



European Organization for
Nuclear Research



Institut National Polytechnique
de Grenoble



Département
Télécommunications
ENSIMAG - ENSERG

PROJET DE TROISIEME ANNEE

RAPPORT FINAL ANNEE SCOLAIRE 2002/2003

Auteur :

Pierre-Alain Doffoel

pa@doffoel.com

Option Applications Réparties & Réseaux

TITRE DU PROJET :

Monitoring of a Transatlantic Gigabit Network

Proposé par l'Entreprise :

CERN

Centre Européen de Recherche Nucléaire

Adresse :

CH – 1211 Genève 23

Nom et Prénom du Responsable :

J.P. Martin-Flatin

jp.martin-flatin@cern.ch

CERN – IT Division

CH – 1211 Genève 23

Nom et Prénom du Tuteur INPG :

Ingrid Krause-Mussig

Ingrid.Krause-Mussig@imag.fr

***Ecole Nationale Supérieure d'Electronique
et de Radioélectricité de Grenoble***

23 avenue des Martyrs - BP 257

38016 GRENOBLE Cedex 1 – France

Tél : (33) 04.76.85.60.00 - Fax : (33) 04.76.85.60.60

***Département Télécommunications
ENSERG – ENSIMAG***

681 rue de la Passerelle – Domaine Universitaire – BP 72

38402 ST MARTIN D'HERES Cedex – France

Tél : (33) 04.76.82.72.22 – Fax : (33) 04.76.82.72.50

Remerciements :

Mon projet de fin d'études s'est déroulé au CERN, Centre Européen de Recherche Nucléaire, situé à Genève en Suisse.

Je tiens à remercier tout particulièrement Jean-Philippe Martin-Flatin, Technical Manager du projet européen DataTAG, pour avoir accepté de me diriger patiemment, pour son aide et ses encouragements permanents, source de motivation inépuisable.

Je tiens également à exprimer ma reconnaissance à Ingrid Krause-Mussig pour son soutien régulier ainsi qu'à tous les membres du projet DataTAG pour m'avoir exposé leur travail avec passion et introduit dans ce cadre de recherche exceptionnel.

Résumé

La complexité et l'hétérogénéité des réseaux IP impose l'utilisation d'outils de gestion de réseaux. Aujourd'hui c'est encore le protocole SNMP qui est le plus communément utilisé. Et pourtant, avec la croissance des technologies Web et des applications distribuées, les manques de ce protocole sont de plus en plus évidents : il paraît indispensable de concevoir de nouveaux systèmes de gestion robustes et automatisés. Durant ce stage, nous avons poursuivi le développement de JAMAP, un prototype de recherche qui implémente l'architecture WIMA décrite par J-P Martin-Flatin dans sa thèse de doctorat. Ce travail a donné naissance à la version 1.3 de JAMAP, insistant sur les aspects de distribution et d'automatisation.

Abstract

Nowadays IP networks are becoming larger and more complex, this has led to the need for robust management tools. Today SNMP is still the most commonly used to monitor networks. However, with the growth of internet technologies as well as distributed applications, the shortcomings of this protocol have become more and more obvious and it is now essential to create new automated network management systems. In this project, we further developed the Java Management Platform JAMAP, a research prototype that implements the WIMA architecture designed by J-P Martin-Flatin in his Ph.D. Thesis. This work leads to the release 1.3 of JAMAP, that focuses on distribution and automation aspects.

Liste des abréviations :

API	Application Programming Interface
CERN	Centre Européen de Recherche Nucléaire
CIM	Common Information Model
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
DCOM	Distributed Component Object Model
DMTF	Distributed Management Task Force
EPFL	Ecole Polytechnique Fédérale de Lausanne
FTP	File Transfer Protocol
GUI	Graphic User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISO	International Organization for Standardization
IT	Information Technology
ITU-T	International Telecommunication Union
J2EE	Java 2 Enterprise Edition
J2SE	Java 2 Standard Edition
JAMAP	Java Management Platform
JDBC	Java Database Connectivity
JDK	Java Development Kit
JVM	Java Virtual Machine
LAN	Local-Area Network
LHC	Large Hadron Collider
MIB	Management Information Base
MIME	Multipurpose Internet Mail Extensions
MTU	Maximal Transmission Unit
NIM	Network Information Model
NSM	Network and Systems Management
OID	Object Identifier
OSI	Open Systems Interconnection
PC	Personnal Computer
PDU	Packet Data Unit
PING	Packet Internet Groper
QoS	Quality of Service
R&D	Research and Development
RFC	Request For Comments
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TCP	Transmission Control Protocol
UDDI	Universal description, Discovery and Integration
UML	Unified Modeling Language
UNESCO	United Nations Educational, Scientific and Cultural Organization
URL	Uniform Resource Locator
WAN	Wide-Area Network
WIMA	Web-based Integrated Management Architecture
WSDL	Web Services Description Language
WWW	World Wide Web
XML	eXtensible Markup Language

Liste des figures :

Figure 1 : Etats membres du CERN	9
Figure 2 : Estimation de la capacité CPU requise au CERN[4].....	11
Figure 3 : Architecture distribuée de JAMAP	16
Figure 4 : modèle Pull	17
Figure 5 : modèle Push.....	18
Figure 6 : Réponse infinie en utilisant le type MIME.....	19
Figure 7 : Architecture utilisée dans JAMAP version 1.3.....	22
Figure 8 : Modèle prévoyant un ordonnanceur pour chaque collecteur de données.....	26
Figure 9 : Un ordonnanceur responsable de l'intégralité des collecteurs de données.....	27
Figure 10 : Implémentation du PushScheduler sur un agent.....	28
Figure 11 : Ajout de souscription au PushScheduler	29
Figure 12 : Implémentation finale des PushSchedulers au niveau de l'agent.....	30
Figure 13 : Organisation hiérarchique des composants dans un registre UDDI	32

Sommaire

1	<i>Contexte du PFE</i>	9
1.1	Historique du CERN	9
1.2	Personnel au CERN	9
1.3	Le projet LHC	10
1.4	IT Division	10
1.5	Le projet DataTAG	11
2	<i>La gestion de réseaux (Network Management)</i>	13
2.1	Pourquoi ?	13
2.2	ICMP : Internet Control Message Protocol	13
2.3	SNMP : Simple Network Management Protocol	13
3	<i>JAMAP, à la base du PFE</i>	14
3.1	Historique de JAMAP : Java Management Platform	14
3.2	Architecture distribuée de JAMAP	15
3.3	JAMAP et les modèles d'information des agents	16
3.4	JAMAP et l'utilisation du modèle Push	17
3.5	JAMAP et l'utilisation du protocole HTTP pour transférer les données entre agents et managers	18
3.6	JAMAP et XML	19
3.7	JAMAP et les technologies Web	19
3.7.1	Les Applets Java	19
3.7.2	Les servlets	20
3.7.3	Les services Web ou « Web Services »	20
4	<i>Etude comparative de solutions : nouveau design de l'ordonnanceur coté agent</i>	22
4.1	La gestion du temps dans une application orientée objets	23
4.1.1	Solution basique : utilisation de boucles.....	23
4.1.2	Solution rapide : intégrer sa logique dans une thread et la faire dormir	23
4.1.3	Utilisation de sémaphores ?	23
4.2	Optimisation de l'ordonnanceur coté agent	25
4.2.1	Solution basique : un ordonnanceur pour un collecteur de données.....	25
4.2.2	Un unique ordonnanceur pour tous les collecteurs de données	27
4.3	Ajout du concept de proxy	29
5	<i>Etude comparative de solutions : découverte dynamique des agents et de leurs caractéristiques</i>	31
5.1	Présentation de UDDI	31
5.1.1	Spécification technique.....	31
5.1.2	Finalité.....	32
5.2	Utilisation de registres UDDI dans JAMAP	32

5.2.1	Informations à découvrir dynamiquement	32
5.2.2	Incompatibilité des registres UDDI avec JAMAP	33
5.3	XML pour configurer JAMAP	33
5.3.1	La carte du réseau dans un fichier networkMap.xml	33
5.3.2	Les informations relatives à un agent donné dans un fichier agentManagement.xml	35
5.3.3	Localisation des fichiers de configuration.....	36
6	Conclusions	37
6.1	Résumé et contributions	37
6.2	Travail futur	38
7	Références :	39

J'ai eu la chance de réaliser mon projet de fin d'études dans le plus grand centre de physique des particules au monde : le **CERN**, Centre Européen de Recherche Nucléaire, situé à Genève en Suisse. Chaque année, des milliers de physiciens du monde entier y viennent pour explorer ce dont la matière est faite et observer les forces qui assurent sa cohésion. Le CERN existe avant tout pour leur fournir les outils nécessaires, à savoir des accélérateurs, qui accélèrent les particules jusqu'à une vitesse proche de celle de la lumière, des détecteurs pour rendre les particules visibles et des ressources informatiques gigantesques afin d'exploiter leurs données.

1 Contexte du PFE

Cette partie présente le cadre général de ce projet de fin d'études et s'appuie sur les données officielles du CERN disponibles à l'adresse <http://www.cern.ch>.

1.1 Historique du CERN

Fondé en 1954 sous les auspices de l'Unesco, le CERN est la plus ancienne des grandes organisations de recherche européennes. Elle comprend **20 états membres**, qui ont des devoirs et privilèges spéciaux : ils contribuent au capital et frais d'exploitation des programmes de recherche, et sont représentés au Conseil, responsable de toutes les décisions importantes concernant l'organisation et ses activités. Certains états ou organisations internationales pour lesquels l'adhésion n'est pas encore réalisable participent en tant qu'**observateurs** (comme la Commission Européenne, l'UNESCO, les Etats Unis[1]). Ce statut leur permet d'assister aux réunions du Conseil et d'en recevoir les documents, sans pour autant y participer. De plus l'organisation accueille des scientifiques des 220 instituts d'états non membres, qui désirent s'impliquer dans certains projets et participent à leur financement.



Figure 1 : Etats membres du CERN

1.2 Personnel au CERN

Le CERN emploie un peu moins de 3000 personnes, hommes et femmes couvrant un large éventail de compétences et de métiers : physiciens, ingénieurs, techniciens, ouvriers qualifiés, administrateurs, secrétaires, etc. Le personnel scientifique et technique conçoit et construit l'appareillage sophistiqué du laboratoire et en assure le bon fonctionnement. Il

contribue également à la préparation et à la mise en oeuvre des expériences scientifiques complexes ainsi qu'à l'analyse et à l'interprétation des résultats.

En plus du personnel fixe près de 6500 scientifiques, soit la moitié des physiciens et physiciennes des particules dans le monde, utilisent les installations de l'organisation : ainsi environ **6000 personnes** sont constamment présentes sur le site, qui ressemble d'avantage à une petite ville qu'à une grande entreprise (plusieurs restaurants, banques, magasins, tunnel privé entre la France et la Suisse, comité d'entreprise dynamique, école d'été etc.)

1.3 Le projet LHC

Le CERN investit une part importante de son budget dans la construction de nouvelles machines et est actuellement en phase de transition avec la construction du **collisionneur de hadrons LHC** (Large Hadron Collider, accélérateur de particules), qui sondera la matière plus profondément que jamais. Prévu pour démarrer en 2007, il permettra à terme des collisions de faisceaux de protons à une énergie de 14 TeV. Des faisceaux de noyaux de plomb seront également accélérés, entrant en collision avec une énergie de 1150 TeV. Un équipement de ce genre sera unique au monde.

En voici les caractéristiques[2] :

- Grand collisionneur de hadrons : accélérateur de particules constitué d'un anneau de 27 km d'aimants supraconducteurs placé dans un tunnel à environ 120 m sous terre.
- Energie de chaque faisceau LHC : 7 téra-électronvolts (tension équivalente à environ 600 milliards de batteries de voiture).
- **Débit d'accumulation des données : 10 péta-octets par an (équivalent à environ 20 millions de CD-Rom).**
- **Puissance d'UC requise : plus de 4 millions de SPECint95 (équivalent à environ 100 000 PC d'aujourd'hui).**
- Débit du réseau local : près d'un téraoctet par seconde sur des dizaines de sites.
- Capacité du réseau longue distance : plusieurs gigabits par seconde vers des centaines de sites.
- Nombre de chercheurs travaillant sur le LHC dans le monde : environ 10 000.
- Nombre d'instituts participant au LHC : environ 1000.
- Nombre de pays dans le monde dont les chercheurs participent au LHC : plus de 50.

Seule une collaboration entre des dizaines de pays peut permettre la réalisation d'un tel projet, dont le budget est supérieur à 1 milliard d'euros.

