Impact des modèles Web-Services sur les architectures de supervision de réseaux

Pierre Humbert

Impact des modèles Web Services sur les architectures de supervision de réseaux

MÉMOIRE

30 Juin 2003

pour l'obtention du

DEA de l'Université Henri Poincaré – Nancy I

(Spécialité Informatique)

par

Pierre HUMBERT

Composition du jury

Membres du jury :	Dominique Méry
	Didier Galmiche
	Noëlle Carbonell
Encadrant :	Olivier Festor

Laboratoire Lorrain de Recherche en Informatique et ses Applications — Equipe MADYNES

Remerciements

Je tiens à remercier toutes les personnes de l'équipe MADYNES du laboratoire LORIA pour m'avoir accueilli lors de ce stage et permis d'acquérir une première expérience de la recherche très agréable.

Je tiens à remercier particulièrement :
- M. Olivier Festor, chargé de recherche à l'INRIA, pour m'avoir permis de travailler au sein de son équipe, pour m'avoir dirigé et aidé pendant toute la durée du stage.
- Ma famille et mes amis pour leur soutien, leur aide et leurs encouragements.

Table des matières

Table des figures

Introduction

Les services Web et la technologie XML (eXtended Markup Language) [29] s'imposent aujourd'hui dans de nombreux domaines de services de l'Internet comme le commerce électronique, le multimédia ou de façon plus générale, dans l'échange d'informations entre applications. La supervision des réseaux et des services n'échappe pas à cette évolution.

La supervision des réseaux et des services consiste à monitorer (c'est-à-dire à observer et à analyser l'état et le comportement des composants du réseau) et à contrôler (à modifier les paramètres des composants du réseau et leur faire exécuter des actions prédéfinies) les éléments matériels et logiciels du réseau (ex : routeurs, terminaux, programmes ...) afin qu'ils satisfassent les demandes des utilisateurs et les contraintes du fournisseur. La supervision repose sur cinq aires fonctionnelles connues sous l'appellation FCAPS : la Gestion de Fautes, qui consiste à détecter, isoler et corriger des opérations anormales, généralement à partir d'évènements générés par le réseau ; la Gestion de Configuration, qui consiste à identifier, collecter et fournir des données de configuration aux entités managées en garantissant la continuité de leur service d'intercomrnunication des services ; la Gestion de la Comptabilité (Accounting), qui consiste à évaluer le coût d'utilisation des ressources managées ; la Gestion de Performances, qui consiste à évaluer le comportement des ressources managées et le test des activités de communication ; enfin, la Gestion de la Sécurité qui consiste à assurer la sécurité du réseau et à protéger les ressources managées.

Le modèle de document XML semble à priori bien approprié pour la fonction de configuration d'éléments, mais couplé aux Web services, il représente une véritable révolution par rapport au modèle gestionnaire/agent utilisé en supervision depuis une décennie et qui est relativement inadapté aux besoins grandissants d'intégration et d'interopérabilité. Dans le cadre de ce DEA, nous proposons de nouveaux modèles de supervision et nous fournissons une première étude sur la généralisation de l'approche Web services à toutes les fonctions de gestion.

Dans un premier temps, nous nous attacherons dans le chapitre 1 à décrire les travaux de recherche actuels tirant partie des avantages de la technologie XML ainsi que leurs limites. La présentation, dans le chapitre 2, des Web services et d'Active XML, un modèle développé dans le projet GEMO à l'INRIA Rocquencourt, sera la base d'une nouvelle approche plus dynamique et plus autonome de la supervision. Cette architecture sera présentée dans le chapitre 3. Afin de valider les idées proposées, nous présenterons dans le chapitre 4 un prototype fonctionnel de supervision de Firewalls. Enfin, nous conclurons le travail par les perspectives de recherches envisagées.

Chapitre 1

Etat de l'art

Dans ce chapitre nous présentons les différentes solutions et travaux de recherche proposés dans le cadre de l'utilisation des modèles XML et Web Services en supervision.

Le plan de management des systèmes actuels est basé sur le modèle Manager/Agent (voir Figure 1.1). À chaque ressource managée est attaché un Agent qui permet d'accéder à sa base d'informations de gestion. Le manager interroge l'agent grâce à un modèle d'information commun (ie. structure des informations, relations entre les données, types des données ...) et à travers un protocole de communication (ie. transport et encodage des informations, opérations de management, adressage des données ...) . De plus, l'agent peut reporter des alarmes au manager.

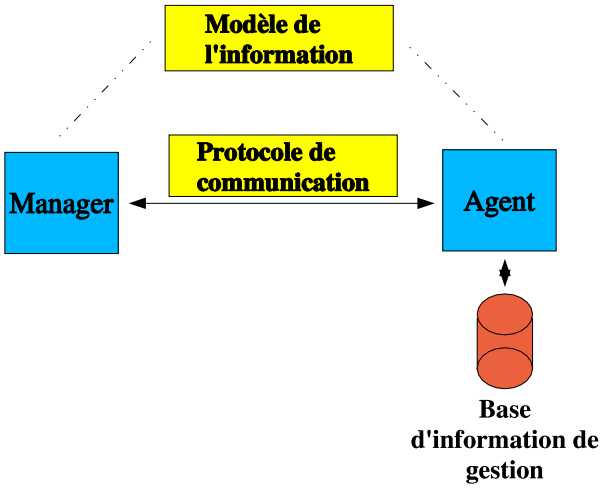

FIG. 1.1 – Schéma Manager/Agent

Le protocole SNMP (Simple Network Management Protocol) [16] est le système de gestion le plus utilisé actuellement. Il est basé sur le modèle Manager/Agent.
La base d'information de gestion est matérialisé sous la forme d'une MIB (Management Information Base) comportant les instances des objets à manager. Il s'agit d'une structure arborescente à partir de laquelle les constructeurs d'équipements réseaux peuvent ajouter des branches pour y inclure des informations spécifiques à leur matériel.

Le modèle d'informations est précisé par la norme SMI (Structure of Management Information) qui correspond à une extension du modèle ASN.1 ; l'identification des objets de ce modèle est effectuée grâce à un OID (Object Identifier) représenté en notation pointée (ex : 1.3.6.2.1.1.13.0 correspond à l'objet SysUpTime).
Le modèle de communication proposé par SNMP assure le transport d'informations entre le manager et l'agent au moyen de paquet UDP. Il comprend 5 opérations simples : GetRequest, GetNextRequest, SetRequest, GetResponse et Trap.

Nous avons constitué un schéma global d'intégration (voir Figure 1.2) du plan de management tel qu'il est défini aujourd'hui, à savoir basé sur un modèle Manager/Agent.

