✗	✗	✗	✗

B.1.3 VxWorks

Test		MBX 821					MVME 2700				
		Min	**Max**	**Avg**	**Err**	**Sdv**	**Min**	**Max**	**Avg**	**Err**	**Sdv**
Tâches	Create	152	157	156	4	2	9	14	12	4	6
	Start	49	54	51	4	6	4	9	5	4	5
	Set Priority	16	17	16	0	0	0	1	0	0	0
	Suspend	14	19	16	4	6	0	4	0	4	2
	Resume	19	19	19	4	0	0	4	1	4	4
	Suspend/Resume	35	36	35	0	0	1	2	1	0	0
	Delete	49	54	49	4	0	0	4	3	4	2
	Context Switch	7	8	7	0	0	1	2	1	0	0
Mémoire	Create	24	29	24	4	3	0	4	1	4	4
	Delete	14	19	16	4	6	0	4	1	4	4
	Get Buffer	17	18	17	1	0	1	2	1	1	0
	Return Buffer	21	23	22	1	1	1	2	1	1	0
	Create *4	24	29	28	4	2	0	4	1	4	4
	Get Buffer *4	17	18	17	1	0	1	2	1	1	0
	Return Buffer *4	21	23	22	1	1	1	2	1	1	0
	Delete *4	14	19	16	4	6	0	4	1	4	3
	Create *16	24	29	28	4	2	0	4	3	4	3
	Get Buffer *16	17	18	17	1	0	1	2	1	1	0
	Return Buffer *16	23	24	23	1	0	1	2	1	1	0
	Delete *16	14	19	17	4	5	0	4	1	4	3
Tps	Get	13	14	13	1	0	0	1	0	1	0
	Pause	28	29	29	1	0	1	2	2	1	0
	Set	30	31	30	5	0	0	1	0	1	0
...											

Test		MBX 821					MVME 2700				
		Min	**Max**	**Avg**	**Err**	**Sdv**	**Min**	**Max**	**Avg**	**Err**	**Sdv**
Evts	Send	15	16	15	1	0	0	1	0	1	0
	Send Ready	19	24	22	4	5	0	4	1	4	4
	Recv No Wait	14	15	14	1	0	0	1	0	1	0
	Send/Recv	27	28	27	1	0	1	2	1	1	0
	Recv	12	13	12	2	0	0	1	0	2	0
	Recv Wait	34	39	34	4	0	4	9	4	4	2
	Recv/Send Preempt	51	52	51	1	0	3	4	3	1	0
	Send Preempt	12	18	17	5	0	0	0	0	5	0
Sém	Create	14	19	17	4	5	0	4	1	4	3
	Delete	14	19	16	4	7	0	4	0	4	3
	P	17	18	17	0	0	0	1	0	0	0
	V	14	15	14	0	0	0	1	0	0	0
	V Ready	14	19	18	4	1	0	4	1	4	3
	P Wait	39	44	39	4	2	4	9	4	4	2
	P/V Preempt	49	54	53	4	2	0	4	2	4	4
	V Preempt	5	15	14	8	6	0	0	0	8	0
Files Messages	Create	19	24	20	4	6	0	4	1	4	4
	Delete	19	24	20	4	6	0	4	1	4	4
	Send	22	23	22	1	0	1	2	1	1	0
	Send Ready	24	29	24	4	0	0	4	1	4	4
	Recv	22	23	22	1	0	1	2	1	1	0
	Recv Wait	39	44	39	4	2	4	9	4	4	3
	Recv/Send Preempt	60	62	60	1	1	3	4	3	1	0
	Send Preempt	16	23	21	5	2	0	0	0	5	0
	Broadcast	14	15	14	1	0	0	1	0	1	0
...											