1.4 IT Division

L'**IT division** du CERN compte plus de **600 personnes** et a une place stratégique dans le projet LHC. Elle est responsable de la construction des équipements de demain qui seront nécessaires pour interpréter les données issues du nouvel accélérateur, dont la mise en service est prévue pour 2007. Le LHC imposera une force de calcul phénoménale impliquant la réalisation de *la plus grande application de calcul intensif de cette décennie*[3]. La figure 2 exprime la puissance requise au CERN pour les années à venir :

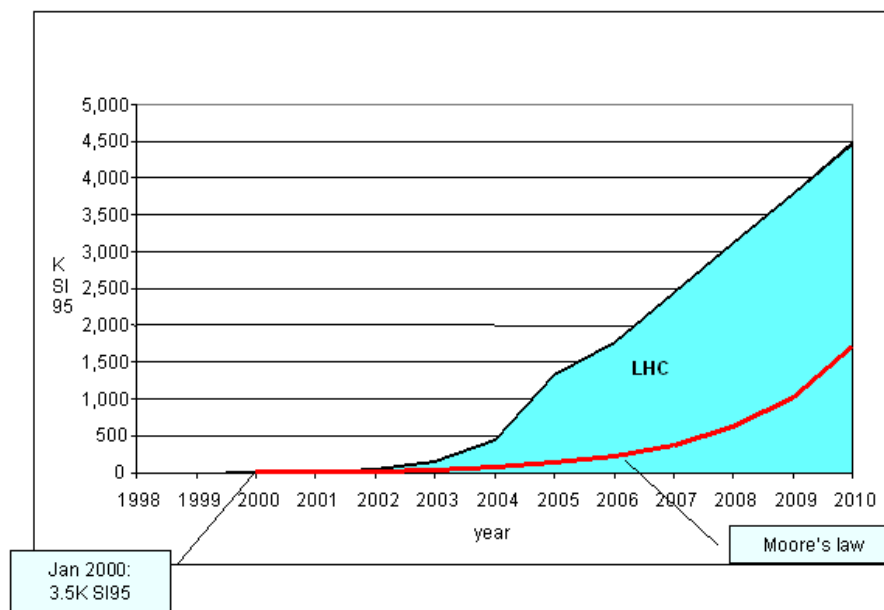


Figure 2 : Estimation de la capacité CPU requise au CERN[4]















Aux vues de cette figure il sera nécessaire de développer continuellement le parc informatique de l'organisation jusqu'en 2010, afin de pouvoir y effectuer **50% des calculs issus du LHC**. Le reste des calculs ne pourra être fait sur place pour des questions de budget mais le projet a prévu l'utilisation d'équipements étrangers en dispatchant le traitement des informations restantes dans d'autres pays : On parle alors de « **Grid Applications** » ou de « **grille de calcul** ». Cette philosophie s'inscrit dans la continuité des autres découvertes du CERN dans le domaine informatique. Rappelons en effet que le World Wide Web (**WWW**) avait été mis au point au CERN afin d'échanger l'information entre des collaborateurs travaillant dans des laboratoires et des universités disséminés dans le monde entier. Dans le cadre des Grid Applications le Web permet d'acheminer rapidement une information déjà traitée, tandis que les technologies des grilles de calcul donneront les moyens de rechercher et analyser les données dans le cadre d'un réseau mondial interconnecté de plusieurs dizaines de milliers d'ordinateurs et de mémoires. Cette capacité d'analyse accrue permettra aux chercheurs d'exploiter avec une bien plus grande efficacité les informations fournies par les détecteurs de particules.

1.5 Le projet DataTAG




L'IT Division du CERN regroupe plusieurs projets[5] : LHC Communication Infrastructure (COMIN), LHC Computing Grid (LCG), EU DataGrid (EDG), EU DataTag, EU EGEE, Joint Controls Project (JCOP).





J'ai été membre du projet **DataTAG** pendant 6 mois (<http://www.datatag.org>), dont le but est la création d'une grille de calcul intercontinentale et l'homogénéisation des environnements, afin d'exploiter au mieux possible les ressources. Ce projet inclut l'utilisation de réseaux hautes performances (liens gigabit entre l'Europe et les Etats-Unis notamment), la duplication fiable de données à grande vitesse, la réalisation de services réseaux avancés (QoS), et enfin l'utilisation de techniques innovantes pour observer les réseaux en temps réel (Network Monitoring).

Ce projet prévoit une collaboration entre de nombreux projets européens dont DataGrid et divers projets des Etats-Unis. Plus exactement les **centres de recherche** impliqués sont les suivants :

CERN : European Organisation for Nuclear Research	
INFN : Italian National Institute for Nuclear Research	
INRIA : Institut National de Recherche en Informatique et en Automatique	
PPARC	
University of Amsterdam	
California Institute of Technology	
StarLight	
Argonne National Laboratory	
Northwestern University	
University of Illinois at Chicago	
University of Michigan	
Stanford Linear Accelerator Center	
Fermi National Accelerator Laboratory	
Lawrence Berkeley National Laboratory	

Enfin les **réseaux de recherche** utilisés dans le cadre du projet sont les suivants :

CANARIE Inc.	
DANTE	
GÉANT	

ESnet	
Internet2 et Abilene	
MB-NG	
SURFnet	

Une collaboration d'une telle envergure n'est pas courante mais nécessaire dans le cadre d'une réalisation aussi ambitieuse que le LHC.

2 La gestion de réseaux (Network Management)

La gestion des réseaux IP a été en constante évolution ces dernières années.

2.1 Pourquoi ?

Avec l'émergence des **réseaux**, les systèmes informatiques sont devenus de plus en plus complexes. Ils sont désormais reliés entre eux et constituent le système nerveux des entreprises. Fiabilité, sécurité, disponibilité, performances, maintenabilité, évolutivité, ..., aucune qualité ne peut faire défaut.

Pour garantir une qualité de service maximale il est donc nécessaire de surveiller ces réseaux, à la base de l'interconnexion des systèmes : on parle de **gestion de réseaux** ou de «**Network Management**».

Au cours de l'histoire sont apparus 2 principaux protocoles dédiés à la gestion de réseaux, que nous allons présenter maintenant : ICMP et SNMP.

2.2 ICMP : Internet Control Message Protocol

Initialement le seul utilitaire d'administration de réseau utilisé était **ICMP** (Internet Control Message Protocol). Ce premier protocole créé en 1981 avait permis de créer le programme PING (Packet Internet Groper), qui permet de tester si un hôte peut être adressé, si un serveur est actif, et peut aider à déterminer des zones de congestion en indiquant le taux de datagrammes perdus. Mais avec la croissance exponentielle d'Internet il fût nécessaire d'élaborer un outil d'administration plus puissant, et c'est ainsi que le protocole SNMP fut approuvé.

2.3 SNMP : Simple Network Management Protocol

Le protocole **SNMP** a été développé par un groupe de travail de l'IETF[6] en 1987 pour répondre aux carences du protocole ICMP.

Il définit le dialogue entre une station d'administration et un noeud du réseau pour :

- connaître l'état d'une entité
- gérer les évènements exceptionnels
- mesurer le trafic et les erreurs à distance

Un **agent SNMP** est installé sur chaque noeud du réseau administrable. Ces noeuds peuvent être des **hôtes** (stations de travail ou serveurs), des **éléments d'interconnexion** (switch, hubs, routeurs), ou des **supports physiques** (câbles).

Cet agent enregistre en permanence des informations relatives à l'entité sur laquelle il est placé et stocke ces informations dans une base de données qui lui est propre appelée base d'informations de gestion : **MIB** (Management Information Base).

Les stations de management peuvent alors échanger des informations sur le réseau avec les agents à l'aide de **requêtes SNMP**.

SNMP propose 5 types de requêtes dans le cadre du "monitoring de réseaux":

- 2 pour lire les informations du nœud
- 2 pour configurer le nœud
- 1 pour gérer les événements exceptionnels du réseau (notifications)

Le protocole SNMP permet donc à un utilisateur de savoir si un noeud est connecté au réseau en envoyant une requête SNMP à l'adresse IP correspondante : si le noeud est connecté il recevra une réponse de cet agent et dans le cas contraire il pourra en déduire un incident. De même un agent peut envoyer une notification à une station de management si une interface est soudainement down.

3 JAMAP, à la base du PFE

Aujourd'hui les réseaux sont de plus en plus grands, complexes et hétérogènes, c'est pourquoi les effets d'un seul homme ne sont plus suffisants pour les gérer. Il doit se faire assister par des outils d'administration de réseau plus ou moins **automatisés**, dont le développement a été négligé jusqu'à présent. Un des objectifs du projet DataTAG est de développer de nouveaux outils afin de surveiller les réseaux grille de demain et c'est ainsi qu'on m'a chargé de reprendre et d'améliorer un prototype existant pour répondre aux carences des technologies actuelles.

3.1 Historique de JAMAP : Java MAnagement Platform

J-P Martin-Flatin, mon superviseur, a décelé plusieurs carences dans l'utilisation du protocole SNMP et a décrit l'architecture « **WIMA** » dans sa thèse de doctorat et son livre « Web-Based Management of IP Networks and Systems », afin de répondre aux nouvelles attentes des managers de réseaux. Cette architecture est implémentée dans un prototype de recherche **JAMAP**, qui a été développé par J-P Martin-Flatin lui même, Laurent Bovet (EPFL) et Claire Ledrich (Eurecom) pendant leur projet de fin d'études.

J'ai aussi contribué au développement de ce prototype pendant 6 mois tout au long de mon projet de fin d'études au CERN, avec comme objectifs d'actualiser l'application existante et de la moderniser en utilisant les dernières innovations dans le domaine des applications

distribuées. Voici quelles ont été les grandes étapes de ce travail (voir l'annexe 1 pour plus de détails) :

- 15 au 21 mars 2003 : immersion dans le monde du « Network Monitoring » et découverte du protocole SNMP en lisant le livre de Rose et McCloghrie « How To Manage your Network using SNMP »
- 21 au 31 mars 2003 : familiarisation avec les plateformes de « Network Monitoring » et des applications distribuées en lisant le livre de J-P Martin-Flatin « Web-Based Management of IP Networks and Systems ».
- 1^{er} avril-20 avril 2003 : découverte de l'application existante JAMAP et intégration dans un environnement de développement Java (IDE, Borland Jbuilder). Portage du code sur un nouveau serveur d'application et déploiement sur le lien transatlantique Genève - Chicago utilisé dans le cadre du projet DataTAG.
- 20 avril 2003 ... 13 septembre 2003 : phase de développement et d'amélioration de JAMAP, voir « JAMAP Release Notes » en annexe 1. Durant cette phase de développement j'ai continué à me documenter en lisant « Java Thread Programming » de Paul Hyde, « Core Servlets and Java Server Pages » de Marty Hall, « Java Servlet Programming » de Jason Hunter (O'Reilly), « Java and SOAP » de Robert Englander (O'Reilly), « Programming Web Services with SOAP » de James Snell (O'Reilly), et enfin « UML Distilled Second Edition » de Martin Fowler et Kendall Scott.

Afin de comprendre l'étude comparative de solutions ils est nécessaire de présenter les principales caractéristiques de cette plateforme de gestion de réseaux et d'en énumérer les originalités : la partie suivante s'appuie sur le livre de J-P Martin-Flatin.

3.2 Architecture distribuée de JAMAP

JAMAP est une application innovante écrite entièrement en **Java** et permettant d'effectuer du **management automatisé** (monitoring du réseau et collecte des données permanents) ou du **management ad hoc** (monitoring du réseau en temps réel et contrôlé par un utilisateur). Cette plateforme se compose de 3 principaux éléments (voir figure 3) :

- les **managers** qui sont responsables de la collecte et l'analyse des données récupérées des agents
- les **agents** qui sont les éléments du réseau surveillés et dont les données sont analysées par les managers
- Les **management stations** sur lesquelles l'utilisateur se connecte afin d'effectuer du management ad hoc ou configurer JAMAP

Le découpage en trois parties indépendantes est stratégique car il peut permettre de réutiliser certaines parties de JAMAP dans d'autres applications ou d'y intégrer des éléments extérieurs : cet aspect de distribution est particulièrement soigné et j'ai contribué à rendre le prototype encore plus **interopérable**[7] pendant mon stage en utilisant les nouvelles technologies Web. L'installation de l'application dans un environnement inconnue s'avère aussi plus simple après mon PFE car toutes les données de configuration comme le modèle d'information des agents ou les données spécifiques aux divers composants de JAMAP sont désormais contenues dans des fichiers XML (cf. deuxième étude comparative de solutions). Ainsi un simple changement dans un fichier XML (=fichier texte humainement visible) et il est

possible d'interfacer JAMAP avec de nouveaux composants, sans avoir à recompilier une partie du code.

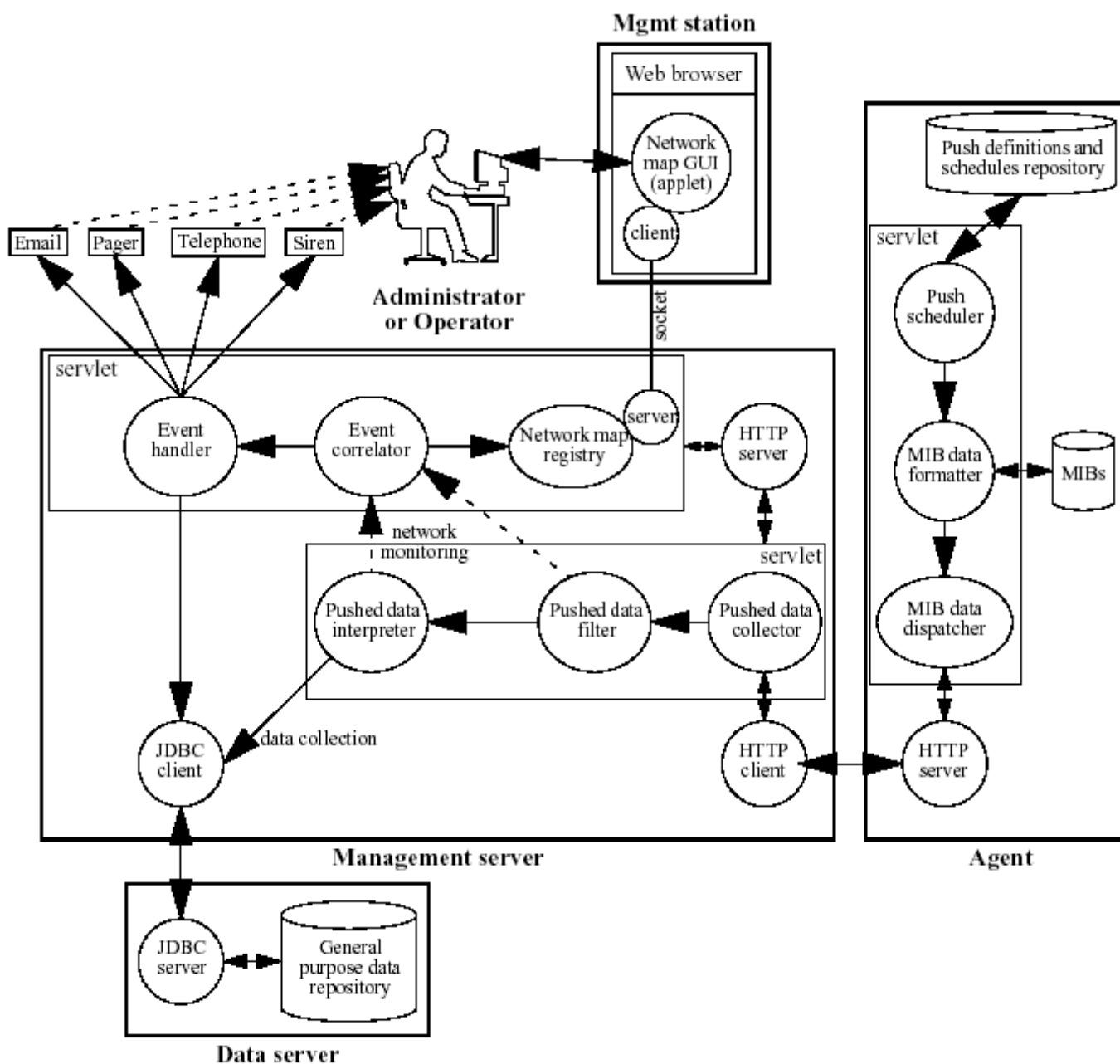


Figure 3 : Architecture distribuée de JAMAP

3.3 JAMAP et les modèles d'information des agents

Le monde de la gestion de réseaux est actuellement partagé entre 2 standards : **SNMP** et **CIM**.