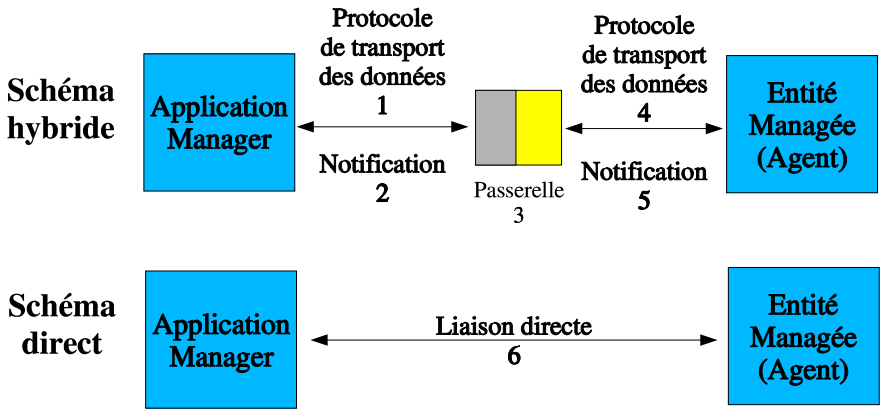

FIG. 1.2 – Schéma global d'intégration

Comme nous le verrons dans la suite, les différentes travaux de recherche étudiés s'inscrivent dans l'un des deux schémas présentés dans la figure 1.2, à savoir soit un schéma hybride mettant en œuvre une passerelle entre un système de management existant (ex : SNMP) et un nouveau système de management, soit un schéma direct entre le manager et l'agent.

L'apparition de la technologie XML ainsi que ses qualités en matière d'intégration et d'interopérabilité ont très justement orienté la recherche en matière de supervision. Cette technologie peut prendre place à différents niveaux au sein d'une architecture de supervision :

- pour la représentation des données de management,
- pour le transport des données entre les entités du système de management,
- pour définir des systèmes asynchrones de notification,
- au sein d'une passerelle dans le cas de systèmes hybrides.

Dans ce cadre, la plupart des technologies associées à XML (DOM, XPath, XQuery,...) sont mises à contribution.

1.1 La recherche sur l'utilisation d'XML en management de réseau

1.1.1 Architecture de management

WIMA

L'architecture WIMA (Web-based Integrated Management Architecture), proposée par Martin-Flatin dans sa thèse [19], est basée sur les technologies standard du Web. Elle repose sur un modèle organisationnel diptyque utilisant d'une part un modèle de type "push" pour la gestion régulière (i.e. le monitoring sur une longue durée) et la notification évènementielle et d'autre part un modèle de type « pull» pour le management ad-hoc (i.e. recherche de données sur une courte durée).

Dans cette architecture, les modèles de communication et d'information sont dissociés, ce qui lui permet de supporter tous les modèles d'informations (SNMP, CIM (voir section WBEM), ...). Le modèle de communication est caractérisé par l'utilisation de connexions HTTP persistantes entre agent et manager : les messages HTTP sont structurés par des parties MIME (Multimedia Internal Mail Exchange) [20]. Chacune de ces parties peut être un document XML, un fichier binaire,...

Dans WIMA, il est recommandé d'utiliser XML pour représenter les données de gestion en transit. A ce sujet, WIMA supporte deux types de mapping entre XML et le modèle d'informations interfacé (SNMP, CIM, ...) : au niveau du modèle ou au niveau du méta-modèle.

Au niveau du modèle, un module SMI MIB (ou CIM schema) particulier est mappé en une DTD (Document Type Definition) spécifique : les balises XML et leurs attributs ont les mêmes noms que les variables de la MIB SNMP (classes et propriétés CIM).

```
Ex: <interface>
      <bandwidth type=''string''>100 Mbit/s</bandwidth>
    </interface>
```

Au niveau du méta-modèle, la DTD est générique et identique pour toutes les MIBs SNMP et shémas CIM. Les éléments XML de la DTD portent des noms génériques tels que class ou property.

```
Ex: <class name=''interface''>
      <property name=''bandwidth'' type=''string''>
        <value>100 Mbit/s</value>
      </property>
    </class>
```

Chacun de ces deux modèles présente des avantages et des inconvénients opposés : le principal avantage du mapping de niveau modèle est la lisibilité des documents XML respectant la DTD. Cependant, le mapping est difficilement automatisable de part les différences entre les différents modèles d'informations supportés. À l'inverse, le mapping au niveau méta modèle permet l'automatisation de la traduction des modèles d'information supportés mais rend la DTD moins lisible et plus verbeuse. De plus, la validation de données XML à l'aide de ce type de DTD ne peut être que basique.
WIMA s'apparente à une architecture hybride (cf. Shéma d'intégration général 1.2).

POSTECH

Tout comme WIMA, l'équipe de POSTECH propose une architecture de management de type manager/agents [15] basée sur l'utilisation de serveurs Web embarqués. L'architecture est décrite sur la figure 1.3 : des agents WBM échangent des informations avec le manager WBM à travers le protocole XML/HTTP. Ce dernier collecte les données de management et les stockent dans un arbre DOM pour permettre leur analyse à long terme. Le manager WBM correspond donc à un client Web pour le management de réseau et l'agent WBM à un serveur Web spécialisé pour fournir des données de management au format XML.

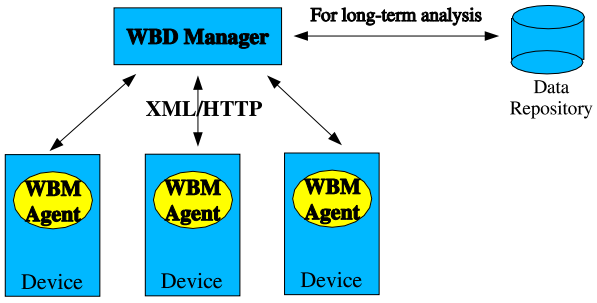

FIG. 1.3 – Architecture de management Web proposée par l'équipe POSTECH [15]

Dans ce cas, le serveur Web est utilisé à la fois pour fournir les opérations à travers une interface Web (pour l'échange synchrone des informations de management avec les agents) et pour réceptionner les messages asynchrones des agents (alarmes et données de management planifiées en mode push).

Dans ce modèle, XML est utilisé à la fois pour la modélisation des informations de management et pour la communication entre manager et agents.

Le modèle d'information utilisé est XML Schéma.
Le modèle de communication repose entièrement sur le protocole HTTP : HTTP GET pour l'obtention d'une donnée de management, HTTP POST pour la mise à jour. Les données de management sont idéntifiées par XPATH [6].

Cette architecture est validée par un prototype XGEMS (XML-based Global Element Management System).

Cette architecture correspond donc à une architecture directe (cf. shéma d'intégration global 1.2), mais la seconde approche de passerelle XML/SNMP, proposée par l'équipe et présentée plus loin, peut également être combinée : dans ce cas, on peut considérer l'approche comme directe et hybride à la fois.

1.1.2 Intégration XML/SNMP

Passerelle XML/SNMP

La technologie XML commence à être utilisée dans les nouveaux modèles de gestion pour ses qualités d'intégration et d'interopérabilité. D'autre part, SNMP est actuellement le modèle le plus utilisé. Pour permettre une compatibilité entre les nouveaux modèles et ceux dèjà utilisés, il est nécessaire de concevoir des passerelles XML/SNMP (Cf. Figure 1.3). De telles passerelles consistent en une double adaptation :

- une traduction du modèle d'information de SNMP (Agent) vers un modèle d'information XML (Manager)
- une translation des opérations effectuées dans un modèle vers l'autre.