Test		MBX 821					MVME 2700				
		Min	**Max**	**Avg**	**Err**	**Sdv**	**Min**	**Max**	**Avg**	**Err**	**Sdv**
	Urgent	22	23	22	1	0	1	2	1	1	0
Files Messages Var	Create	58	58	58	4	0	0	4	2	4	4
	Delete	24	29	26	4	7	0	4	1	4	4
	Create *4	58	63	58	4	3	0	4	2	4	4
	Delete *4	24	29	28	4	2	0	4	1	4	4
	Create *16	58	58	58	4	0	0	4	3	4	2
	Delete *16	24	29	26	4	6	0	4	2	4	4
	Send	20	22	22	1	0	1	2	1	1	0
	Send Ready	19	19	19	4	0	0	4	1	4	4
	Recv	18	19	18	1	0	1	2	1	1	0
	Recv Wait	34	39	38	4	4	4	9	5	4	4
	Recv/Send Preempt	60	62	60	1	0	3	4	3	1	0
	Send Preempt	21	26	22	5	3	0	0	0	5	0
	Broadcast	13	14	13	1	1	0	1	0	1	0
	Urgent	19	19	19	1	0	1	2	1	1	0
	Send *4	22	23	22	1	0	1	2	1	1	0
	Recv *4	19	20	19	1	0	1	2	1	1	0
	Broadcast *4	13	14	13	1	0	1	1	1	1	0
	Urgent *4	19	20	19	1	0	1	2	1	1	0
	Send Ready *4	19	24	22	4	6	0	4	1	4	4
	Recv Wait *4	34	39	37	4	5	4	9	4	4	3
	Recv/Send Preempt *4	61	63	61	1	0	3	4	3	1	0
	Send Preempt *4	22	27	23	5	5	0	0	0	5	0
	Send *16	25	25	25	1	0	1	2	1	1	0
	Recv *16	23	23	23	1	0	1	2	1	1	0
...											

Test		MBX 821					MVME 2700				
		Min	**Max**	**Avg**	**Err**	**Sdv**	**Min**	**Max**	**Avg**	**Err**	**Sdv**
	Broadcast *16	13	13	13	1	0	0	1	0	1	0
	Urgent *16	24	24	24	1	0	1	2	1	1	0
	Send Ready *16	24	24	24	4	0	0	4	2	4	4
	Recv Wait *16	34	39	36	4	6	4	9	4	4	3
	Recv/Send Preempt *16	64	66	64	1	0	3	4	3	1	0
	Send Preempt *16	25	31	28	5	5	0	0	0	5	0
Réseau	Create/Close	343	344	343	1	0	✗	✗	✗	✗	✗
	Client TCP 1	6650	6700	6687	50	493	✗	✗	✗	✗	✗
	Client TCP 1Z	6650	6750	6697	50	125	✗	✗	✗	✗	✗
	Client TCP 2	9700	9800	9760	50	421	✗	✗	✗	✗	✗
	Client TCP 3	31200	31500	31262	50	2598	✗	✗	✗	✗	✗
	Client TCP 4	52300	52450	52350	50	1578	✗	✗	✗	✗	✗
	Client UDP 1	1500	1600	1560	50	421	✗	✗	✗	✗	✗
	Client UDP 1Z	1500	1600	1565	50	552	✗	✗	✗	✗	✗
	Client UDP 2	3750	3800	3780	50	631	✗	✗	✗	✗	✗
	Client UDP 3	24450	24550	24507	50	598	✗	✗	✗	✗	✗
	Client UDP 4	44400	44550	44542	50	335	✗	✗	✗	✗	✗
	Client UDP 4Z	32350	32500	32482	50	862	✗	✗	✗	✗	✗
	Server UDP 1	1600	1650	1647	50	125	✗	✗	✗	✗	✗
	Server UDP 1Z	1600	1650	1637	50	493	✗	✗	✗	✗	✗
	Server UDP 2	3700	3800	3755	50	236	✗	✗	✗	✗	✗
	Server UDP 3	20150	20200	20185	50	552	✗	✗	✗	✗	✗
	Server UDP 4	36050	36300	36075	50	657	✗	✗	✗	✗	✗
	Server UDP 4Z	27700	27800	27712	50	493	✗	✗	✗	✗	✗
	Server TCP 1	1750	1800	1775	50	657	✗	✗	✗	✗	✗
...											