Le Simple Network Management Protocol (SNMP) a été standardisé par l'IETF (Internet Engineering Task Force) et est utilisé par des millions d'équipements depuis déjà plusieurs années.

Le Common Interface Model (CIM) a été standardisé par le DMTF (Desktop Management Task Force) après l'apparition du standard SNMP et s'appuie d'avantage sur les nouvelles technologies, comme XML ou une programmation orientée objet.

Dans le cadre de développement du prototype JAMAP nous ne souhaitons pas prendre partie pour une technologie spécifique et mettons l'accent sur la portabilité et l'intégration dans un environnement quelconque. Ainsi il est possible d'utiliser ces 2 technologies voire même d'en utiliser d'autres en s'appuyant sur le fichier de configuration XML propre à chaque agent (se référer au chapitre 6 pour plus d'informations).

Cette caractéristique de JAMAP est intéressante car les solutions existantes aujourd'hui sont toutes prioritaires et imposent un modèle d'information ainsi que des équipements bien précis.

3.4 JAMAP et l'utilisation du modèle Push

Pour transmettre des données entre 2 éléments du réseau, nous pouvons utiliser 2 méthodes : **pull** et **push**.

Le modèle pull est fondé sur le paradigme requête-réponse. A chaque cycle de collecte des données, le manager (c'est-à-dire le client SNMP) interroge les agents pour connaître la valeur de plusieurs variables de MIB (on a un serveur SNMP par agent). Chaque agent répond alors séparément à chaque requête du manager et ne prend aucune initiative.

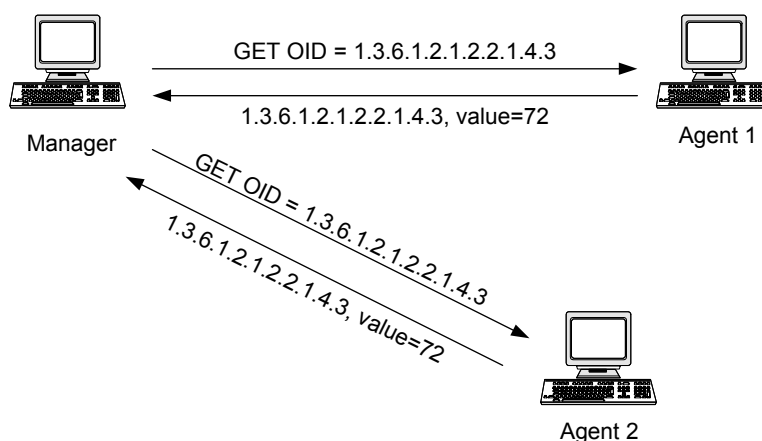


Figure 4 : modèle Pull

Inversement, le modèle push est basé sur le paradigme publication-souscription. Ce modèle implique trois étapes :

- **publication**: chaque agent annonce les MIB qu'il gère et les notifications qu'il est susceptible d'envoyer.
- **souscription**: agent par agent, l'administrateur abonne le gestionnaire (programme sur l'agent) à différentes variables de MIB et notifications ; pour les variables de MIB, la fréquence à laquelle leurs valeurs doivent être envoyées au gestionnaire est également indiquée.
- **distribution**: chaque agent gère indépendamment ses cycles d'information grâce à un ordonnanceur qui va « pousser » les données auxquelles le manager a souscrit.

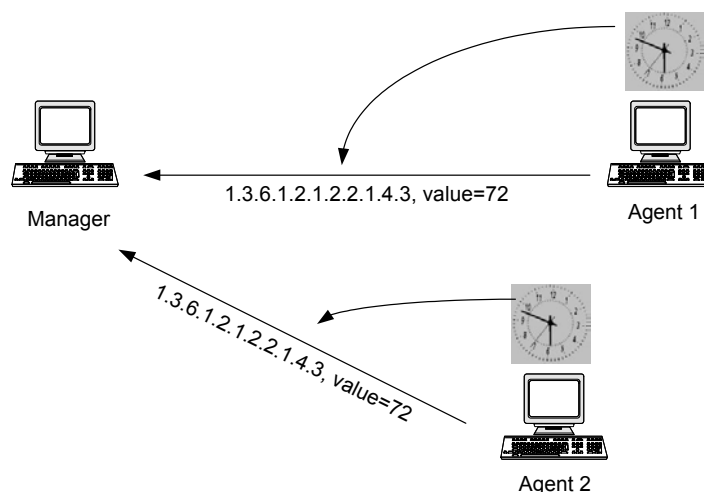


Figure 5 : modèle Push

Ces 2 modèles n'ont pas les mêmes conséquences : avec la méthode « pull » c'est le manager qui prend l'initiative et pour chaque cycle on trouve une requête du manager et la réponse de l'agent sur le réseau. Au contraire avec le modèle « push » l'initiative est prise par l'agent on n'a qu'un seul flux de données sur le réseau qui est la réponse de l'agent : ce modèle génère donc moins de trafic et limite ainsi les perturbations. En outre cette solution présente encore l'avantage de diminuer la charge CPU du manager en déléguant une partie du travail aux agents.

WIMA préconise l'utilisation du push pour la gestion régulière et l'utilisation du pull pour la gestion ad hoc à partir de l'applet de souscription de données par exemple. Cette partie était déjà implémentée à mon arrivée et j'ai seulement modifié l'ordonnanceur coté agent (voir le chapitre 5 étude comparative de solutions).

3.5 JAMAP et l'utilisation du protocole HTTP pour transférer les données entre agents et managers

WIMA prévoit l'établissement d'une connexion persistante entre chaque agent et manager afin d'échanger les données. Celle-ci doit être initialisée depuis le manager pour des raisons de sécurité, celui-ci se trouvant à un endroit protégé. Il est donc nécessaire de trouver une technologie adéquate pour réaliser cette connexion, et nous avons le choix entre plusieurs solutions comme les sockets, Java RMI, ou encore **HTTP**. Les sockets et Java RMI sont à exclure car ces technologies ne permettent pas de traverser les firewalls naturellement et poseraient certains problèmes de time-out en désaccord avec une connexion persistante. Au contraire le protocole HTTP est compatible avec les équipements de sécurité cités précédemment et paraît être la solution du futur pour les applications distribuées comme JAMAP. Cette solution pose cependant un problème de direction des données, car la connexion est initialisée du manager vers l'agent et nous voudrions envoyer des données de l'agent vers le manager, ceci étant contradictoire avec HTTP qui est orienté connexion. Il a donc fallu trouver un moyen d'envoyer une réponse infinie (de la part de l'agent) à une seule requête (celle du manager ouvrant la connexion) et pour cela J-P Martin-Flatin a utilisé la même idée que dans Netscape qui consiste à envoyer une réponse infinie à un simple GET en utilisant le type **MIME**. Plus précisément pour chaque cycle les agents vont envoyer une MIME part contenant des données et leurs définitions.

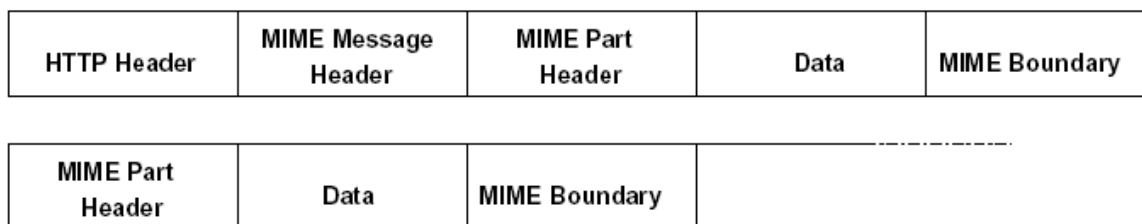


Figure 6 : Réponse infinie en utilisant le type MIME

3.6 JAMAP et XML

L'architecture WIMA se compose de plusieurs éléments qui doivent sans cesse communiquer entre eux. Pour cela nous avons souhaité pouvoir utiliser plusieurs **modèles de communications**, afin de pouvoir opérer avec divers environnements. Le modèle de communication privilégié est cependant celui utilisant **XML**, car cette technologie, en plus d'être simple à lire, facilite l'interopérabilité en étant indépendante des plateformes et logiciels. Cette solution étant gratuite elle paraît donc idéale pour réaliser le prototype de recherche JAMAP.

3.7 JAMAP et les technologies Web

Un des grands avantages de JAMAP est l'utilisation des **technologies Web** afin d'offrir aux utilisateurs une ergonomie satisfaisante tout en étant portable sur tous les environnements : que ce soit sur des stations Unix, Microsoft, ou sur un téléphone GPRS, il est possible de configurer et d'utiliser cette application. Lors de ce stage j'ai eu l'occasion de mettre à jour les technologies jusqu'à présent utilisées dans JAMAP, et ai continué à distribuer l'application en utilisant de nouveaux standards comme les **Web Services**.

La dernière version de JAMAP à mon arrivée au CERN reposait essentiellement sur l'utilisation d'applets et de servlets. Les servlets étaient exécutées sur un serveur Jigsaw 2.2.0 et les applets prévues pour un environnement JDK 1.2. Nous allons faire un bref rappel sur ces technologies Web, afin de pouvoir comprendre en quoi les Web Services apportent un réel changement dans le monde des applications distribuées.

3.7.1 Les Applets Java

Une **applet Java** est un programme écrit en Java situé initialement sur un serveur Apache contenant un site Web et destiné à être exécuté côté **client**, qui doit alors être doté d'une «Java Virtual Machine». Cette technologie présente l'avantage d'être portable et de pouvoir utiliser toute la puissance de Java au sein de pages HTML (un exemple d'applet lancée dans un browser est donné en annexe 3).

Dans le cadre du projet les applets peuvent être exécutées à partir d'un environnement JDK 1.2 et permettent de configurer JAMAP graphiquement. Le prototype stockant les informations de persistance dans des fichiers XML suite à mon travail, il est aussi possible de configurer la plateforme de gestion de réseaux en éditant directement les fichiers de configuration.

3.7.2 Les servlets

Les **servlets** sont des programmes écrits en **Java** qui s'exécutent coté **serveur**. Elles possèdent une adresse Web afin de pouvoir être initialisées par de simples GET ou POST HTTP. Techniquement une servlet est une classe Java héritant de classes spécifiques du package `Javax.servlet` (un exemple de servlet est donné en annexe 4). L'avantage de cette technologie est sa portabilité, les serveurs d'applications les supportant étant gratuits et adaptés à tous les environnements (ils sont généralement écrits en Java et utilisent donc la machine virtuelle de Java comme middleware).

J'avais l'avantage d'avoir utilisé cette technologie lors d'un projet à la Technische Universität de Berlin avant mon arrivée au CERN et ai immédiatement pris l'initiative d'utiliser un nouveau serveur d'application pour supporter cette technologie dans le cadre de JAMAP : le serveur Apache Tomcat 4.1.24. Ce dernier est en train de s'imposer dans le monde open-source et nous verrons dans la prochaine partie que le changement a été judicieux !

3.7.3 Les services Web ou « Web Services »

La notion de services Web ou « Web Services » est apparue récemment suite aux vagues de middleware qui sont arrivées sur le marché ces dernières années. Rappelons qu'un middleware peut être considéré comme un logiciel utilisé par d'autres logiciels. Les serveurs utilisant CORBA ou des solutions propriétaires de Microsoft comme DCOM en sont de très bons exemples. L'utilisation de telles technologies pose des problèmes d'interopérabilité qui sont en partie résolus grâce à l'arrivée des services Web.

En effet la tendance est aujourd'hui à faire coopérer différents services ensemble, qu'ils soient exécutés dans des environnements semblables ou non. Cela n'était pas possible avant l'idée d'utiliser du XML pour interfacier les services entre eux, car tous les flux de données (contenant des objets par exemple) étaient hard codés avec une technologie propriétaire. Ainsi le choix d'un middleware imposait les choix des composants, et les entreprises se voyaient confrontées à des problèmes d'homogénéisation imposant l'écriture redondante d'interfaces.

Cette image est en train de disparaître avec l'arrivée des services Web, qui sont des composants logiciels accessibles via des protocoles standards du Web et interfacés avec du XML, donc un langage indépendant des plateformes et auto descriptible (cf. partie XML). Ce concept de « Web Services » regroupe les technologies **WSDL**, **SOAP** (le plus souvent) et **UDDI** (le plus souvent) que nous allons décrire maintenant.

- **WSDL** (Web Services Description Language)

Un service Web est décrit dans un document WSDL accessible en ligne, précisant pour un service donné les protocoles, serveurs, ports utilisés, opérations pouvant être effectuées, formats des messages d'entrée et sortie, ainsi que les exceptions pouvant être renvoyées. Ce fichier est nécessaire pour générer la requête XML adéquate coté client afin d'utiliser un service donné. Cette requête sera ensuite envoyée au point d'accès du service en utilisant le protocole adéquat (SOAP tend à s'imposer).