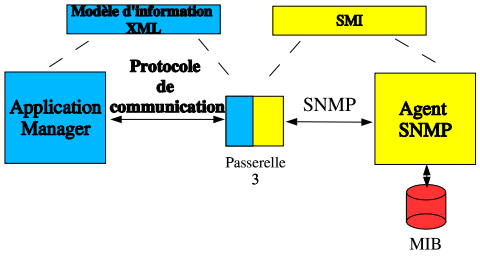

FIG. 1.4 – Intégration XML/SNMP

Dans la suite de cette partie seront présentées trois passerelles proposées par l'équipe du centre de recherche Avaya aux Etats Unis, l'équipe de l'Université Postech en Korée et enfin celle de Strauss à l'Université de Braunschweig en Allemagne.

AVAYA Cette équipe propose une interface de management basée sur XML [24] pour lire et écrire les informations de management dans des agents SNMP (Cf. Figure 1.6) . Le modèle d'informations utilisé du côté XML est défini par le langage XML Schema. Ce prototype consiste en 3 parties :

- Un outil permettant la génération automatique de schémas XML à partir des modules d'information SMI de SNMP. Ce « mapping » consiste en une traduction rigoureuse des modules MIB (cf Figure 1.5).

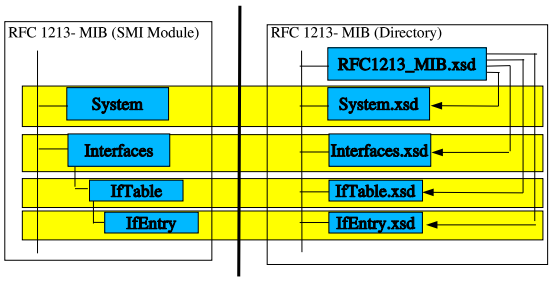

FIG. 1.5 – Mapping rigoureux des modules MIB en XML Schéma proposé par l'équipe Avaya

- Un protocole de communication (XML-RPC) pour retrouver et modifier les informations contenues dans les Mibs. Le protocole de messages définit un schéma XML pour l'ensemble

des commandes d'interrogation (GET,SET,LIST,CREATE,DELETE) et identifie les variables de la MIB grâce au modèle XPATH.
– Un adapteur pour retrouver et modifier les informations des périphériques au format MIB à partir de données exprimées sous forme XML (translation des opérations)

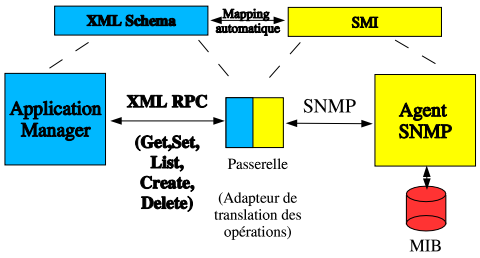

FIG. 1.6 – Intégration passerelle Avaya

POSTECH Cette équipe propose, dans le cadre de son travail sur une passerelle XML/SNMP [32], un algorithme et une implémentation permettant la traduction d'une MIB SNMP en XML. Le modèle de traduction est simple : tout comme dans l'approche précédente, les modules de la MIB sont traduits en XML Schemas.

Pour permettre l'intégration des périphériques SNMP managés, ils proposent 3 méthodes permettant de réaliser la translation de l'interaction présentant chacune des avantages et des inconvénients en fonction du système dans lequel elles sont intégrées (Cf. Figure 1.7) :

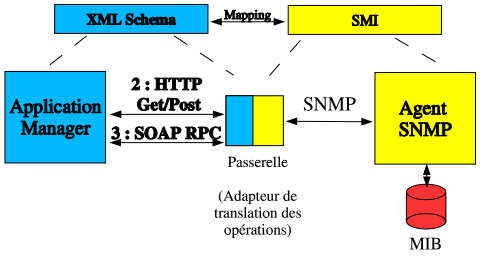

FIG. 1.7 – Intégration passerelle Postech

– La première approche élimine le besoin de communication entre le manager et la passerelle en fusionnant la passerelle à l'application de management
– La seconde approche utilise le protocole HTTP pour la communication entre le manager et la passerelle : HTTP GET pour l'obtention d'une donnée de management, HTTP POST pour la mise à jour.
– La troisième approche utilise le protocole SOAP RPC en guise de protocole de management avec le manager.

Dans les trois cas, le modèle d'information utilisé est XML Schéma.
De plus amples détails sont donnés en Annexe A.

Strauss F. Strauss a récemment proposé une nouvelle approche de traduction d'une MIB SNMP en XML Schéma, mise en oeuvre dans le cadre d'un prototype de passerelle XML/SNMP [27]. Tout comme les approches précédentes, les modules MIB sont traduits en schémas XML. Cependant, à l'inverse de ces approches qui réalisent un « mapping» strict, cette approche tente de représenter les données de façon à préserver la lisibilité et la facilité de traitement des documents XML. En particulier il s'agit d'obtenir des documents relativement plats et d'éviter ainsi les profondes hiérarchies d'un mapping direct des OIDs. De la même façon, le schéma XML est conçu de façon à permettre le stockage de données de plusieurs agents, prélevées à différentes dates, dans un même document XML. Enfin, les éléments textuels sont décrits de façon lisible par l'homme (ex : les String sont écrite sous forme ASCII).
L'identification des modules MIB se fait par des espaces de noms.

```
Exemple : <?xml version=''1.0''?>
          <snmp-data xmlns=''...''>
            <context agent=''ciscobs.de'' community=''public'' port=''161''
             time=''2002-12-12T23:42+0100''>
              <IF-MIB:interfaces>
                <IF-MIB:ifNumber>7</IF-MIB:ifNumber>
              </IF-MIB:interfaces>
            [...]
              <IF-MIB:ifEntry ifIndex=''2''>
                <IF-MIB:ifDescr>FastEthernet0/0</IF-MIB:ifDescr>
                <IF-MIB:ifSpeed>10000000000</IF-MIB:ifSpeed>
                [...]
              </IF-MIB:ifEntry>
            [...]
            </context>
          </snmp-data>
```

Le prototype de passerelle XML/SNMP implémentant cette approche est basé sur un traducteur nommé mibdump qui génère une séquence d'opérations Snmp (GetNext) pour retrouver les données et les traduire en élément XML conformément au schéma. Une passerelle complète est en cours de développement permettant de traduire des requêtes HTTP pour les documents XML conformes aux schémas XML étudiés. Cette partie du prototype est similaire à la deuxième approche de POSTECH.