Test	MBX 821					MVME 2700				
	Min	**Max**	**Avg**	**Err**	**Sdv**	**Min**	**Max**	**Avg**	**Err**	**Sdv**
Server TCP 1Z	1750	1800	1782	50	598	✗	✗	✗	✗	✗
Server TCP 2	4000	4050	4022	50	651	✗	✗	✗	✗	✗
Server TCP 3	199950	20045	20039	50	1012	✗	✗	✗	✗	✗
Server TCP 4	200100	20015	20012	50	657	✗	✗	✗	✗	✗
Server TCP 4Z	199200	20020	20017	50	651	✗	✗	✗	✗	✗

Annexe C

Ingénierie Logicielle

C.1 Introduction

Les deux prototypes OpenTAZ et OpenTAZ-CCM sont conçus en technologie CORBA. Le premier est constitué d'un client et d'un serveur alors que le second est constitué d'un client et d'un serveur lui même décomposé en plusieurs composants.

La présentation de l'architecture externe de ces prototypes a déjà été faite dans les chapitres 5 et 6.

Nous nous attachons ici à présenter l'architecture interne de ces prototypes en terme de classes.

C.2 Architecture d'OpenTAZ

OpenTAZ est un logiciel conçu en objet, nous décrivons ci-dessous les principales classes qui le composent, au moyen de descriptions et de diagrammes de classes UML.

C.2.1 Le système de gestion des événements

Le système de gestion des événements dans OpenTAZ est dérivé du design pattern *Publish-Subscribe*, il gère tous les événements du serveur. Il est composé des différentes classes suivantes :

❶ **EventMonitor** : Le moniteur des événements est le centre du système. Il réceptionne les événements (*eventArrived()* équivalent au *publish()*) produits et les redispatche vers les objets qui en ont fait la demande (*subscribe()/unsubscribe()*).

❷ **Event** : Un Event modélise les objets événements. Il exporte une méthode (*match()*) qui permet de savoir si un événement rentre dans les critères

d'une demande faite par un abonné. Ceci permet de raffiner le modèle ou l'on s'abonne à tous les événements d'un type donné. Par exemple, on peut alors s'abonner à un événement *Clock* correspondant à une date précise.

❸ **EventSubscriber** : C'est une interface que doivent implémenter tous les abonnés à un événement auprès de l'*EventMonitor*. Elle contient une méthode (*callback()*) qui est appelée lors de l'arrivée d'un événement correspondant à la demande.

❹ **EventRequest** : Permet à l'*EventSubscriber* de raffiner sa demande et de spécifier seulement un sous-ensemble des événements d'un type donné.

❺ **Clock** : *Clock* est un générateur d'événements de type "top d'horloge" et qu'il envoie directement à l'*EventMonitor*.