Un document WSDL est écrit en XML et contient donc les éléments suivants[8] :

- *<definitions>* : Cet élément contient la définition du service et constitue la racine de tout document WSDL. Ce tag peut contenir les attributs précisant le nom du service, et les espaces de nommage.
- *<message>* et *<portType>* : Ces éléments définissent les opérations offertes par le service, leurs paramètres d'entrée et de sortie, etc. En particulier, un *<message>* correspond à un paramètre d'entrée ou de sortie d'une *<operation>*. Un *<portType>* définit un ensemble d'opérations. Une *<operation>* définit un couple message entrée / message sortie. Par exemple, dans le monde Java, une opération est une méthode et un *portType* une interface.
- *<binding>* : Cet élément associe les *<portType>* à un protocole particulier. Les bindings possibles sont SOAP, CORBA ou DCOM. Actuellement seul SOAP est utilisé. Il est possible de définir un binding pour chaque protocole supporté.
- *<service>* : Cet élément précise les informations complémentaires nécessaires pour invoquer le service, et en particulier l'URI du destinataire. Un *<service>* est modélisé comme une collection de ports, un *<port>* étant l'association d'un *<binding>* à un URI. Il est aussi possible de définir des types de données complexes dans un tag *<types>* juste avant le tag *<message>*. Enfin, chaque élément WSDL peut être documenté à l'aide de l'élément *<documentation>*. Cet élément contient des informations liées à la compréhension du document par les utilisateurs humains du service.

- Le protocole **SOAP**

Actuellement c'est SOAP qui est utilisé comme protocole de communication pour transporter les requêtes XML d'invocation de services.

Un message SOAP a la structure suivante[8] :

- *Le HTTP Header* : Le protocole HTTP envoie une requête POST. L'entête HTTP se trouve juste avant le message SOAP, et définit le destinataire du message, les règles d'encodage HTTP, etc.
- *L'enveloppe SOAP* : L'enveloppe contient l'espace de nommage définissant la version de SOAP utilisée, ainsi que les règles de sérialisation et d'encodage.
- *Le header SOAP* : Cette partie du message optionnelle sert à transmettre des informations nécessaires pour l'exécution de la requête SOAP aux intermédiaires qui recevront le message. On y précise généralement des informations liées aux transactions, à l'authentification, etc.
- *Le Body SOAP* : Le body SOAP contient toutes les informations que l'on veut transmettre à l'application distante.

- La découverte de services et l'utilisation de **UDDI**

La technologie UDDI permet de découvrir des services dynamiquement (autodiscovery). Nous décrivons ce standard dans le chapitre 6 : étude comparative de solutions.

Un des objectifs du PFE était d'intégrer cette nouvelle technologie dans JAMAP afin de renforcer sa philosophie d'interopérabilité. Cela imposait l'ajout d'un middleware supportant les Web Services coté serveur.

J'ai décidé d'utiliser le serveur de Web Services Apache **AXIS**, qui s'interface particulièrement bien avec Apache Tomcat. Ce serveur est actuellement la référence dans le

monde open-source en tant que successeur du projet Apache SOAP. J'avais à nouveau la chance d'être familier avec cette technologie suite à un projet de commerce électronique effectué à l'institut de recherche FOKUS de Berlin.

Pour résumer voici un bilan des middleware et technologies utilisées dans JAMAP suite à mon travail :

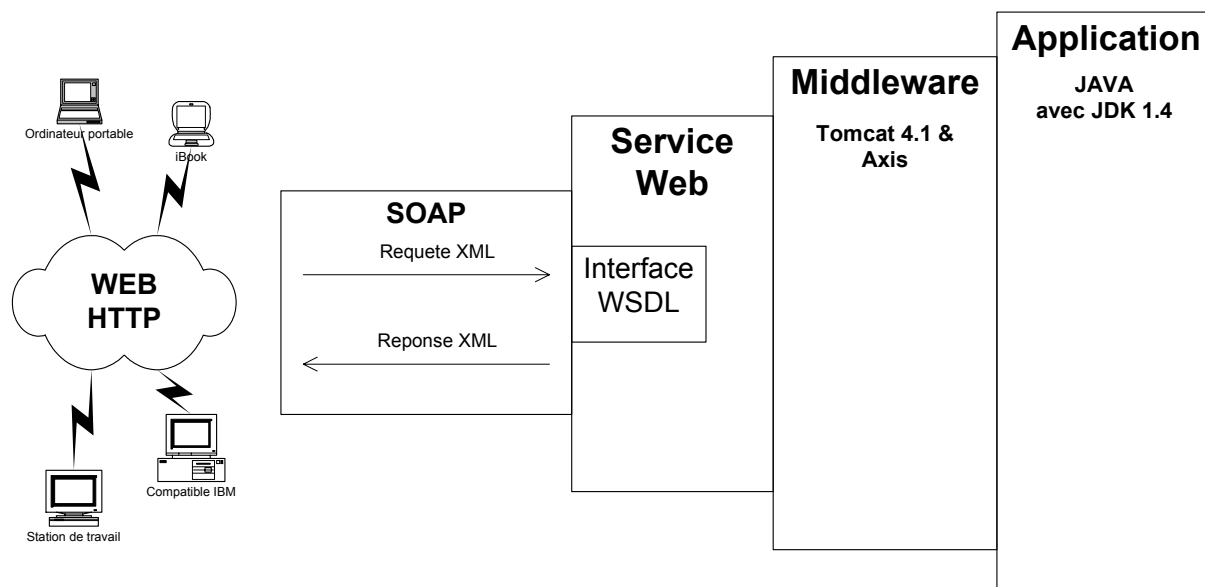


Figure 7 : Architecture utilisée dans JAMAP version 1.3

A la fin du projet nous avons bien réussi à utiliser un service Web pour la partie « configuration » de notre plateforme. Nous avons aussi initialement convenu avec mon superviseur d'utiliser cette technologie pour la découverte des agents du réseau mais nous expliquerons plus tard pourquoi un autre choix s'est imposé (voir la deuxième étude comparative de solutions).

Les deux parties suivantes illustrent en quoi ce projet de fin d'études a été avant tout un **travail d'ingénieur**, avec ses difficultés et ses responsabilités.

En effet le fait de devoir innover dans un cadre de recherche implique très souvent une réflexion sur plusieurs scénarios, afin de trouver une solution optimale aux problèmes posés.

4 Etude comparative de solutions : nouveau design de l'ordonnanceur coté agent

JAMAP release 0.4 souffrait de nombreux problèmes de synchronisation : mauvaise gestion du temps, boucles infinies... Un des éléments les plus gênants était son manque d'exactitude dans la génération des données coté agent, du à une implémentation trop simple de l'**ordonnanceur** ou « **scheduler** ». Ce dernier présentait le défaut de ne pas tenir compte de l'horloge système et poussait ses données sur la connexion persistante de manière aléatoire et sans aucune optimisation.

Un des premiers objectifs du PFE a été de reprendre le design de ce composant, afin d'envoyer les données souscrites le plus précisément possible aux collecteurs de données. Mon premier réflexe a été d'utiliser le « **multithreading** » et d'implémenter un nouveau scheduler le plus rapidement possible : après de longues heures de travail j'ai cependant

réalisé que cette tâche était bien plus difficile que je me l'étais imaginé et ai pris conscience de la nécessité de réfléchir avant tout sur le **design** des programmes avant leurs détails d'implémentation : Jean-Philippe Martin-Flatin m'avait d'ailleurs sensibilisé sur ce point dès le début du stage mais il m'a fallu expérimenter pour bien le comprendre !. J'en suis donc arrivé à penser aux améliorations possibles de cet ordonnanceur coté agent.

4.1 La gestion du temps dans une application orientée objets

Cette partie présente différentes manières d'effectuer des tâches à intervalles réguliers dans un environnement de programmation orienté objets.

4.1.1 Solution basique : utilisation de boucles

Une première solution pour effectuer une tâche à intervalles réguliers est de faire tourner son programme dans le vide entre 2 traitements : cette solution était initialement présente dans JAMAP, le programme incrémentant simplement une variable pendant les phases d'attente. Ce procédé souffre cependant de problèmes de précision car la vitesse d'exécution des programmes est dépendante de la fréquence du microprocesseur de la machine et de son environnement. Ainsi cette solution ne permet pas de gérer le temps correctement entre deux phases d'analyses, ce qui n'est pas pensable pour une application de surveillance de réseaux qui se doit d'être fiable.

4.1.2 Solution rapide : intégrer sa logique dans une thread et la faire dormir

A première vue il est possible d'utiliser les **threads** pour mettre en attente un programme. Cette idée permet de solutionner les problèmes relatifs aux fréquences de CPU, la machine virtuelle de Java gérant le temps système et proposant des fonctions d'exclusion mutuelle ou de blocage sur sémaphores très précises. Il serait donc possible de rendre notre scheduler multithread en étendant le package Java Thread, et de le faire dormir le temps nécessaire grâce à la fonction **sleep** (ce temps est la durée entre deux poussages de données de l'agent vers le manager sur la connexion persistante). Le programme aurait donc l'allure suivante :

```
//calcul de la durée a attendre en tenant compte de toutes les souscriptions
this.sleep(durée a attendre);
//envois de données au manager
```

Cette solution présente cependant un problème majeure, car il n'est plus possible de réveiller un thread en plein sommeil avec les nouvelles API Java, et nous avons besoin de cette fonctionnalité. En effet lorsqu'un utilisateur va souscrire à de nouvelles données ou annuler certaines souscriptions le scheduler de l'agent concerné devra réagir en temps réel et prendre en compte les nouveaux choix immédiatement : le thread en plein sommeil devra donc être réveillée et il faudra recalculer le nouveau temps de sommeil en fonction des nouveaux paramètres. En utilisant un simple thread et la fonction sleep, cela n'est pas encore possible. Il faut donc à nouveau envisager une autre solution.

4.1.3 Utilisation de sémaphores ?

La solution précédente résolvait seulement une partie du problème mais était un bon point de départ vers la solution finale. L'idée d'utiliser des threads pour gérer le temps dans ce cas de figure est définitivement la bonne, mais il fallait seulement trouver le moyen de

pouvoir atteindre les programmes en cas d'incidents, même s'ils étaient dans une phase d'attente. Pour réaliser cette fonctionnalité il est possible d'utiliser les fonctions de **synchronisation sur sémaphores** fournies en standard par Java.

Plus précisément les objets Java peuvent se synchroniser entre eux en utilisant les méthodes **wait()**, **notify()** et **synchronized()**[9] :

Synchronized() : cette méthode permet d'acquérir le **moniteur** d'un objet, ce qui est indispensable pour pouvoir appeler les fonctions wait() ou notify(). On peut aussi déclarer une méthode entière comme synchronized, ce qui garantit que le thread exécutant cette méthode a acquis le moniteur.

Wait() : le thread qui appelle la méthode wait() libère le moniteur dont il avait possession et se bloque en attendant d'être notifié. Lorsqu'il est signalé, le thread reprend son exécution à l'instruction suivant le wait(), après avoir réacquit le moniteur.

Notify() : la méthode notify() permet de signaler son réveil à un thread qui avait appelé wait() sur le même objet. Il se peut que le thread notifié se bloque sur le moniteur si le thread qui a effectué le notify() le possède encore. Le thread notifié reprendra réellement son exécution lorsque celui qui l'a réveillé libérera le moniteur et qu'il en obtiendra la possession.

Nous allons donc réaliser notre ordonnanceur en étendant la classe Thread. Une fois le temps d'attente calculé, la méthode se synchronisera sur tout l'objet (elle possèdera donc le moniteur de l'objet de type PushScheduler) et attendra au moyen de la commande wait(durée). Le programme aura donc l'allure suivante :

```
//calcul de la durée a attendre en tenant compte de toutes les souscriptions
Synchronised (this)
{
//phase d'attente
wait(duree a attendre);
}
//envois de données au manager, cas standard ou wait interrompu volontairement
```

Nous devons maintenant prévoir au sein de ce PushScheduler une méthode permettant de le signaler lorsqu'il est en phase d'attente, afin de considérer les nouvelles souscriptions de l'utilisateur. Nous prévoyons donc la fonction stopthread qui sera aussi synchronisée sur l'objet entier et utilise la méthode notify() :

```
public void stopThread(){

synchronized(this){
notify();
}
}
```

L'annexe 5 présente l'implémentation de ces méthodes dans JAMAP.

Ainsi l'utilisation des wait/notify nous aura permis de résoudre les problèmes de timings tout en conservant un programme réagissant aux instructions de l'utilisateur en temps réel.

4.2 Optimisation de l'ordonnanceur coté agent

Rappelons la fonction principale de l'ordonnanceur coté agent : pousser les données souscrites par les différents managers ou collecteurs de données de façon régulière. Nous avons déjà résolu les problèmes de timings dans la section précédente et allons maintenant réfléchir sur l'implémentation de cet ordonnanceur. La récupération des informations coté agent (requête SNMP par exemple) peut se faire en utilisant des API gratuites : ainsi à première vue il ne reste plus qu'à pousser le résultat des requêtes sur la connexion persistante du manager approprié. Mais un travail de design s'impose à nouveau avant la phase de codage. En effet il peut y avoir plusieurs façons de pousser les données de l'agent, lorsqu'il est connecté à plusieurs collecteurs de données en même temps : nous allons présenter des solutions possibles afin de trouver au final un bon compromis.