SCLI

SCLI [25], proposé en Janvier 2001, est un outil qui implémente une interface de commandes en ligne pour interagir avec des agents SNMP de façon interactive. SCLI n'est pas un outil générique pour interfacer n'importe quelles données de MIB. Au contraire, il a été conçu pour reconnaître la structure et la sémantique d'un grand nombre de modules MIB. Il est ainsi capable de présenter et d'accepter des informations décrites sous une forme plus compréhensible par l'utilisateur. Par exemple, lorsqu'un utilisateur demande des informations sur une interface spécifique, elle peut être nommée plutôt que d'utiliser la notion de variable ifIndex de SNMP. De plus, les données sont formatées sous forme de tables où les nombres sont associés à la bonne unité et les items sont rassemblés depuis différentes MIBs (ex IF-MIB,IP-MIB,ENTITY-MIB).

L'utilisation de cet outil facilite le travail des utilisateurs inexpérimentés, même si l'outil nécessite une personne expérimentée pour développer de nouveaux modes SCLI.

En plus de la forme textuelle, SCLI permet de l'extraction des informations sous une forme XML. En ce sens, il est possible, par exemple, de traiter ensuite les données avec une feuille de style XSL.

1.2 Au niveau de la standardisation

1.2.1 WBEM

Web-Based Enterprise Management (WBEM) [21] est une initiative du DMTF (Data Management Task Force) qui a pour objectif de fournir un cadre unificateur pour la gestion des réseaux, des services et des applications au niveau de l'entreprise afin d'assurer l'interopérabilité dans le plan de gestion.
WBEM consiste en un modèle commun de l'information CIM (Common Information Model), un modèle de communication basé sur XML pour l'encodage de l'information (une DTD pour la représentation du CIM) et http pour le transport proprement dit. De plus WBEM fournit une spécification des opérations CIM permettant l'accès aux données. CIM fournit un modèle de l'information orienté objet et les schémas CIM sont implémentés pour manager les ressources réseau tels que les switchs et les routeurs. WBEM est actuellement en train d'être mis à jour pour inclure les nouveaux standards émergents tels quel SOAP [11].

1.2.2 IETF NetConf

Durant le 54ème meeting IETF à Yokohama en juillet 2002 [31], une session concernant le management de configuration basé sur XML a été tenue. Le but était de répertorier les besoins en matière de management de configuration des opérateurs, d'identifier les avantages et les inconvénients des technologies actuelles et d'évaluer quelques solutions technologiques basées sur XML telles que SOAP/WSDL, SyncML, WBEM, JUNOScript. Il y a quelques Draft Internet qui représentent le point de départ pour répondre à ces besoins.

Parmis ces derniers, deux protocoles ont été présentés :

Enns [23]

Il s'agit d'une proposition d'un mécanisme simple permettant de manager un périphérique. Ainsi les données d'état, celles de configuration et les notifications systèmes peuvent être retrouvées et de nouvelles données de configuration peuvent être téléchargées et manipulées. Ce protocole est orienté connexion et est basé sur le mécanisme XML-RPC : un client encode un RPC (Appel de procédure à distance) en XML, l'envoi au serveur, qui lui répond par un mécanisme similaire ; seul un ensemble restreint d'opérations (partielles ou totales) sont fournies pour manager un périphérique et retrouver des informations d'état : get-config, edit-config, copy-config, delete-config, get-state,kill-session.

Ce protocole nécessite deux canaux de communication distincts, plus un troisième optionnel :
- Le canal de management : traite les informations pour manager les sessions. Il permet d'annoncer les possibilités de chaque peer, de manager (annuler) les RPCs du canal des opérations et d'envoyer des rapports de progression ;
- Le canal des opérations : traite les séries de RPC ;
- Le canal de notifications : traite les notifications asynchrones.

Le protocole offre en outre un mécanisme permettant au client de découvrir l'ensemble des extensions supportées par le serveur (blocage de configuration, validation/configuration temporaire, notifications ...) lors d'une phase initiale d'échange.

XCLI

XCLI [17] correspond à un adaptateur pour la CLI (Command Line Interface) qui utilise des templates XML pour représenter un groupe de commandes CLI. Lors de l'invocation d'une commande par le mécanisme de XCLI, le template correspondant est chargé en mémoire par l'API X-CLI associée et est ensuite converti en un ensemble de commande CLI en passant les arguments requis. Les commandes CLI sont alors exécutées sur le périphérique à travers le protocole Telnet. Le procédé est également utilisé pour la connexion aux ressources managées grâce à un template de type DCPF (Device Connection Procedure File).

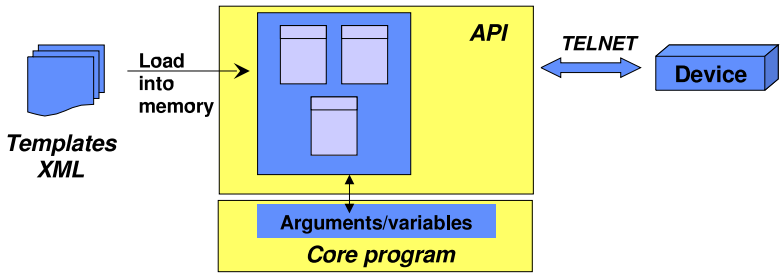

Fig. 1.8 – L'adaptateur XCLI

XCLI élimine ainsi le besoin de programmation de la couche dépendante des instructions CLI (de l'OS) grâce à l'externalisation de la logique de l'application.

1.3 Dans le monde industriel

1.3.1 Junos+JunoScript

Développée par la société Juniper Networks en janvier 2001, Junoscript [26] a été créé en vue de faciliter le développement d'applications et de scripts personnalisés de façon simple et rapide en utilisant le langage XML.

Il permet en particulier à des applications clientes d'accéder aux données de configuration et opérationnelles d'un routeur Juniper à travers l'échange de requêtes et de réponses formées par des documents XML (XML-RPC). La grammaire de ces documents est fournie par Juniper sous la forme de DTD et de XML Schéma. Ce modèle simple permet de réduire les coûts d'implémentation et l'impact sur le périphérique managé. De nombreux protocoles peuvent être utilisés pour établir la connexion entre l'application cliente et le serveur JunoScript (Telnet, SSL, SSH ...). Les messages envoyés par le client constituent des requêtes encapsulées dans des balises "rpc" à partir desquelles le serveur renvoie sa réponse dans des balises "rpc-reply". Le contenu correspond à l'action à effectuer (authentification du client, obtention de configuration, modification de configuration, blocage exclusif d'une configuration, ...). L'ensemble des données de configuration peuvent être représentées dans les messages à la fois comme des éléments XML ou dans un format texte utilisé également par la CLI de JUNOS.

1.3.2 SyncML DM

Le standard SyncML [4] a été créé en 2001 par Ericsson, IBM, Lotus, Matsushita, Motorola, Nokia, Openwave, Starfish et Symbian. Il permet la synchronisation entre un client et un serveur de bases de données décrites par un modèle nom/valeur. En fonctionnement normal, le client et le serveur gardent en mémoire l'ensemble de toutes les opérations effectuées depuis la dernière bonne synchronisation et se les échangent ensuite. Les opérations que l'on peut ainsi effectuer sont : PUT, GET, ADD, COPY, DELETE, REPLACE, auquel s'ajoutent des commandes permettant d'assurer l'atomicité d'un groupe de commandes (Atomic, sequence), d'exécution et d'alerte (RPC).