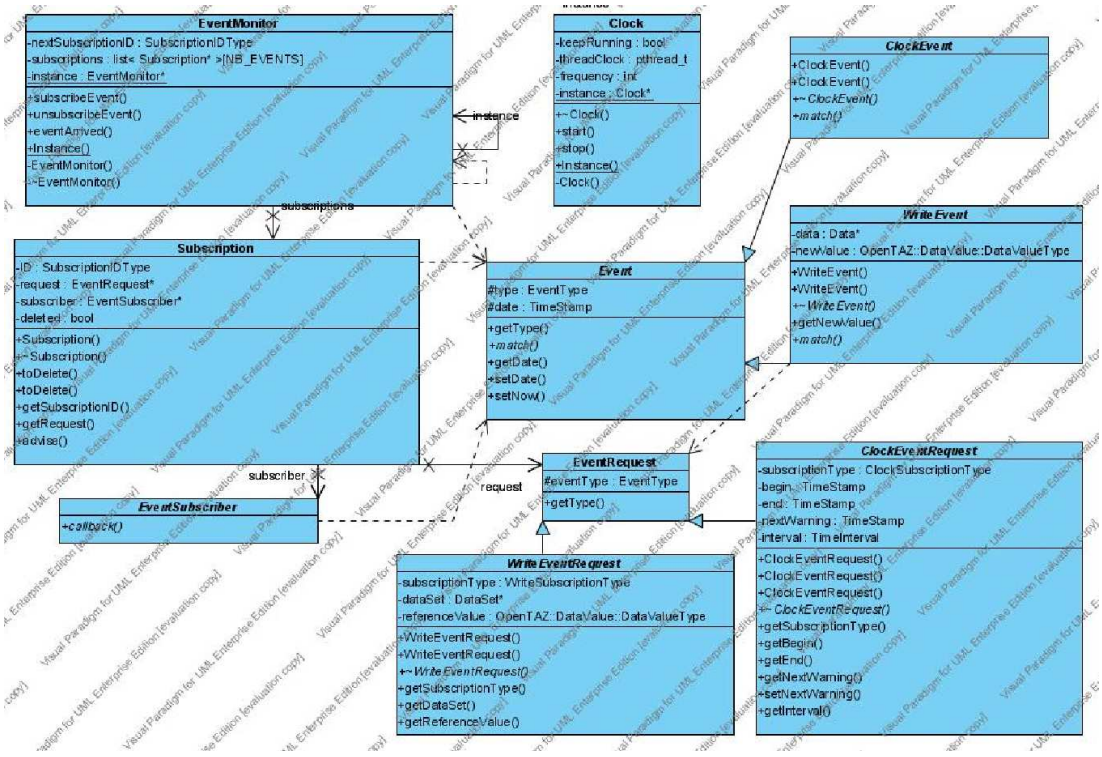

FIG. C.1: Classes d'OpenTAZ - Events

C.2.2 Autres classes

Un objet Database permet de gérer tous les objets de données exportés par un serveur, qu'ils soient de type *Device* ou *Data*.

Les autres classes de données suivent au plus près les descriptions de TASE.2, ainsi nous avons des objets *Device*, *Data*, *DataSet*, *TransferSet* et *DataSetTransferSet*.