4.2.1 Solution basique : un ordonnanceur pour un collecteur de données

Une solution envisageable est de prévoir un ordonnanceur pour chaque collecteur de données : il faudra alors stocker dans une table de correspondance (table de hachage par exemple) les associations collecteur de données (coté manager) → ordonnanceur (coté agent). Voici un exemple concret de cette implémentation, avec un agent et trois collecteurs de données :

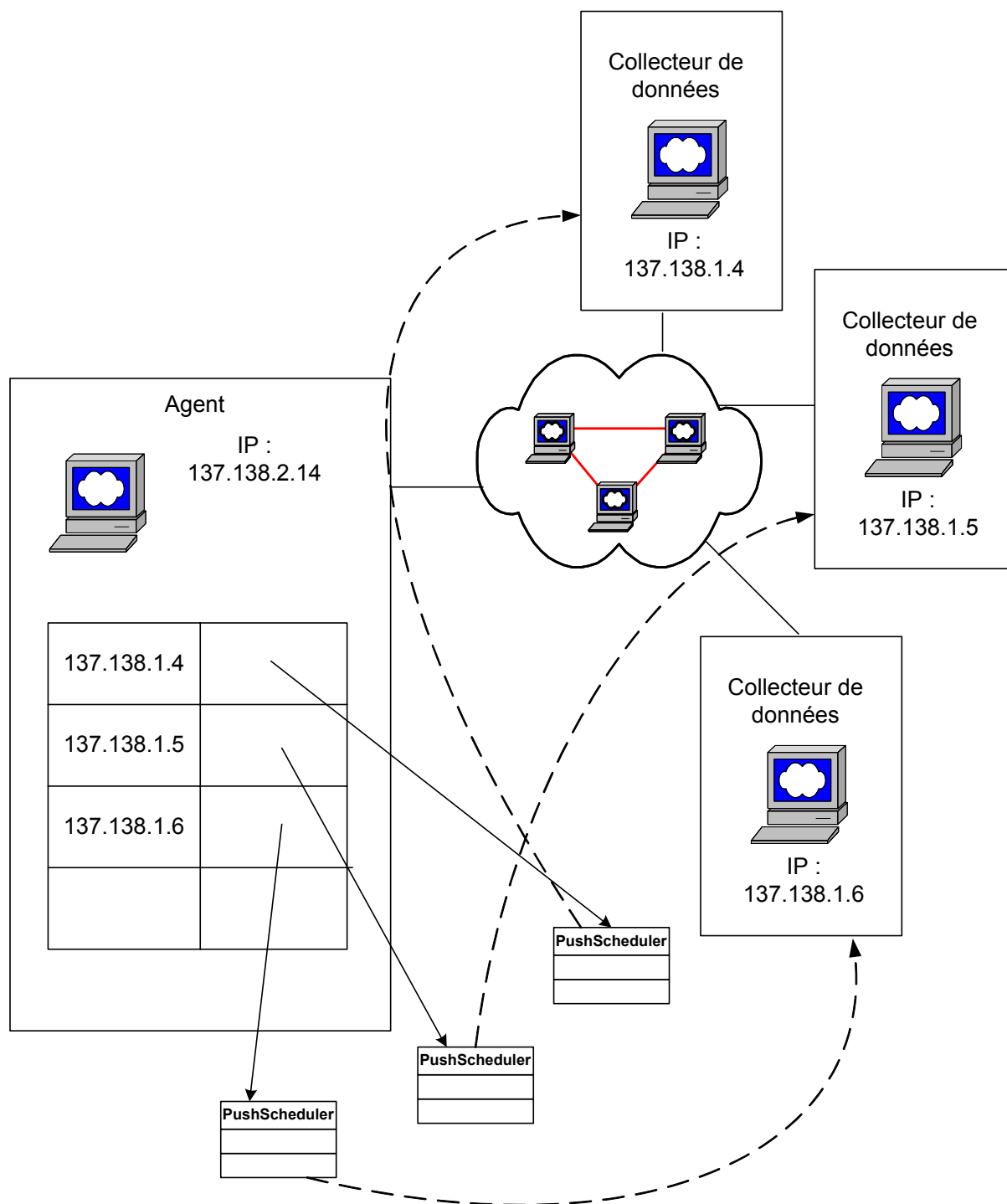


Figure 8 : Modèle prévoyant un ordonnanceur pour chaque collecteur de données

Cette solution est la plus simple. Elle souffre cependant de nombreux défauts, intolérables pour une application de network monitoring.

En effet imaginons que deux collecteurs de données veuillent recevoir la même information : avec cette implémentation chaque scheduler va effectuer sa requête sur l'agent sans même savoir qu'un autre scheduler a déjà récupéré la même donnée quelques dixièmes de secondes avant. Il génèrera donc du trafic inutile avec sa requête et risquera même de déstabiliser l'agent surveillé (il est fortement déconseillé de bombarder un routeur de requêtes SNMP). Ce modèle n'est donc pas adapté à notre prototype et nous devons trouver un moyen **d'optimiser**

les requêtes sur les agents afin de minimiser notre occupation du réseau : cela revient à gérer les souscriptions des collecteurs de données dans un même ordonnanceur.

4.2.2 Un unique ordonnanceur pour tous les collecteurs de données

Dans cette partie nous proposons de créer un seul ordonnanceur pour chaque agent, qui s'occupera de tous les collecteurs de données à la fois. La figure illustre cette proposition.

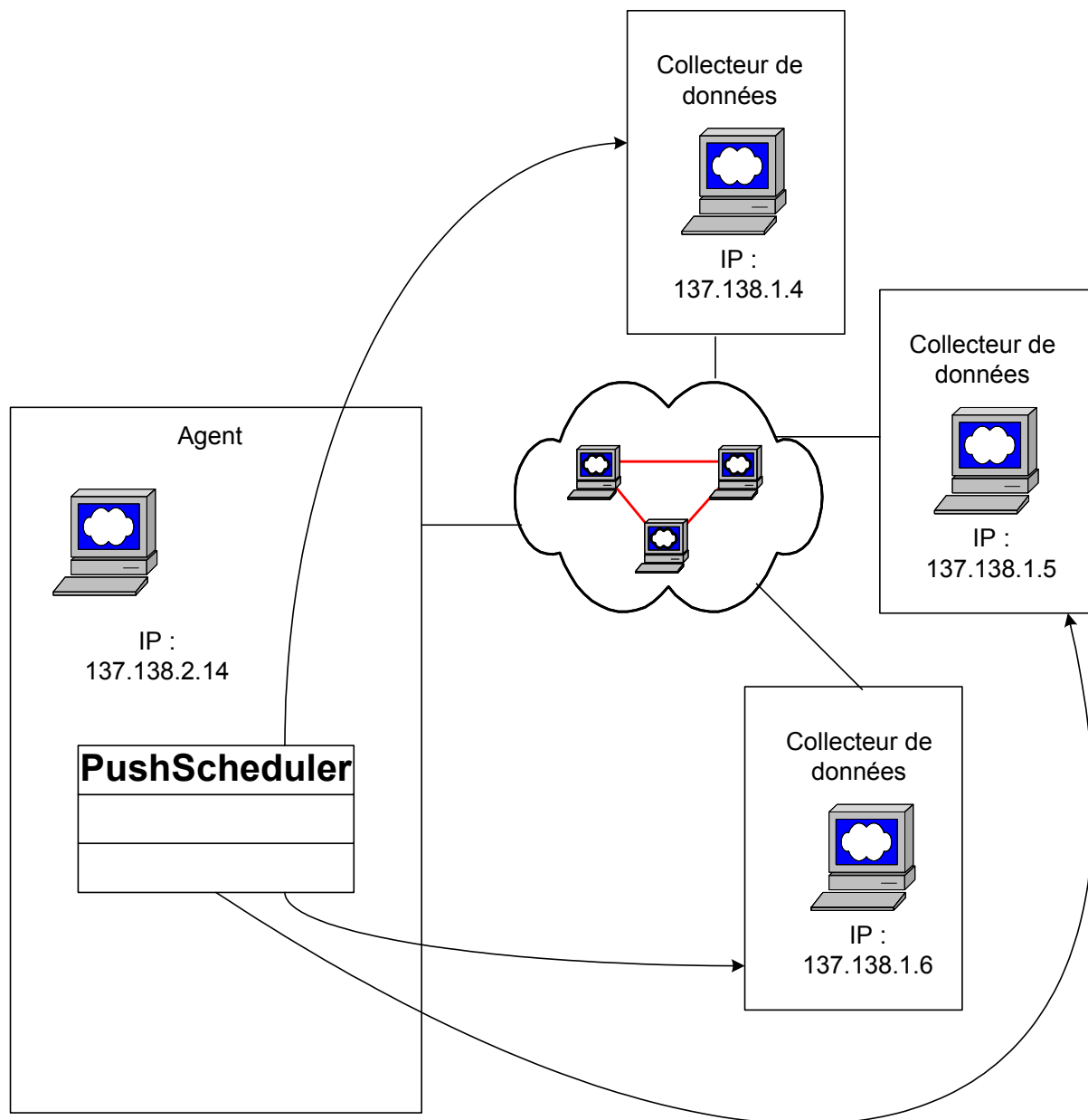


Figure 9 : Un ordonnanceur responsable de l'intégralité des collecteurs de données

Nous allons maintenant donner les détails d'implémentation de cette solution. A chaque cycle le scheduler interrogera le SubscriptionController (fonction GetNextPush), chargé de gérer les souscriptions de tous les collecteurs de données en même temps. Ce dernier a pour fonction de renvoyer au scheduler deux informations : les données qu'il devra pousser au prochain cycle et l'heure de cet événement.

Le scheduler peut ensuite se bloquer sur sémaphores en instanciant la classe `WaitThread` présentée dans la section 4.1.1.3, et plus tard effectuer ses requêtes sur l'agent (fonction `getObject`). Si deux collecteurs désirent la même information on ne fait qu'une seule requête sur l'agent : la réponse est ensuite envoyée sur chaque collecteur dans la boucle « For each subscription and controller » .

Voici un **diagramme de séquences** en UML illustrant cette partie du code :

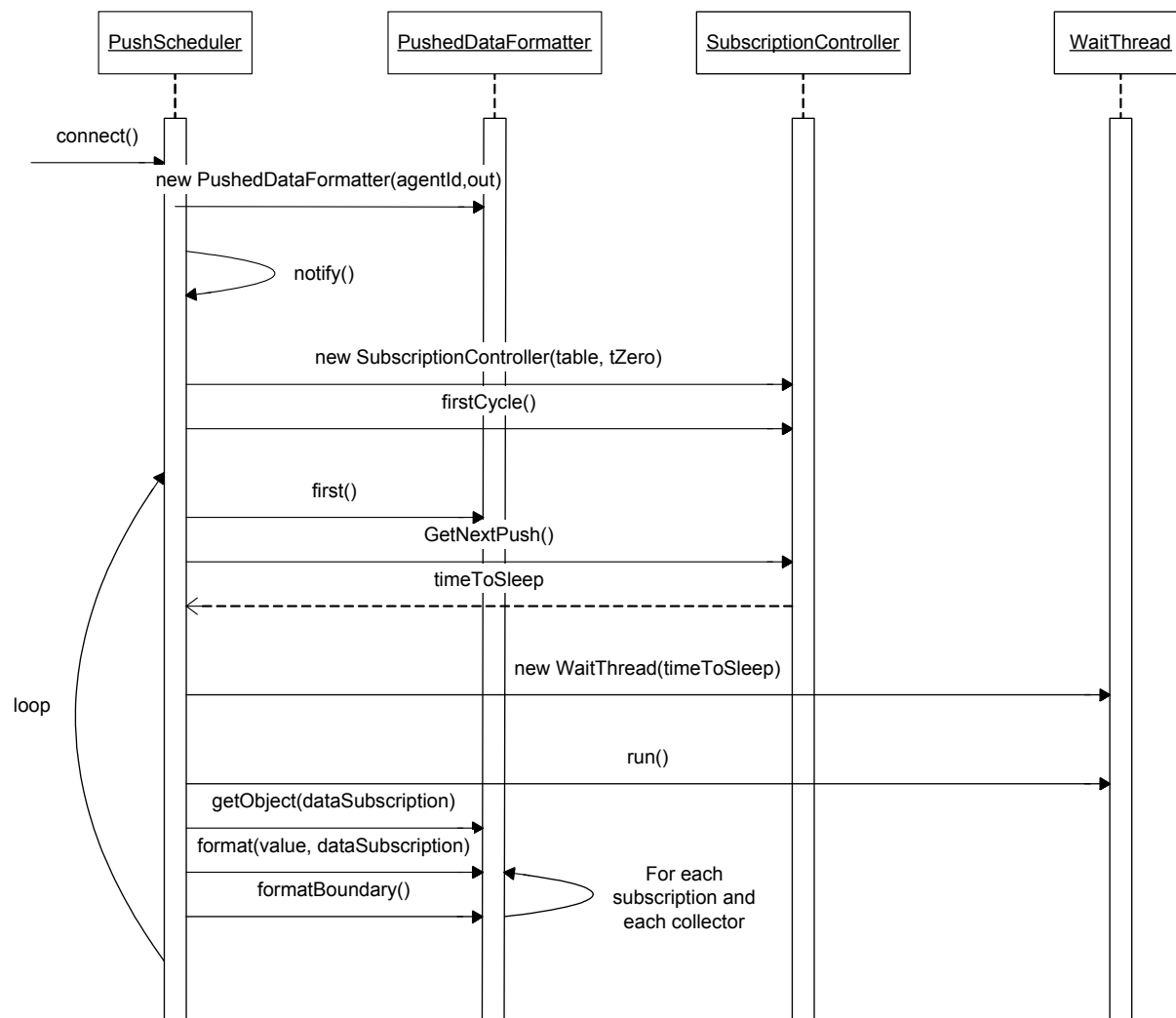


Figure 10 : Implémentation du PushScheduler sur un agent

Nous avons bien réussi à implémenter un `PushScheduler` respectant les contraintes horaires et minimisant les requêtes coté agent. Décrivons maintenant son comportement lors de l'ajout de nouvelles souscriptions.

Afin de minimiser les perturbations sur l'agent nous avons introduit la variable `waitForPushTolerance`, que l'utilisateur a la charge de configurer dans un fichier. Celle-ci définit la tolérance du `PushScheduler` à regrouper la nouvelle souscription avec d'autres déjà existantes. Nous allons illustrer cette fonction avec un exemple concret :

Un collecteur 1 s'est abonné à la variable IfInOctets toutes les 5 minutes. Un nouveau collecteur 2 souhaite aussi s'abonner à la même variable toutes les 10 minutes et va le signaler au PushScheduler à un instant aléatoire. Avec notre proposition si la tolérance définie par l'utilisateur est supérieure à 5 minutes (on considère les cas extrêmes) nous synchroniserons les interrogations sur l'agent des collecteurs 1 et 2 afin de ne faire qu'une seule requête. Dans le cas contraire le PushScheduler interrompra sa phase de sommeil et traitera la nouvelle souscription immédiatement. Une telle logique permet à nouveau d'alléger les traitements sur l'agent.