Le standard SyncML a initié le modèle SyncML DM [3] en tant que standard industriel ouvert et universel pour le management à distance de terminaux mobiles. Ce dernier reprend les principales caractéristiques de SyncML (DTD, opérations, framework de sécurité ...) en se limitant à une synchronisation unidirectionnelle (Serveur vers Client).

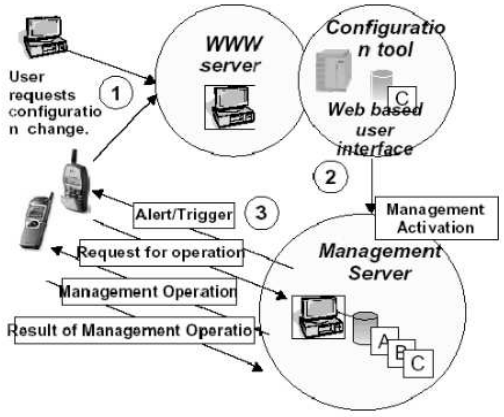

FIG. 1.9 – Interaction dans le modèle SyncML DM [3]

L'interaction entre les terminaux à configurer et le serveur central de configuration se déroule selon le schéma 1.9 : tout d'abord le terminal enregistre sa demande de reconfiguration (1) auprès d'un serveur web. Lorsque la configuration à été mise à jour sur le serveur pour le manager (2), le serveur alerte le terminal (3). Le protocole d'échange des opérations de management peut alors débuter. La particularité de ce schéma d'interaction est qu'il permet aux terminaux de ne pas rester continuellement à l'écoute ; il est de plus possible d'utiliser des protocoles de communication différents pour l'échange des données de management (ex : WAP, HTTP, HTTPs) et la notification (ex : SMS).

Par ailleurs, un framework de description de ressources a été mis en place de façon à gérer les fonctionnalités propriétaires des différents équipementiers.

1.3.3 Cisco's Configuration Registrar

Le Cisco Configuration Registrar [1] est un système basé sur les technologies Web pour distribuer de façon automatique les fichiers de configuration aux périphériques réseau Cisco IOS. Ce système collabore avec l'agent de configuration du CNS (Cisco Networking Services) situé sur chaque périphérique. Le système fournit un fichier de configuration initial au périphérique Cisco lorsqu'il démarre pour la première fois sur le réseau. Ce fichier est constitué à partir d'un template de configuration (fichier texte comprenant des commandes CLI) faisant référence à des informations spécifiques au périphérique stockées dans un annuaire. Le protocole http est utilisé pour dialoguer avec l'agent ainsi que pour transférer les données de configuration au format XML qui sera parsé et interprété par l'agent client.

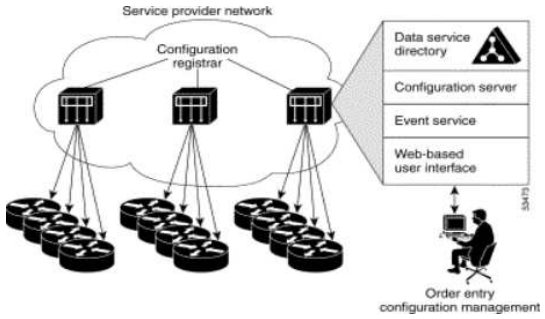

FIG. 1.10 – Architecture Cisco Configuration Registrar [1]

1.4 Synthèse

Différents systèmes ont été présentés dans ce chapitre. Ils exploitent tous les avantages de la technologie XML à différents niveaux. Cependant, ces travaux sont basés sur des hypothèses implicites, et cherchent souvent à répondre à des problèmes différents (e.g. gestion de configuration, monitoring, aide au management, intégration de système de gestion ...).

L'objectif de cet état de l'art était (1) de couvrir un large spectre de fonctionnalités, (2) de donner une idée intuitive des différents problèmes abordés, (3) d'identifier et de peser les avantages liés à l'utilisation d'XML en matière de management de réseau et enfin (4) d'identifier les lacunes inhérentes à chacune des solutions vis à vis des besoins actuels. D'autres travaux de recherches ont été étudiés et seront cités dans le reste du document, et ce, dans un cadre de comparaison plus précis.

Nous pouvons néanmoins d'ores et déjà, élaborer une première base de comparaison informelle. L'intérêt principal de cette comparaison est qu'elle fait apparaître des points clés lors de l'étude et la caractérisation de tels systèmes. Il est à noter que nous rappelons l'architecture SNMP, la plus ancienne et la plus répandue, en guise de référence.

Architectures	Modèle de communication	Protocole de transport	Encodage de transport	Modèle de données	Nommage
SNMP	C/S 1NMS + agents	UDP	ASN.1	MIB (SMI)	OID
SNMP v2/v3	idem + inform man-man				
WBEM	C/S	HTTP	XML	CIM Object Repository	Espace de noms +liste attributs
NetConf (Enns)	C/S	Protocoles orientés cnx	Xml-Rpc		Chemin complet des entités parentes
NetConf (XCLI)	C/S	Telnet			
WIMA	C/S	HTTP	MIME	MIB ou CIM	
Postech	C/S	HTTP	XML	XML (DOM)	1 fct par URL
JUNOScript	C/S	SSH,SSL, Telnet,Serie	Rpc light Soap, Beep	JUNOS	Chemin complet des entités parentes
SyncML DM	C/S	Http(s),sms OBEX/WAP	XML	XML	Chemin complet des entités parentes
Configuration Registrar	C/S	HTTPS	XML	Repository Text	LDAP

FIG. 1.11 – Récapitulatif technologique des approches présentées

Architectures	Richesse du protocole	Transaction	Notification sûre	Autocfg. plan mgmt	Gestion des erreurs	Sécurité	Contrôle d'accès
SNMP	Opérations de base Get,GetNext Set	Non	Non	Non	Simple Code erreur entier	Auth. triviale commu-nautés	Droits accès par var.
SNMP v2/v3	GetBulk Invoke Create/Del tables					Crypt. Alarme sécurité	
WBEM	15 opérations	Non	Non	Non		Externe HTTPs	Ctrl accès
NetConf (Enns)	Get-conf Edit-conf Copy-conf Delete-conf +extensions	Oui,partiel canal mgmt des RPCs Candidate Bloquage	Oui canal notif. RPC	Non	Réponse en RPC Détails Ok si pas err	Ext.	Ext.
NetConf (XCLI)	Mapping commandes natives CLI	Non		Non	Oui Result cmdes	Ext.	Ext.
WIMA	Get Set		Oui	Non			
Postech	Get Set		Oui Alert. async.	Non		Ext.	Ext.
JUNOScript	Load,Get, Sup config	Oui Config Candidate Rollback DAct cfg	Non	Non	Réponse en RPC Détails Ok si pas err	Ext.	Ext.
SyncML DM	Put,Get Add,Del Copy	Oui atomic	Oui Alert. RPC	O/N		Ext.	Ext.
Cisco Configuration Registrar	commandes natives CLI	Non	Oui CNS Event Service	O/N		Ext.	Ext.