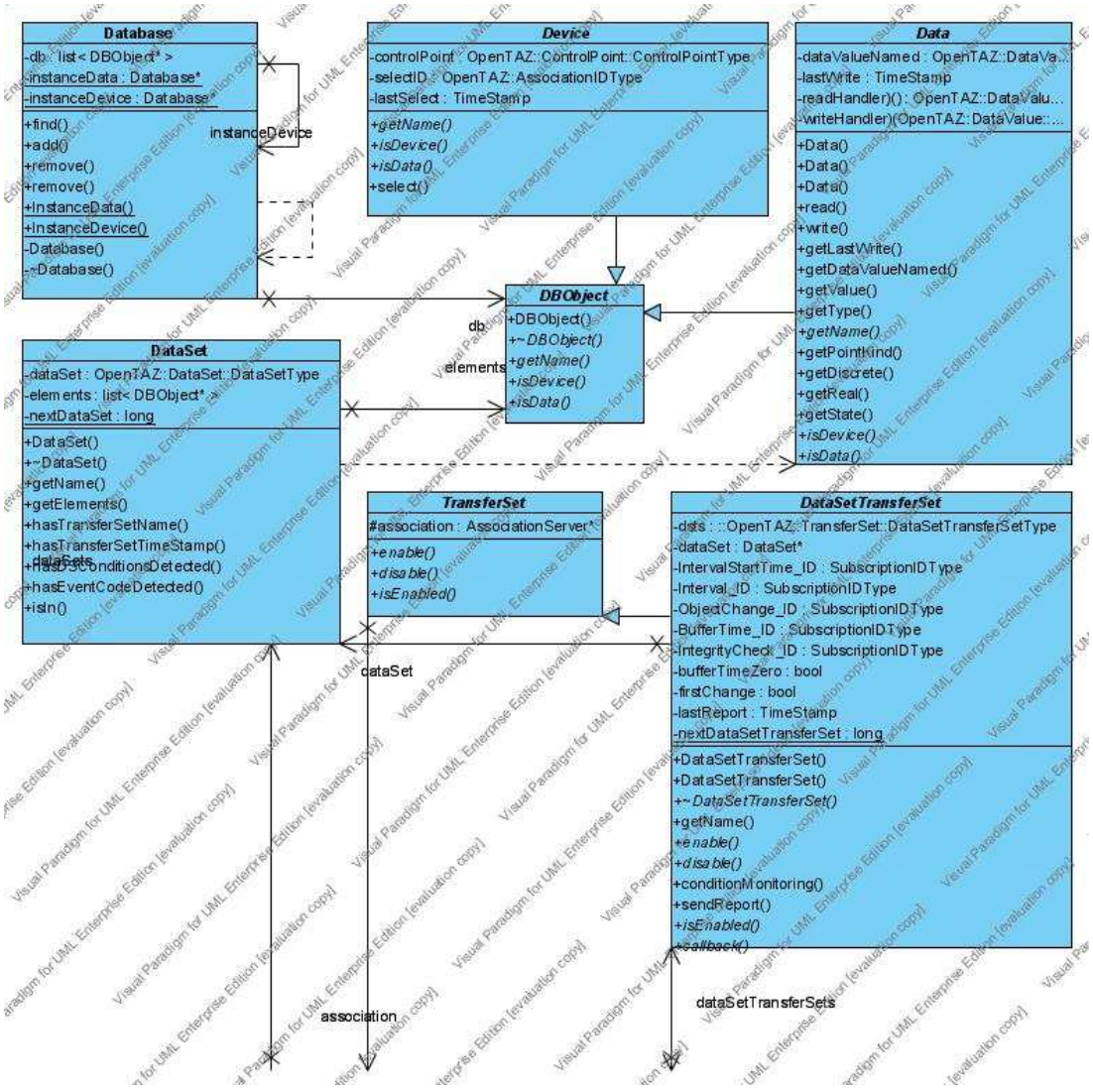

FIG. C.2: Classes d'OpenTAZ - Datas

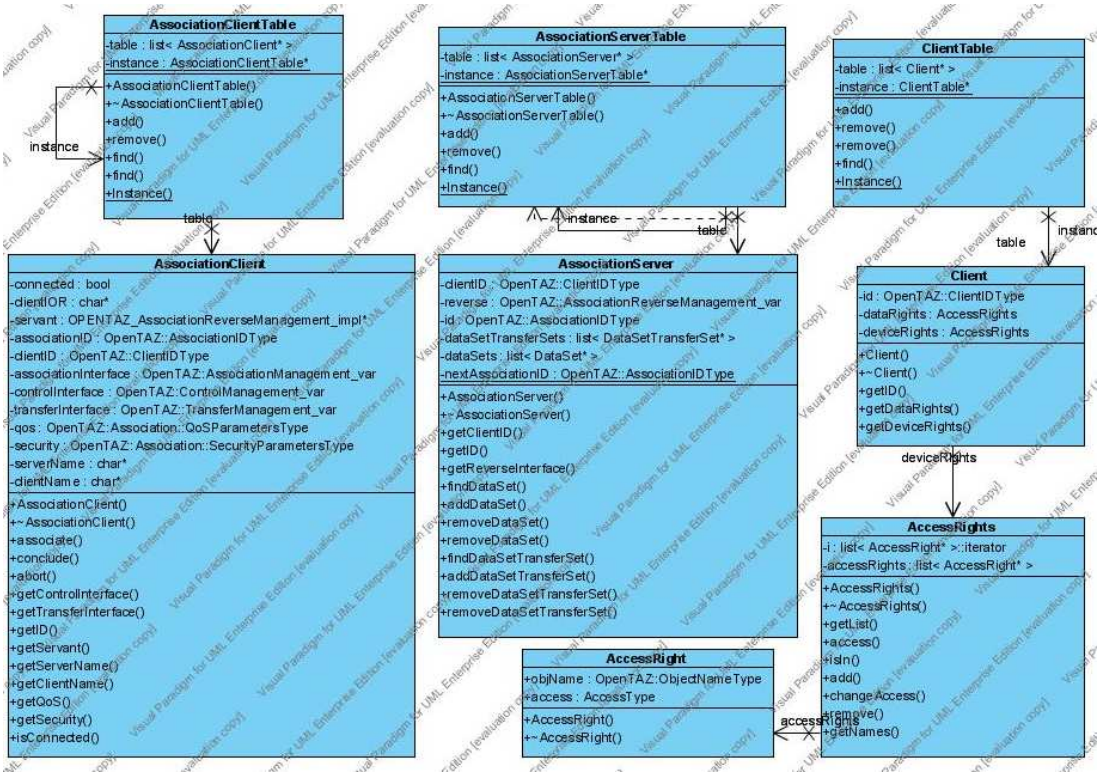

FIG. C.3: Classes d'OpenTAZ - Autres

Des classes permettent d'encapsuler les concepts d'association côté serveur et côté client et de client (dans le serveur). A chaque fois une classe Singleton permet de gérer toutes les instances.

Enfin, la classe *MyORB* est une classe qui encapsule toutes les spécificités de notre implémentation CORBA (Mico) et dont le but est de faciliter des portages éventuels vers d'autres implémentations CORBA (TAO par exemple).

C.3 Architecture d'OpenTAZ-CCM

OpenTAZ-CCM est une évolution du prototype OpenTAZ. Nous nous attachons donc à décrire les différences avec OpenTAZ en termes de classes puisqu'il en reprend une grande partie. Les différences sont dues essentiellement au passage en composants qui font intervenir de nouvelles classes.