Voici maintenant le diagramme séquence UML de cette méthode addSubscription ou « ajout de souscription » :

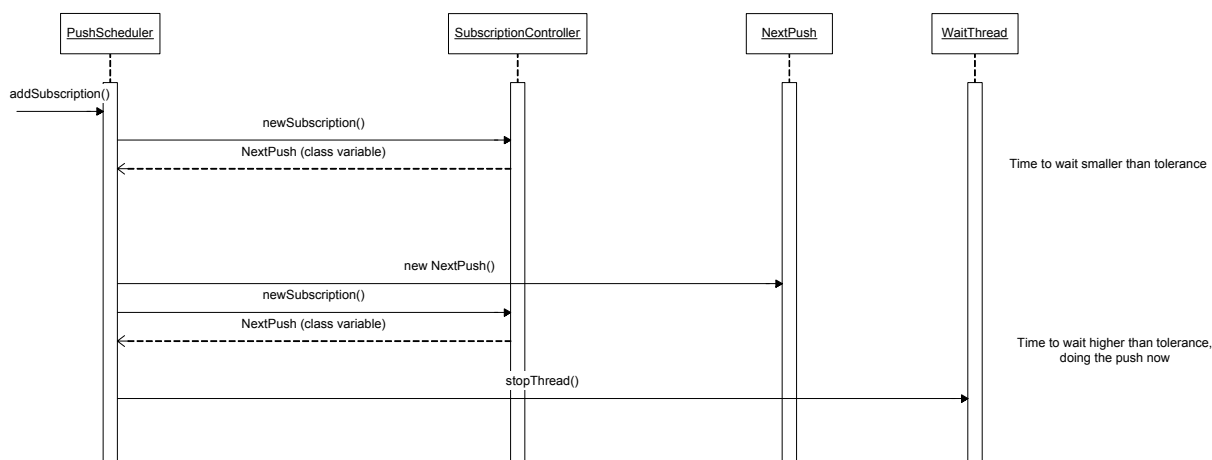


Figure 11 : Ajout de souscription au PushScheduler

4.3 Ajout du concept de proxy

Le PushScheduler tel qu'il a été décrit précédemment a été implémenté et intégré dans la release 0.7 de JAMAP. Quelques mois plus tard il a du subir quelques modifications lorsque le concept de proxy a été ajouté à l'application distribuée. L'arrivée de cette nouvelle fonctionnalité a permis de démontrer la validité des solutions choisies dans les parties précédentes.

Cette notion de proxy a été ajoutée à JAMAP afin de pouvoir gérer des éléments du réseau ne pouvant supporter l'ajout d'un serveur d'application particulier (c'est le cas des routeurs par exemple). Ainsi les agents du réseau supportant JAMAP peuvent désormais interroger d'autres éléments afin d'en remettre les données aux collecteurs : ils jouent un rôle d'intermédiaire. Concrètement ces agents doivent gérer les souscriptions d'autres agents en plus des leurs, ce qui nécessite de légères modifications dans leur design. Or le PushScheduler avait déjà été optimisé afin d'être intégré à un agent particulier : il était donc naturel de ne pas le modifier et de créer une instance de PushScheduler par agent géré. Outre quelques changements mineurs cette solution a seulement nécessité l'ajout d'une table de correspondance (table de hachage par exemple) au niveau de la PushDispatcherServlet, référençant les associations agent géré → PushScheduler pour cet agent.

La figure suivante illustre ces changements :

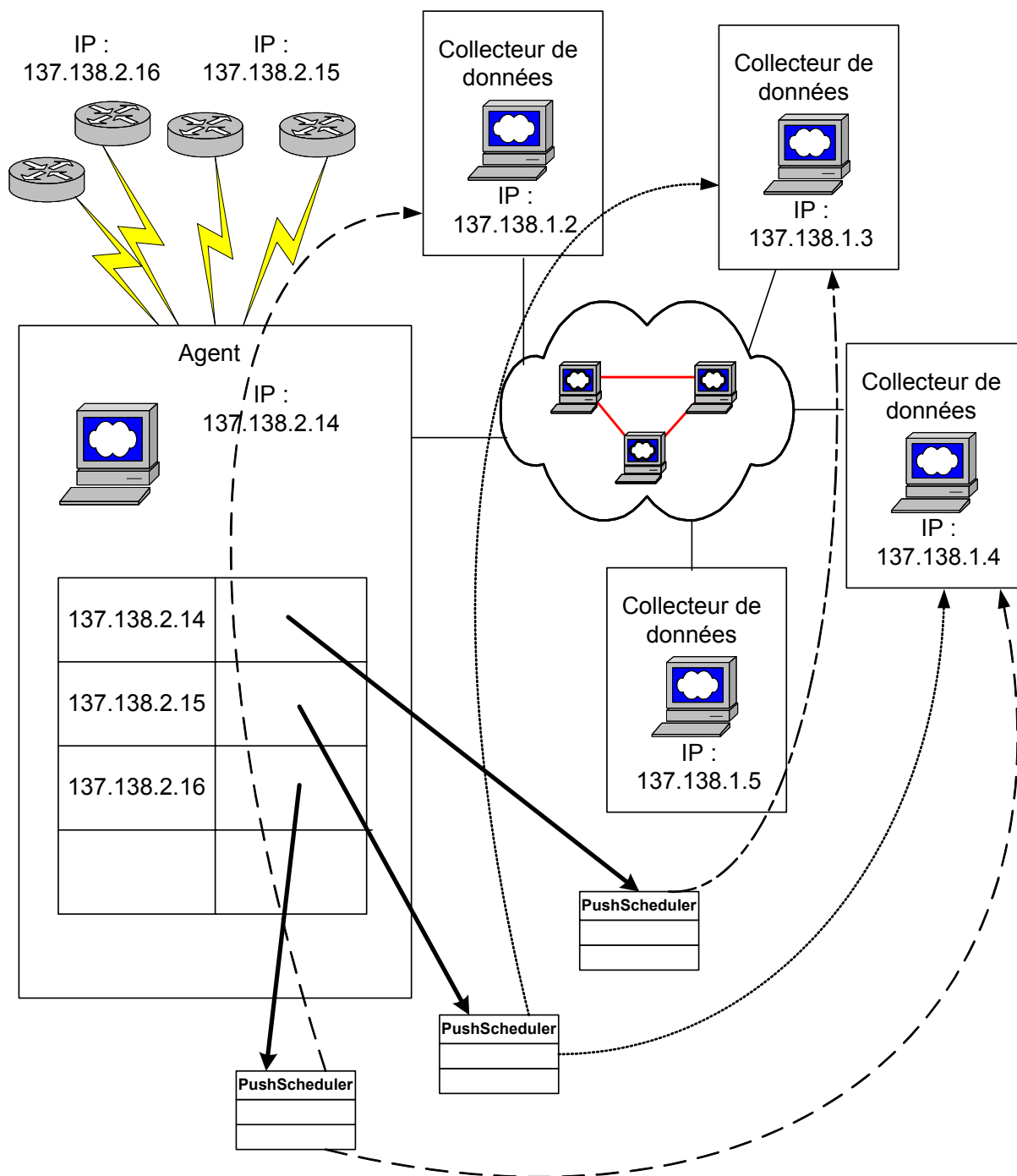


Figure 12 : Implémentation finale des PushSchedulers au niveau de l'agent

Ainsi un design soigné permet d'ajouter par la suite de nouvelles fonctionnalités très rapidement !

5 Etude comparative de solutions : découverte dynamique des agents et de leurs caractéristiques

L'objectif de ce stage était avant tout d'ajouter de nouvelles fonctionnalités à JAMAP en utilisant les dernières technologies liées aux services Web. Ainsi il paraissait naturel de considérer les registres **UDDI** (Universal Description, Discovery and Integration) afin d'améliorer notre application distribuée.

5.1 Présentation de UDDI

UDDI a émergé suite à une collaboration entre Ariba, IBM et Microsoft. Ce terme désigne un protocole **d'annuaire**, basé sur **SOAP**, et permettant de découvrir des services Web dynamiquement.

5.1.1 Spécification technique

Le modèle d'information utilisé par un registre UDDI est défini dans un schéma XML et comprend quatre éléments[10] :

- **businessEntity** : cet élément décrit une entreprise ou une organisation en fournissant des informations similaires à celles contenues dans les pages jaunes d'un annuaire téléphonique : nom, localisation, description, personne à contacter...
- **businessService** : cet élément propose une description très générale d'un service fourni par une entreprise ou une organisation.
ex : service d'expédition, service de paiements...
- **bindingTemplate** : cet élément contient une description technique d'un service donné. Il inclut un point d'accès (Access Point) d'un Web Service ou un lien vers un système permettant de le découvrir.
- **tModel** : cet élément contient des liens vers les documents techniques utilisés par les développeurs de services Web (ex : fichier WSDL) ainsi que des méta informations sur ces documents : il est à la source de l'interopérabilité des services entre eux.

Les trois premiers éléments sont organisés de façon **hiérarchique** : une **businessEntity** contient un ou plusieurs **businessService**, qui contient lui même un ou plusieurs **bindingTemplate**. Les **tModels** par contre n'appartiennent pas à cette hiérarchie et sont simplement pointés par un ou plusieurs **bindingTemplate** : ainsi cet élément peut être commun à plusieurs **businessEntity**. La figure 13 illustre cette organisation.

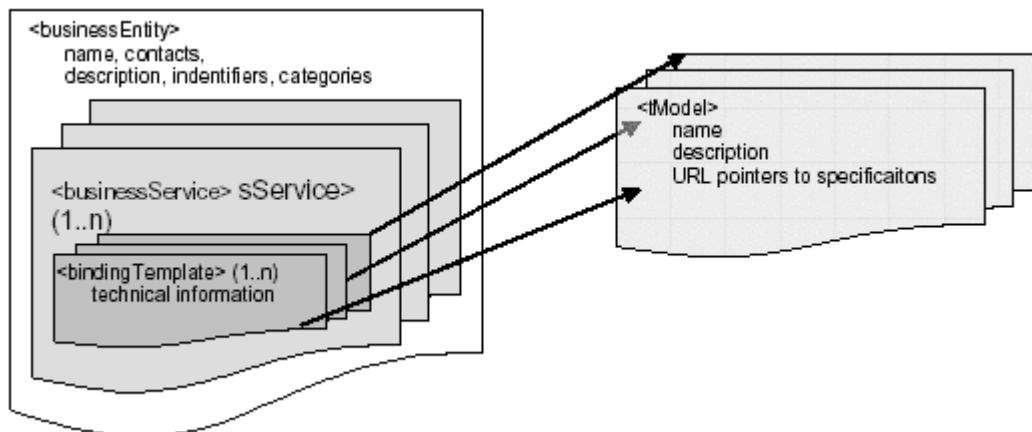


Figure 13 : Organisation hiérarchique des composants dans un registre UDDI

Les annuaires UDDI sont synchronisés entre eux et hébergés par des opérateurs UDDI dont la liste figure sur le site Web <http://www.uddi.org>.

5.1.2 Finalité

Les registres UDDI sont utilisés dans le monde industriel pour publier des informations spécifiques aux entreprises (ex : localité), des renseignements sur les services qu'elles offrent (ex : transactions bancaires), et des informations techniques sur le mode d'accès à ces services (ex : fichiers WSDL). Ils s'accompagnent aussi d'API facilitant la publication et la découverte des services, au sein d'une application distribuée.

Voici un exemple concret de l'utilisation de tels registres : une entreprise de pizza vient d'ouvrir un nouveau point de distribution aux Etats-Unis et désire désormais proposer un service de commande en ligne. Elle réalise donc en back-office le design de son application d'e-commerce et nécessite alors un partenaire pour le paiement des clients en ligne. En consultant un registre UDDI et en cherchant le service approprié dans la même région il lui suffira de télécharger les informations techniques du service bancaire découvert dynamiquement et de l'intégrer dans son application distribuée comme un simple composant.

5.2 Utilisation de registres UDDI dans JAMAP

Nous aimerions disposer d'un certain nombre de services Web dans JAMAP et pourrions donc les référencer dans des registres UDDI.

5.2.1 Informations à découvrir dynamiquement

Une évolution intéressante de la plateforme de management JAMAP serait la découverte dynamique des agents du réseau et de leurs caractéristiques. En effet ces informations étaient initialement stockées dans un fichier Java, ce qui ne s'avérait pas très pratique : lorsqu'un administrateur réseau souhaitait ajouter un nouveau routeur dans les éléments surveillés par JAMAP, il devait éditer ce fichier et y insérer la nouvelle adresse, le compiler et enfin le copier sur toutes les machines de management. Outre la quantité de travail cette solution n'était pas très conviviale pour un novice. Ainsi nous avons eu l'idée avec mon superviseur de stocker ces informations dans un nouveau composant, afin de pouvoir les éditer facilement : le registre UDDI était alors une solution intéressante.

5.2.2 Incompatibilité des registres UDDI avec JAMAP

Dans le cadre de la découverte dynamique des agents nous avons besoin d'un système de **publication/souscription**, qu'un registre UDDI est susceptible d'offrir à première vue. Les informations que nous souhaitons publier sont l'adresse de l'agent avec ses caractéristiques comme son domaine d'appartenance (management domain), les proxys qu'il peut gérer, son modèle d'information (SNMP avec les MIBS correspondantes ou CIM), les adresses de ses servlets, les URI de ses pages HTML de configuration... Or il se trouve que les registres UDDI ne sont pas prévus pour une telle granularité et que le concept de services dans ces registres intervient à un niveau beaucoup plus général : le but est de publier les activités principales d'une entreprise pour qu'elles puissent être découvertes par la suite. Or avec JAMAP nous aimerions pouvoir structurer d'avantage d'informations dans le registre et y définir notre propre schéma XML : cette fonctionnalité n'est malheureusement pas disponible avec la spécification de UDDI actuelle, c'est pourquoi la seule solution envisageable était de stocker les informations sous forme d'XML dans le champ instanceParms fourni par UDDI. Après un essai d'implémentation nous avons réalisé que l'idée d'utiliser de tels registres n'était pas profitable à JAMAP; à vrai dire nous avons déploré l'existence d'un système de publication/souscription plus standard, permettant aux développeurs d'applications de gestion de réseaux d'y définir leurs propres schémas d'informations.

La section suivante présente la solution répondant le mieux à nos attentes.

5.3 XML pour configurer JAMAP

Après l'échec précédent nous avons décidé de stocker les informations de configuration de JAMAP dans des fichiers XML respectant des schémas adaptés à la gestion de réseaux.