FIG. 1.12 – Récapitulatif fonctionnel des approches présentées

Voici quelques constatations :
- Tous les modèles étudiés sont de type Client/Serveur.
- Toutes les architectures de management se basent sur un protocole de transport largement déployé.
- Certaines architectures comme Postech et SyncML DM utilisent XML à la fois pour la représentation des données et pour le transport, alors que d'autres comme WIMA ne l'utilisent que pour le transport.
- Les protocoles de management utilisés sont plus ou moins riches, depuis les commandes de base unitaires de SNMP, aux commandes de haut niveau (commandes de blocs, gestion des transaction, ...) de JUNOScript et NetConf (Enns).
- Certaines approches récentes comme SyncML DM et Cisco CR permettent d'envisager de l'autoconfiguration du plan de gestion.
- Mise à part SNMP et WBEM, toutes les architectures relèguent les aspects sécuritaire au protocole de transport sous-jacent.

De façon générale, nous constatons qu'XML est intégré petit à petit au sein des nouvelles approches de management.

L'utilisation d'XML pour la représentation des données et le transport représente un réel avantage dans la mesure où l'on bénéficie à la fois de ses qualités d'intégration des données et d'interopérabilité entre les systèmes. Dans le même ordre d'idées, une approche basée sur des commandes de haut niveau permet non seulement de simplifier les opérations de management mais aussi de mettre en place des systèmes transactionnels.

L'inconvénient majeur est que toutes les approches présentées sont de type Manager/Agent. Plus précisément, elles mettent toutes en oeuvre un modèle Client/Serveur dans lequel le manager joue le rôle de client et les agents le rôle de serveurs. Les problèmes de passage à l'échelle de ce type d'architecture sont bien connus (surcharge du serveur lorsqu'il y a beaucoup de clients). De plus cette architecture ne favorise pas l'autonomie du système de management. Force est de constater qu'XML n'a été utilisé que dans le but d'améliorer d'anciens concepts et non pas pour en favoriser de nouveaux.

Pour éviter les inconvénients décrits ci-dessus et pour bénéficier des avantages énoncés, nous proposons une approche différente dans le chapitre suivant.

Chapitre 2

Active XML et les Web Services

2.1 Les Web Services

Les services Web sont des applications modulaires, autodescriptives, publiables, localisées et invoquées via des protocoles standard du Web. Un service Web est décrit dans un document WSDL (Web Service Description Language) [13], précisant les méthodes pouvant être invoquées, leur signature, et les points d'accès du service (URL, port, etc.). Ces méthodes sont accessibles via SOAP [11] : la requête et la réponse sont des messages XML transportés par HTTP. Un service Web est accessible depuis n'importe quelle plate-forme ou langage de programmation. On peut utiliser un service Web pour exporter des fonctionnalités d'une application et les rendre accessibles via des protocoles standards. Le service Web sert alors d'interface d'accès à l'application, et dialogue avec elle au moyen de middleware (Corba, RMI, DCOM, EJB, etc.).

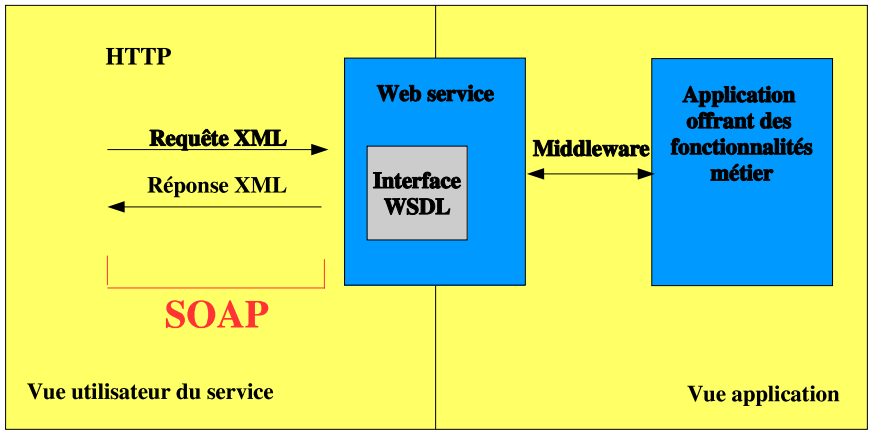

FIG. 2.1 – Schéma de fonctionnement d'un Web Service

Les services Web sont aujourd'hui associés à trois spécifications XML :
- SOAP : pour le transport des données et l'infrastructure de communication ;
- WSDL : pour la description des services offerts ;
- UDDI : annuaire permettant le référencement des services par les fournisseurs et leur découverte par les utilisateurs.

2.1.1 SOAP

SOAP (Simple Object Access Protocol) est développé sous les auspices du W3C. Il est défini comme un protocole léger permettant d'échanger des informations structurées et typées dans un environnement distribué. S'appuyant sur XML, SOAP propose un mécanisme simple de représentation des différents aspects d'un message entre applications. N'imposant aucun modèle de programmation spécifique, SOAP peut donc être utilisé dans tous les styles de communication : synchrone ou asynchrone, point à point ou multipoint, Intranet ou Internet.

La norme SOAP est composée de quatre parties :
- La structure de l'enveloppe SOAP qui définit une structure générale décrivant le contenu, le destinataire, et la nature du message.
- Les règles d'encodage SOAP qui définissent le mécanisme de sérialisation utilisé pour échanger des objets.
- SOAP RPC qui définit, pour les utilisations synchrones, une convention de représentation des appels et des réponses des procédures distantes.
- La définition de l'utilisation de HTTP comme couche de transport des messages SOAP.

Les règles de codage utilisent abondamment XML Schema et la notion d'espace de nom pour décrire la structure des données constitutives des messages SOAP.

2.1.2 WSDL

WSDL (Web Services Description Language) est, comme son nom l'indique, un langage de description de Web Services, au format XML. Il est l'équivalent de IDL (Interface Definition Language) [5] pour la programmation distribuée (CORBA [2], DCOM [12]). Il permet de séparer la description des fonctionnalités abstraites offertes par un service des détails concrets d'une description de service, tels que "comment" et "où" cette fonctionnalité est proposée. En clair, il définit, de manière abstraite et indépendante du langage, l'ensemble des opérations et des messages qui peuvent être transmis vers et depuis un service web donné.

Le WSDL décrit quatre ensembles de données importants :
- informations d'interface décrivant toutes les fonctions disponibles publiquement,
- informations de type de données pour toutes les requêtes de message et requêtes de réponse,
- informations de liaison sur le protocole de transport utilisé,
- informations d'adresse pour localiser le service spécifié.