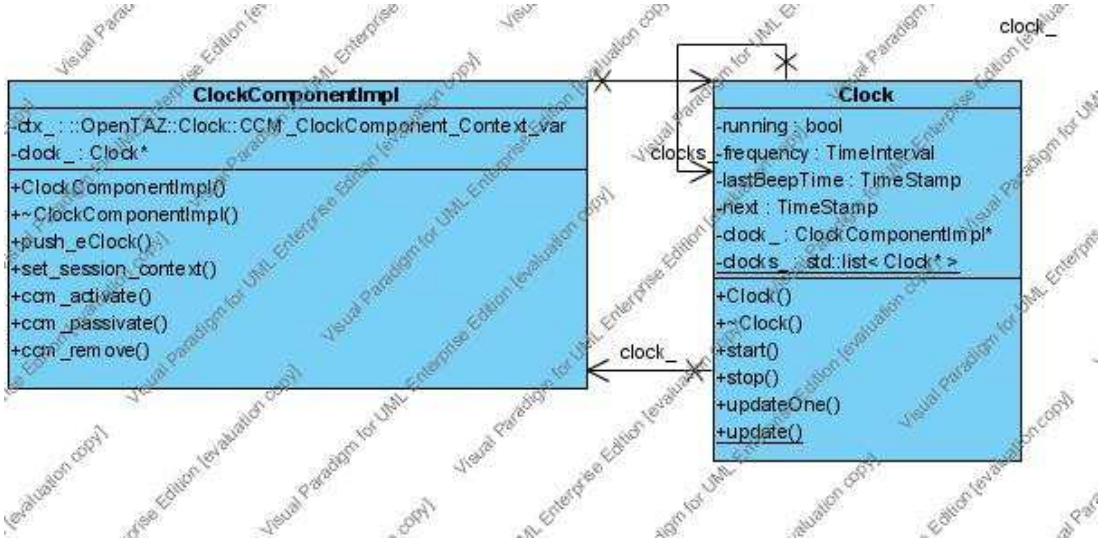

FIG. C.4: Classes d'OpenTAZ-CCM - Composant Clock

- Le système d'événements développé pour OpenTAZ a été amélioré pour gérer également les événements CORBA. Ainsi les événements envoyés par les composants *Database* et *Clock* (voir la suite) sont réceptionnés dans le serveur par l'*EventMonitor* et dispatchés dans le reste du serveur.
- L'ancien objet *Clock* est maintenant un composant plutôt qu'un objet. Il envoie des événements CORBA plutôt que des événements intra-processus dans OpenTAZ. La fonctionnalité reste la même.
- La gestion des données est modifiée car la classe gérant les données est maintenant un composant. Des classes proxy (*xxxShadow*) sont donc ajoutées dans le serveur pour faciliter l'accès au composant en exportant la