5.3.1 La carte du réseau dans un fichier networkMap.xml

Nous avons créé un premier schéma XML networkMap.xsd représentant la carte du réseau géré par notre application de monitoring. Après réflexion ce fichier devait contenir les domaines de management, les agents gérés, leurs proxys éventuels, leurs collecteurs de données ainsi que l'emplacement de leur fichier de configuration. Nous en sommes donc arrivés au schéma suivant :

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!-- networkMap contains the list of agents managed by Jamap for each management
  domain -->

  <xs:simpleType name="IPv4Address">
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="collapse" fixed="true"/>
      <xs:pattern value="((1?[0-9]?[0-9]|2[0-4][0-9]|25[0-5]).){3}(1?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="IPv6Address">
    <xs:restriction base="xs:string">
```

```

    <xs:whiteSpace value="collapse" fixed="true"/>
    <xs:pattern value="((1?[0-9]?[0-9]|2[0-4][0-9]|25[0-5]).){7}(1?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])"/>
  </xs:restriction>
</xs:simpleType>

```

```

<xs:element name="networkMap">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="managementDomain" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

```

<xs:element name="managementDomain">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string" minOccurs="1" maxOccurs="1" />
      <xs:element ref="agent" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

```

<xs:element name="agent">
  <xs:complexType>
    <xs:sequence>
      <xs:choice>
        <xs:element name="ipv4Address" type="IPv4Address"/>
        <xs:element name="ipv6Address" type="IPv6Address"/>
      </xs:choice>
      <xs:choice>
        <xs:element name="ipv4CollectorAddress" type="IPv4Address"/>
        <xs:element name="ipv6CollectorAddress" type="IPv6Address"/>
      </xs:choice>
      <xs:element ref="proxy" minOccurs="0" maxOccurs="1" />
      <xs:element name="agentManagement" type="xs:anyURI" minOccurs="1"
maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

```

<xs:element name="proxy">
  <xs:complexType>
    <xs:sequence>
      <xs:choice>
        <xs:element name="ipv4Address" type="IPv4Address"/>
        <xs:element name="ipv6Address" type="IPv6Address"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

```

    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>

```

La création d'un schéma XML networkMap.xsd adapté à la gestion de réseaux a permis d'établir des restrictions sur les données de configuration et implique la validité des adresses IP ou des URI dans les fichiers networkMap.xml. Un exemple de tel fichier est donné en annexe 6.

5.3.2 Les informations relatives à un agent donné dans un fichier agentManagement.xml

Nous avons introduit un deuxième schéma XML agentManagement.xsd, responsable des informations spécifiques à un agent donné. Après réflexion ce fichier devait contenir le modèle d'information de l'agent, l'adresse de ses servlets et de ses applets, le point d'accès de ses services Web et l'adresse de son fichier WSDL. Nous en sommes donc arrivés au schéma suivant :

```

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<!-- agentManagement contains a description of the agent dynamically used by Jamap-->

<xs:element name="agentManagement">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="accessPoint" type="xs:anyURI"/>
      <xs:element name="WsdFileLocation" type="xs:anyURI"/>
      <xs:element name="dataSubscriptionApplet" type="xs:anyURI"/>
      <xs:element name="subscriptionSheetServlet" type="xs:anyURI"/>
      <!-- getServlet is the servlet for retrieving instantaneous management data as OID with
SNMP-->
      <xs:element name="getServlet" type="xs:anyURI"/>
      <xs:element name="pushDispatcherServlet" type="xs:anyURI"/>
      <xs:element name="agentConfigurationServlet" type="xs:anyURI"/>
      <xs:element ref="informationModels"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="informationModels">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="snmpInformationModel" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="cimInformationModel" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:simpleType name="encodingType">

```

```

<xs:restriction base="xs:string">
  <xs:enumeration value="BER"/>
  <xs:enumeration value="XML"/>
  <xs:enumeration value="serializedJava"/>
  <xs:enumeration value="plainText"/>
</xs:restriction>
</xs:simpleType>

<xs:element name="snmpInformationModel">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="rfc" type="xs:string" minOccurs="1" maxOccurs="unbounded" />
      <xs:element name="snmpv1CommunityString" type="xs:string" minOccurs="1"
maxOccurs="1" />
      <xs:element name="encodingType" type="encodingType" minOccurs="1"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="cimSchemaType">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="cimSchemaName" type="xs:string" minOccurs="1" maxOccurs="1" />
      <xs:element name="cimSchemaVersion" type="xs:string" minOccurs="1" maxOccurs="1"
/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="cimInformationModel">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="cimSchemaType" minOccurs="1" maxOccurs="unbounded" />
      <xs:element name="encodingType" type="encodingType" minOccurs="1"
maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>

```

A nouveau nous avons essayé d'imposer le maximum de restrictions sur les données de configuration contenues dans les fichiers agentManagement.xml. Un exemple de tel fichier est donné en annexe 7.

5.3.3 Localisation des fichiers de configuration

Les fichiers de configuration de JAMAP étaient désormais au format XML et répondaient à des schémas XML bien précis. Nous avons alors introduit la notion de serveur de données, responsable du stockage de tous ces fichiers. Dans la version actuelle de JAMAP ce serveur de données est un serveur Apache mais il est possible d'interfacer une base de données comme une base MySQL très facilement en modifiant quelques lignes dans un fichier de configuration.

Ces fichiers sont à la base du dynamisme de l'application distribuée.

Cette étude de solutions comparatives a été très instructive : en effet nous en sommes arrivés à l'utilisation de schémas XML après avoir constaté les carences des registres UDDI pour notre application de monitoring. Ces schémas nous auront permis de résoudre le problème de découverte dynamique des agents dans JAMAP.

6 Conclusions

Dans cette partie nous allons résumer le travail effectué durant ce PFE et présenter le futur de JAMAP et les contributions que pourraient y apporter de nouveaux étudiants.

6.1 Résumé et contributions

En résumé JAMAP est un prototype de recherche qui tend à devenir une plateforme de gestion de réseaux effective au cours des années, grâce aux développements effectués dans un cadre universitaire. L'accent est mis sur l'innovation et l'utilisation de technologies performantes afin de proposer une nouvelle solution pour gérer les réseaux de demain. La dernière version de cette plateforme est téléchargeable à l'adresse Web suivante : <http://www.cern.ch/jpmf>.

JAMAP ne dépend pas de l'Operating System et a été testé sous Windows XP et Linux Mandrake 9.0. Nous avons utilisé JDK 1.4, Tomcat 4.1.27 et Axis 1.1 comme middleware.

Mes contributions dans ce projet auront été les suivantes (cf. annexe 1) :

- Portage du prototype sur un nouveau serveur d'application et mise à jour des classes dépréciées
- Restructuration des classes de l'application afin de distinguer clairement celles liées à l'agent et celles liées au manager
- Rassemblement des constantes dans une seule classe afin de faciliter l'intégration dans d'autres environnements
- Nouveau design de l'ordonnanceur coté agent afin de minimiser les requêtes SNMP et réduire l'utilisation du CPU en utilisant un système multithread
- Découpage des servlets en plusieurs éléments afin de pouvoir partager certaines parties et offrir un design plus soigné aux futurs développeurs
- Ajout du support XML entre le corrélateur d'évènements et les collecteurs de données
- Ajout du concept de proxy SNMP
- Nouveau design de l'application en stockant les informations liées aux agents dans des fichiers XML validés par des schémas XML et permettant une découverte dynamique des agents par JAMAP
- Nouveau design des pages HTML de configuration de JAMAP en utilisant la technologie Java Server Pages

- Ajout de services Web coté agent afin de renforcer la philosophie d'interopérabilité de JAMAP
- Ajout du concept de profiles permettant à un utilisateur d'appliquer ses souscriptions (OID) à certains agents ou groupes d'agents

6.2 Travail futur

JAMAP demeure un prototype de recherche et pourrait encore subir de nombreuses améliorations. L'aspect de distribution pourrait encore être amélioré en utilisant d'avantage de services Web et le protocole SOAP. Il faudrait aussi découvrir les collecteurs de données et les managers dynamiquement comme pour les agents. Enfin un point important serait le développement d'un corrélateur d'évènements fonctionnant en temps réel et pouvant mémoriser l'image du réseau : cette tâche est complexe et demandera un sérieux travail de design.

L'annexe 8 présente un certains nombres d'évolutions possibles, qui pourraient être implémentées par de futurs étudiants.

7 Références :

- [1] Pour plus d'informations sur les nations représentées, consulter <http://public.web.cern.ch/public/about/aboutCERN.html>
- [2] Site officiel du LHC : <http://lhc-new-homepage.web.cern.ch/lhc-new-homepage/>
- [3] Manuel Delfino, chef de la Division technologie de l'information du CERN
- [4] Sources : Les Robertson, CERN - IT Division , CMS Conference, ITEP – Moscow. SI95 is the SPEC Integer Benchmark 1995 unit, of the "Standard Performance Evaluation Corporation" <http://www.specbench.org/>
- [5] <http://it-div.web.cern.ch/it-div/structure/>
- [6] Internet Engineering Task Force
- [7] interoperable : a word describing systems that can communicate or work with each other
- [8] sources : http://www.improve-institute.com/decouverte_eai_services_web.html
- [9] sources : cours de systèmes de l'ENSIMAG
- [10] sources : <http://www.uddi.org> - UDDI Technical White Paper

Annexe 1 : JAMAP Release Notes

JAMAP 1.3 - September 2003 - Pierre-Alain Doffoel

- * Agents should support a Web Service to allow managers to manage (create, modify, delete) their subscriptions.
- * Add concept of "configuration setting": different users may define different configuration settings (i.e., groups of OIDs subscribed to). We should have a GUI to select what configuration we want to be active among a set of pre-defined ones. We may define configuration settings on a per-subscription basis, per-agent basis, or even for a group of agents.

JAMAP 1.2 - August 2003 - Pierre-Alain Doffoel

- * Created several XML configuration files where we moved many agents settings (static publication). The URIs of the PushDispatcherServlet, AgentConfigurationServlet, SubscriptionSheetServlet, GetServlet and DataSubscriptionApplet are no longer imposed: each agent may follow its own naming policy. The information models supported by each agent are also published there. We have one agentManagement.xml file per agent and a single networkMap.xml file for all management domains.
- * To parse XML, we now use the standard SAX API instead of the jclark package.
- * XML files are validated by using SAX API 1.2 (included in JDK 1.4).
- * Changed the way we bootstrap the management application. We now have two HTML pages. The first one is static and generic (e.g., connect the event manager to all collectors in current management domain). The other is generated dynamically based on data retrieved from an XML configuration file (automated discovery of agents present in domain X).
- * Introduced the concept of data server, a generic machine where all XML configuration files (and their corresponding XSD files for validation) are stored.

JAMAP 1.1 - July 2003 - Pierre-Alain Doffoel

- * Added support for an SNMP proxy to do GETs. We use the same servlets and applets whether we go via a proxy or not.
- * Debugged the real-time spies and monitors when using the DataSubscriptionApplet.
- * Cleaned up site-specific settings.

JAMAP 1.0 - July 2003 - Pierre-Alain Doffoel

- * Added support for XML between the data/notification collectors and the event manager.
- * To avoid using signed applets, tunnel requests to retrieve subscription information from the agents.
- * Checked distribution aspects all over the code and fixed several bugs.
- * Made a clean release of JAMAP -- no support for Web Services.

JAMAP 0.8 - June 2003 - Pierre-Alain Doffoel

- * On the manager side, split the PushedDataCollectorServlet into two servlets: one for push (called the PushedDataCollectorServlet) and another for configuration management (called RuleHandlingServlet).
- * No more blocking reads in the data/notification collectors: use wait/notify instead.
- * Recoded end-of-line delimiters (`\r\n` vs. `\n`) to make them independent of the operating system.

JAMAP 0.7 - June 2003 - Pierre-Alain Doffoel

- * Made the PushScheduler multithreaded and fixed bug with push frequency.
- * On the agent side, split the PushSchedulerServlet into two servlets: one for push (called PushDispatcherServlet) and another for configuration management (called AgentConfigurationServlet).
- * Implemented a smarter algorithm for the PushScheduler. We now have only one PushScheduler for all data collectors. The different push cycles are now synchronized in order to reduce network and end-host overload (this can be customized by changing the tolerance value in Constants.Java).
- * Made it possible to subscribe or unsubscribe to data in real-time.
- * Added the possibility to either debug the project locally using JBuilder or run it with Tomcat (by changing a boolean variable in Constants.Java).

JAMAP 0.6 - May 2003 - Pierre-Alain Doffoel

- * Reorganized packages to better highlight which classes are used on the agent or the manager side.
- * Added output messages when starting the application.
- * Ported the project from Jakarta Tomcat 4.0 to 4.1.
- * All constants are now in Constants.Java: nothing in the rest of the code is site-specific.

JAMAP 0.5 - May 2003 - Pierre-Alain Doffoel

- * Ported project from Visual Cafe to JBuilder.
- * Ported from Jigsaw 2.2.0 to Jakarta Tomcat 4.0. The buffering problem experienced with Jigsaw has disappeared in Tomcat.
- * Ported from JDK 1.3 to JDK 1.4.1.
- * Recoded handling of notifications.
- * Moved many constants from the code to jamap.Constants.Java.
- * Temporary workaround to handle SNMP columns.
- * Added many error messages.
- * Fixed some security problems with applets.
- * Simplified installation and customization of JAMAP.

JAMAP 0.4 - July 2001 - Claire Ledrich

- * Added support for multiple agents, multiple data collectors, and multiple notification collectors.
- * Added support for XML.
- * Re-designed the event manager and the notification handling.
- * Re-designed the edition of rules and their mapping to incoming data or events.
- * Decreased network overhead by making several optimizations.
- * Decreased the number of pushed data analyzers running on the data collectors.
- * Encapsulate multiple data per MIME part.

JAMAP 0.3 - October 2000 - J.P. Martin-Flatin

- * Packaged for external release.

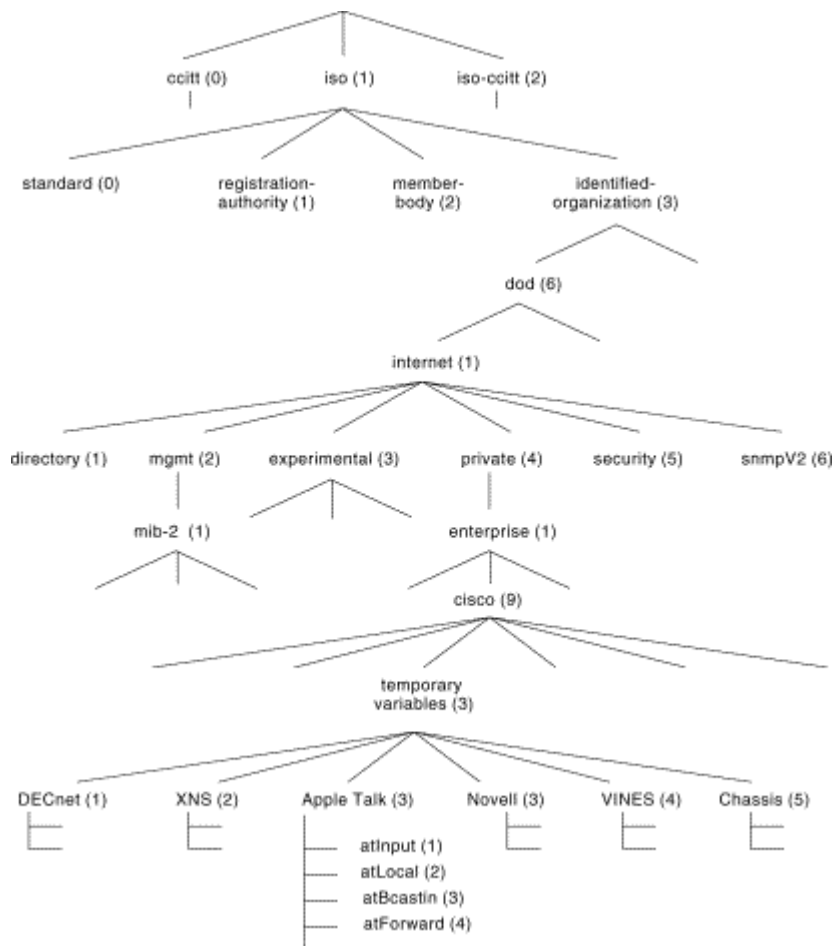
JAMAP 0.2 - August 1999 - J.P. Martin-Flatin

- * Debugged the spies
- * Cleaned up the code.

JAMAP 0.1 - March 1999 - Laurent Bovet

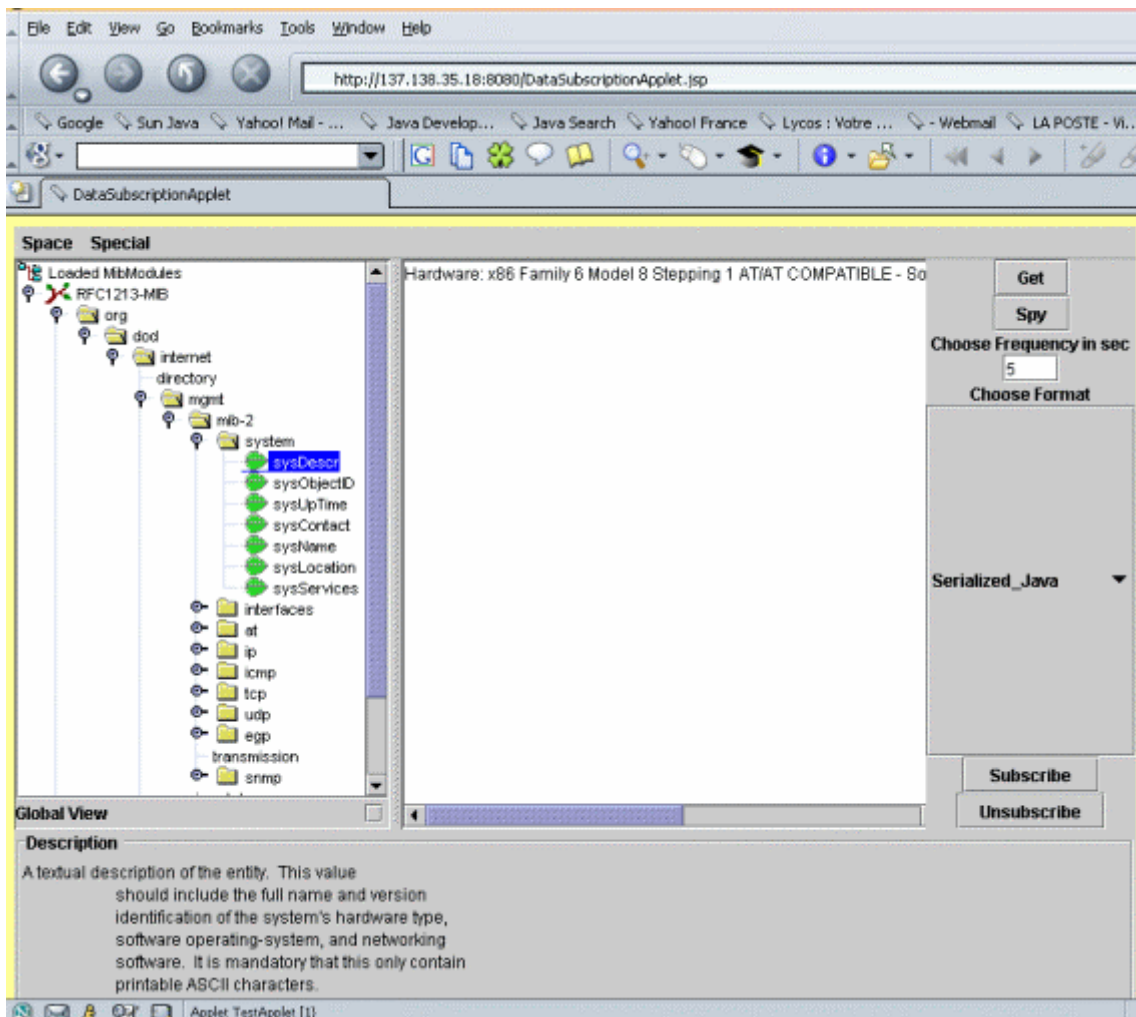
- * Initial internal release.

Annexe 2 : exemple de MIB (SNMP)



Sources : Cisco (<http://www.cisco.com>)

Annexe 3 : Exemple de l'applet de souscription dans JAMAP



Annexe 4 : Exemple de servlet dans JAMAP