2.1.3 UDDI

UDDI (Universal Description, Discovery and Integration) a été créé par IBM, Microsoft et Ariba. C'est une architecture répartie qui permet aux fournisseurs de services Web (Business providers) d'enregistrer leurs services, et aux applications de rechercher les services correspondant à leurs besoins, de façon normalisée. UDDI est donc un annuaire distribué de services Web et d'entreprises (Business/Service Registry). UDDI se comporte lui-même comme un service Web dont les méthodes sont appelées via le protocole SOAP.

2.2 Active XML

Active XML est une approche qui exploite les services web pour réaliser de l'intégration de données au sein d'une architecture pair-à-pair (Cf. figure 2.2). Typiquement, l'intégration de données consiste à rechercher et à combiner des données provenant de différentes sources en respectant un schéma prédéfini. Ces données peuvent ensuite servir de source à un autre schéma d'intégration et ainsi de suite de façon hierarchique.
De façon à offrir une plus grande flexibilité et un meilleur passage à l'échelle, Active XML élimine cette contrainte hierarchique : chaque peer joue un rôle symétrique, à la fois fournisseur et intégrateur de données.
De plus, Active XML apporte une solution au problème de l'hétérogénéité des sources de données externes grâce à l'utilisation du modèle Web Service, aussi bien pour la communication inter-peers que la communication entre un peer et une source de données externe.

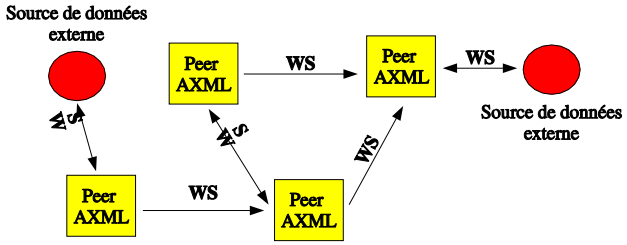

FIG. 2.2 – Schéma général de présentation d'Active XML

Active XML est centré autour de documents AXML qui correspondent à des documents XML contenant des appels à des Web Services. Le langage permet à la fois de spécifier de tels documents et de définir grâce à XQuery de nouveaux Web services basés sur ces derniers.

Les caractéristiques de ce framework sont :
- la résistance au passage à l'échelle
- le contrôle du moment d'activation des appels de services
- le contrôle de la durée de vie des données recueillies
- l'échange de données explicites ou implicites (par références)

2.2.1 Le peer AXML

Un peer AXML (Cf. figure 2.3) est composé à la fois d'un repository de documents AXML et d'un repository de documents de définition des web services qu'il expose. Les web services exposés correspondent à l'éxécution d'une requête XQuery [7] sur un document AXML du peer. Nous allons détailler les notions de documents AXML et de document de définition de web services dans la suite de cette partie.

2.2.2 Le document AXML

Un document AXML peut être vu comme une matrice partiellement matérialisée intégrant du code XML statique ainsi que des données dynamiques obtenues par un appel à un Web service. De ce fait, si la source de données externes change, la partie dynamique du document est changée.

Cet appel de service implicite est représenté par l'introduction d'une paire de balise <sc> au sein du document AXML comme présenté dans l'exemple ci-dessous :

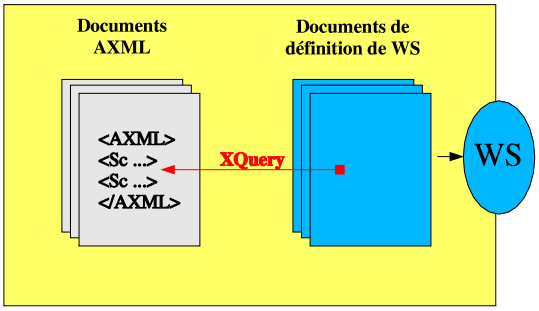

FIG. 2.3 – Schéma d'un peer Active XML

```
<myAuctions>Auctions I'm interested in
  <category name= « Toys »>
    <sc>auction.com/getOffers(« toys »)</sc>
  </category>
</myAuctions>
```

Le résultat de cet appel produit l'intégration des données réceptionnées au sein du document AXML comme ci-dessous :

```
<myAuctions>Auctions I'm interested in
  <category name= « Toys »>
    <sc>auction.com/getOffers(« toys »)</sc>
    <auction aID= « 1 »><description> Stuffed bear toy</description></auction>
    <auction aID= « 2 »><description> ? </description></auction>
    <sc>auction.com/getOffers(« importedtoys »)</sc>
  </category>
</myAuctions>
```

Comme on peut le constater dans l'exemple ci-dessus, la réponse à un appel de service peut lui-même en comporter un autre.

Activation des appels

L'activation des appels de services est contrôlée par deux attributs additionnels de la balise "sc" : mode et frequency. On distingue trois types d'appel :

- Les appels explicites (mode « pull », immediate).
  L'appel est activé ou réactivé dans deux cas :
  - lorsque la durée spécifiée par l'attribut « frequency » (1 jour, 1 semaine,...) est écoulée
  - lorsque d'autres appels de services sont effectués
- Les appels implicites (en mode « pull », lazy).
  L'appel est activé lorsqu'il a expiré et que les données sont demandées
- Les appels en mode « push ».
  Basés sur le mécanisme de souscription (le serveur web envoie les données au client), l'appel est activé lors de la synchronisation avec une source de données externes.

Il est à noter que selon la façon d'effectuer l'appel, on peut contrôler quel peer effectue le travail.

Fusion des résultats des appels de services

Les résultats contenant des éléments avec des identifiants sont fusionnés de façon à ce que l'identifiant reste unique.

Contrôle de la durée de vie des données

Les résultats des appels étant stockés dans le document appelant, il est nécessaire de pouvoir remplacer les anciens résultats par les nouveaux et de ne pas récupérer les données obsolètes pour éviter une accumulation. Un attribut « ExpiresOn(date) » peut être attaché à tous les noeuds de données dans un document AXML pour spécifier une date à partir de laquelle les données ne sont plus pertinentes : dans ce cas les données sont considérées comme effacée du document. De plus, dans le cas où il n'est pas précisé, un noeud hérite la valeur de son ancêtre.

```
Ex: <category name= ‘‘toys’’ expiresOn= ‘‘Feb. 19th,2002’’>
```

Dans cet exemple, lorsque la date d'expiration est dépassée, la catégorie "toys" ne sera pas retournée en réponse aux requêtes des autres peers.

Par ailleurs, un attribut « Valid(rt : time of answer) » permet de spécifier la durée pendant laquelle les données résultantes d'un appel de service sont conservées.

```
Ex: <sc valid=’’rt+1year’’>auction.com/GetOffers(‘‘A’’)</sc>
```

Dans cet exemple, l'appel de service spécifie que les enchères de catégorie "A" seront archivées pendant une année.

2.2.3 Définition d'un service AXML

Le framework AXML permet d'effectuer des appels à n'importe quel Web Service, mais permet également d'en définir des nouveaux. Un service AXML est défini par une requête XQuery paramétrée relative à un document AXML. Notons qu'Active XML prévoit d'utiliser une extension de la norme XQuery pour permettre non seulement la création de requêtes de consultation mais aussi celles de modification.