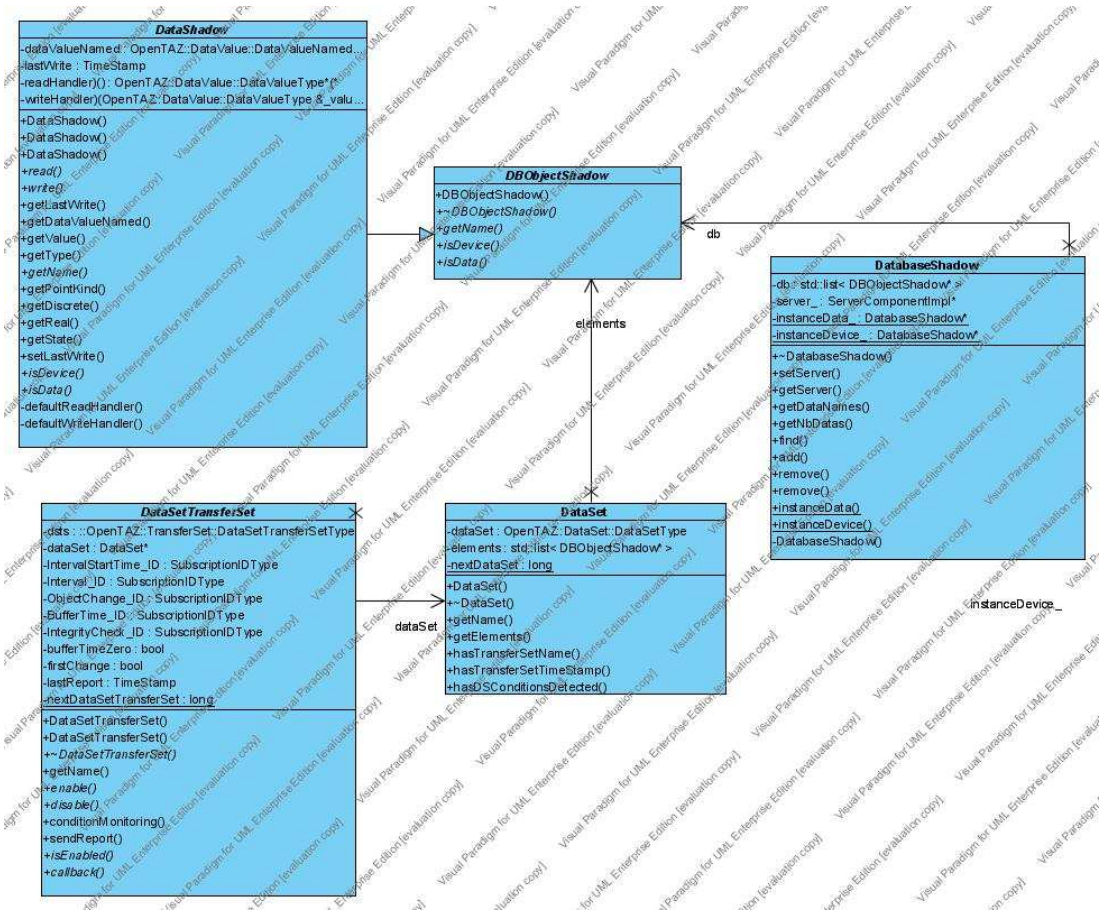

FIG. C.5: Classes d'OpenTAZ-CCM - Database

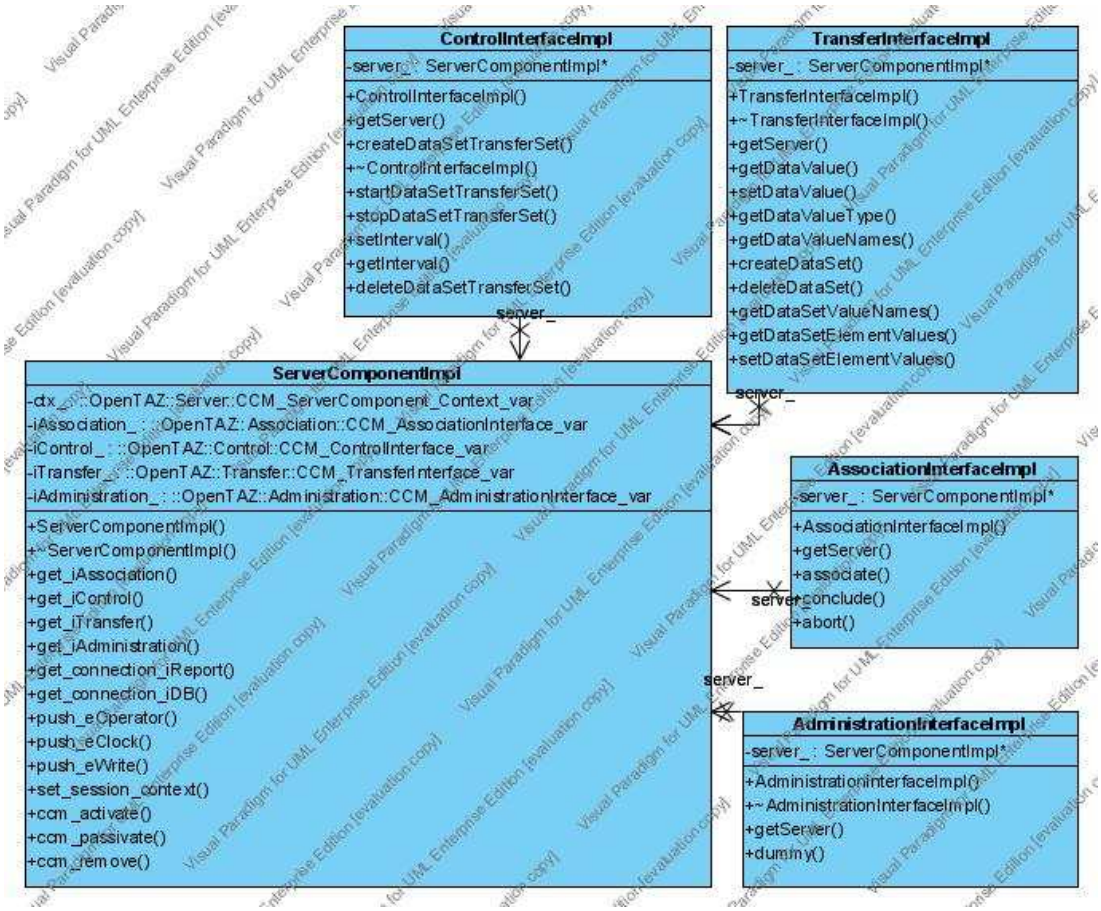

FIG. C.6: Classes d'OpenTAZ-CCM - Serveur

même interface que l'ancien objet d'OpenTAZ et ainsi maximiser la réutilisation du code. Le composant envoie des événements *Write* au serveur pour lui indiquer une écriture sur une donnée (et donc déclencher éventuellement l'envoi d'un transfer report).

- ➡ Un objet Serveur permet d'implémenter le composant CORBA correspondant. Il encapsule les implémentations des interfaces du serveur (Control, Transfer, Association et Administration) et réceptionne les événements en provenance des composants *Clock* et *Database*.
- ➡ Un gestionnaire de traces a été ajouté pour faciliter le déverminage.