```

package jamap.agent.servlet;

import Java.io.*;
import Java.net.*;
import Javax.servlet.*;
import Javax.servlet.http.*;
import com.adventnet.snmp.beans.*;

public class GetTable extends HttpServlet {

    SnmpTable table;

    public void init(ServletConfig config) throws ServletException {

        super.init(config);

        String hostName;

        table = new SnmpTable();

        try {
            hostName = InetAddress.getLocalHost().getHostName();
        } catch(UnknownHostException e) {
            e.printStackTrace();
            hostName = null;
            System.exit(-1);
        }

        table.setTargetHost(hostName);
        table.setTargetPort(jamap.share.Constants.snmpPort);
        table.setDataTypes(SnmpTable.STRING_DATA);

        try {
            table.loadMibs(jamap.share.Constants.preloadedMibs);
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

    public void doGet (HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

        try {
            table.setTableOID(request.getQueryString());
            table.run();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}

```

```
jamap.share.other.Table result =
    new jamap.share.other.Table(table.getRowCount() + 1, table.getColumnCount());

int i,j;

for(i=0; i<table.getColumnCount(); i++) {
    result.setValueAt(table.getColumnName(i), 0, i);

    for(j=0; j<table.getRowCount(); j++)
        result.setValueAt(table.getValueAt(j,i), j+1, i);
    }

response.setContentType(jamap.share.Constants.mimeResponse);
ObjectOutputStream out = new ObjectOutputStream(response.getOutputStream());

out.writeObject(result);
out.flush();
out.close();
}
}
```

Annexe 5 : Timing et real-time ensemble

```
package jamap.agent.scheduling;
```

```
public class WaitThread
```

```
    extends Thread {
```

```
    long timeToSleep;
```

```
    public WaitThread(long timeToSleep) {
```

```
        this.timeToSleep = timeToSleep;
```

```
    }
```

```
    public synchronized void stopThread() {
```

```
        notify();
```

```
    }
```

```
    public synchronized void run() {
```

```
        try {
```

```
            wait(timeToSleep);
```

```
        }
```

```
        catch (InterruptedException ex) {
```

```
            System.out.println(
```

```
                "WaitThread was interrupted to take into account the new subscriptions");
```

```
        }
```

```
    }
```

```
}
```

Annexe 6 : Exemple de fichier networkMap.xml

```
<networkMap>

<managementDomain>
<name>CERN</name>

<agent>
<ipv4Address>
137.138.35.18
</ipv4Address>
<ipv4CollectorAddress>137.138.35.18</ipv4CollectorAddress>
<agentManagement>
http://137.138.35.18:8080/agent/agentManagement137.138.35.18.xml
</agentManagement>
</agent>

<agent>
<ipv4Address>
137.138.35.32
</ipv4Address>
<ipv4CollectorAddress>137.138.35.18</ipv4CollectorAddress>
<proxy>
<ipv4Address>
137.138.35.18
</ipv4Address>
</proxy>
<agentManagement>
http://137.138.35.18:8080/agent/agentManagement137.138.35.32.xml
</agentManagement>
</agent>

<agent>
<ipv4Address>
192.91.239.254
</ipv4Address>
<ipv4CollectorAddress>137.138.35.18</ipv4CollectorAddress>
<proxy>
<ipv4Address>
137.138.35.18
</ipv4Address>
</proxy>
<agentManagement>
http://137.138.35.18:8080/agent/agentManagement192.91.239.254.xml
</agentManagement>
</agent>

</managementDomain>
</networkMap>
```


Annexe 7 : Exemple de fichier agentManagement.xml

```
<agentManagement>

<accessPoint></accessPoint>
<WsdFileLocation></WsdFileLocation>
<dataSubscriptionApplet>http://137.138.35.18:8080/DataSubscriptionApplet.jsp</dataSubscriptionApplet>
<subscriptionSheetServlet>http://137.138.35.18:8080/servlet/jamap.servlet.SubscriptionSheetServlet?</subscriptionSheetServlet>
<getServlet>http://137.138.35.18:8080/servlet/jamap.servlet.Get?</getServlet>
<pushDispatcherServlet>http://137.138.35.18:8080/servlet/jamap.servlet.PushDispatcherServlet?</pushDispatcherServlet>
<agentConfigurationServlet>http://137.138.35.18:8080/servlet/jamap.servlet.AgentConfigurationServlet?</agentConfigurationServlet>

<informationModels>
<snmpInformationModel>
<rfc>/mibs/rfc1213-mib.txt</rfc>
<snmpv1CommunityString>public</snmpv1CommunityString>
<encodingType>plainText</encodingType>
</snmpInformationModel>
</informationModels>

</agentManagement>
```

Annexe 8 : Travail futur possible

*** JAMAP 1.4 ***

- * Add support for low-level SOAP (MIME header, Unit consists of SOAP envelope + body).
- * To work out what URIs the event manager should invoke on the data and notification collectors, we should use information retrieved from an XML configuration file instead of constants stored in Constants.Java (i.e., we should do between the event manager and the collectors as we currently do between collectors and agents). The XML file should be parsed by a validating parser.
- * For all units, manage a vector of Timestamps (one timestamp per agent, one per collector, etc.).

*** JAMAP 1.5 ***

- * Add support for JDBC. Allow administrator to store time series of SNMP OIDs in either a relational database or flat files.
- * Add support for an SNMP proxy to do SETs.

*** JAMAP 2.0 ***

- * General code cleanup.
- * Investigate whether we should use "ant" to configure Constants.Java and site-specific paths.

*** Future releases of JAMAP ***

- * Add support for an SNMP proxy to deal with SNMPv2 notifications.
- * Add proper support for SNMP columnar objects (the workaround currently in place only shows the first column of a columnar object).
- * For each agent, store the DataSubscriptionTable and the NotificationSubscriptionTable on a remote Data Server as well (not only on the agent), and allow the agent to retrieve these tables upon reboot.
- * Upgrade AdventNet package to the latest freely available code release.
- * Add a rule mapper for the event correlator. Currently, we only have a rule mapper for the pushed data analyzers.
- * InstantaneousPDA should be able to manage a vector of rules and do

multiplexing (i.e., for a given OID coming at a given frequency from a given agent, apply N rules instead of just one).

- * Replace the use of public attributes by set and get methods and private attributes.
- * Use signed applets to trust the applets inside a web browser
- * Reduce memory usage by multithreading rules created from the templates.
- * Define rule templates for spatial and spatio-temporal rules.
- * Distribute the rule management to different DataCollectors.
- * Add a button to "index.html" in order to stop the application in a clean way.
- * Should the rule editor be transformed into a rule template editor? Think about it.
- * Add a directory tree for storing spatial rules. Devise a naming scheme.
- * Define a smarter RuleMapper for rules other than InstantaneousRules.
- * Collectors should send an event (and only one) when they receive one or several Units with an unknown format from the same agent.
- * On the PushedDataCollector and EventCorrelator, open an HTTP connection to the log servlet once and for all and never close the connection. This means adding a PushForwardConsumer on the EventCorrelator (for eventLog) and on the PushedDataCollector (for dataLog).
- * Distribute event correlation. This is a hard problem and may take a while!

ANNEE SCOLAIRE 2002/2003

PROJET DE TROISIEME ANNEE

RAPPORT FINAL

SEPTEMBRE 2003

TITRE DU PROJET :

Monitoring of a Transatlantic Gigabit Network

RESUME :

La complexité et l'hétérogénéité des réseaux IP impose l'utilisation d'outils de gestion de réseaux. Aujourd'hui c'est encore le protocole SNMP qui est le plus communément utilisé. Et pourtant, avec la croissance des technologies Web et des applications distribuées, les manques de ce protocole sont de plus en plus évidents : il paraît indispensable de concevoir de nouveaux systèmes de gestion robustes et automatisés. Durant ce stage, nous avons poursuivi le développement de JAMAP, un prototype de recherche qui implémente l'architecture WIMA décrite par J-P Martin-Flatin dans sa thèse de doctorat. Ce travail a donné naissance à la version 1.3 de JAMAP, insistant sur les aspects de distribution et d'automatisation.

Proposé par l'Entreprise :

CERN
Centre Européen de Recherche Nucléaire
CH – 1211 Genève 23

MOTS - CLES :

Network Monitoring, Web Services, UDDI, Java, Jbuilder, XML Schema, SOAP, WSDL, JDBC, UML, SNMP, CERN, JAMAP, WIMA.