Exemple :

```
let sc auction.com/getOffers() be
  for $cat in document(‘‘auction.com/a.xml’’)//category,
      $a in $cat/auction,$aID in $a/@aID/text(),$des in $a/description/text()
  return <auction aID={$aID}>
           <description>{$des}</description>
         </auction>
```

Dans cet exemple, noux définissons un web service exposant une fonction getOffets ne prenant aucun paramètre. Cette fonction retourne, pour chaque catégorie d'enchère décrite dans le document AXML a.xml, une description de celles-ci sous la forme suivante :

```
<auction aID=’1’><description>Stuffed bear toy</description></auction>
<auction aID=’2’><description>...</description></auction>\\
```

De surcroît, AXML permet de spécifier des Web services "continus".

Service continu

Il s'agit de fournir un flux continu et infini pour un seul appel. Cela permet de représenter des « capteurs ».

2.3 Synthèse

Nous avons présenté succinctement dans ce chapitre les modèles Web Services et Active XML. Les Web Services correspondent à des applications modulaires, autodescriptives, publiables et invoquées via des protocoles standard du Web. Trois spécifications XML y sont associées : SOAP pour le transport des données, WSDL pour la description des services offerts et UDDI pour le référencement des services et leur découverte.

Active XML est, quant à elle, une approche qui exploite les Web Services pour réaliser de l'intégration de données au sein d'une architecture pair-à-pair. Elle permet de spécifier des documents contenant des appels à des Web Services et de définir de nouveaux Web Services basés sur ces derniers. De plus, cette approche autorise le contrôle du moment d'activation des appels de services et la durée de vie des données recueillies. Enfin, elle présente une bonne résistance au passage à l'échelle.

Ces deux approches sont à la base de la nouvelle architecture de management que nous proposons dans le chapitre suivant.

Chapitre 3

Contributions

3.1 Une nouvelle approche

3.1.1 Motivations

Les modèles classiques de supervision n'ont pas été conçus dans le but d'être orientés données : il en résulte qu'ils sont relativement inadaptés aux *besoins grandissant d'intégration et d'interopérabilité* en matière de supervision. En effet, les récentes orientations de la recherche tendent à montrer ces besoins : WBEM consiste à fournir un chapeau unificateur à l'ensemble des acteurs de management afin de garantir une plus grande interopérabilité. On développe par ailleurs des architectures de supervision à base d'XML (JunoScript, XMLConf, SyncML ...) et des passerelles XML/SNMP pour la facilité d'intégration qu'offre cette technologie.
D'un autre côté, les Web Services se présentent comme une solution aux problèmes d'interopérabilité des applications ; ils commencent à être envisagés dans le domaine de la supervision de réseau comme le montre les groupes de recherche OASIS[1] et le NMRG[2] de l'IRTF.
Active XML, qui est une technologie basée sur les Web Services présente un autre avantage : elle a été conçue pour répondre aux besoins d'intégration de données.
Aussi nous proposons une nouvelle approche pour le management de réseau basée sur cette technologie pour répondre à ces deux besoins.

Par ailleurs, jusqu'alors, le plan de management a été élaboré de manière *statique*. En effet, dans le modèle classique de type manager/agent chaque entité est fortement assignée à un rôle prédéfini, d'où *absence d'autonomie* et *un modèle de communication figé* ; de la même façon, *les informations de management échangées entre ces entitées sont figées* (ex MIB) dont le contenu est contraint par les modèles partagés (SMI). De plus, le plan de management a également été élaboré de manière *centralisée*. Ce type d'architecture ne permet pas le passage à l'échelle.
Il faut donc ouvrir le plan de management de façon à lui offrir *une plus grande dynamicité*, *une plus grande autonomie*, *une plus grande souplesse d'utilisation* et *un meilleur passage à l'échelle.*

Enfin, jusqu'à présent les systèmes n'étaient pas autoconfigurables. Certaines approches récentes comme Cisco CR et SyncML DM tendent à favoriser l'autoconfiguration qui apparaît aujourd'hui comme un vrai besoin.
Notre approche doit tenir compte de ce *besoin grandissant d'autoconfiguration.*

[1] `http://www.oasis-open.org/committees/mgmtprotocol/`
[2] `http://www.irtf.org/charters/management.html`

3.1.2 Le modèle

Les modèles actuels sont basés sur des architectures centralisées de type manager/agent 3.1, présentant plus ou moins de délégation : il est en effet possible de télécharger des programmes de management sur les agents afin d'y intégrer une partie des opérations de management et de réduire ainsi la charge du manager.

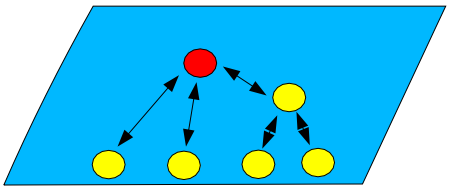

FIG. 3.1 – Architecture Manager/Sub-Manager/Agents

Á l'inverse des architectures classiques, nous proposons un modèle complètement décentralisé P2P 3.2, basé sur Active XML.

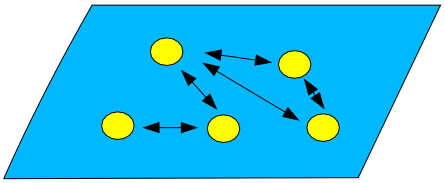

FIG. 3.2 – Architecture décentralisée (P2P)

Dans ce cas, chaque entité du plan de management correspond à un peer AXML autonome : leurs caractéristiques et leurs rôles en matière de management de réseau sont communs. De part cette généricité du modèle, le système n'est plus du tout figé, ce qui lui procure une plus grande souplesse d'utilisation et une plus grande dynamicité.

L'objectif de la mise en place d'un tel modèle est de permettre l'abstraction complète des données managées par le système et de la politique de management : le plan de management est ainsi complètement indépendant, ce qui lui permet d'être utilisé conjointement avec un maximum de périphériques et de politiques de managements.

D'une manière générale, chaque peer AXML contient un ensemble de documents de management (AXML) lui permettant d'échanger des informations grâce à des appels à des Web Services déployés par les autres peers. Certains peers AXML correspondent à un périphérique à manager. Le manager a pour charge de fournir au système l'ensemble des documents de management et leurs Web Services associés en fonction de la tâche de management qu'il souhaite opérer.

3.1.3 Aspect dynamique

La dynamique du plan de management modélise la capacité du système de management à évoluer pour répondre à de nouveaux besoins. Nous envisageons deux modèles de plan de management permettant d'assurer cette dynamicité.

– soit chaque entité intègre un document de management spécifique fournit par le manager en guise de plan de travail : dans ce cas, la dynamicité du modèle se situe dans le mécanisme de provision des entités.

– soit chaque entité intègre un document de management commun inclus d'origine : dans ce cas, la dynamicité du modèle se révèle dans la généricité de ce document.

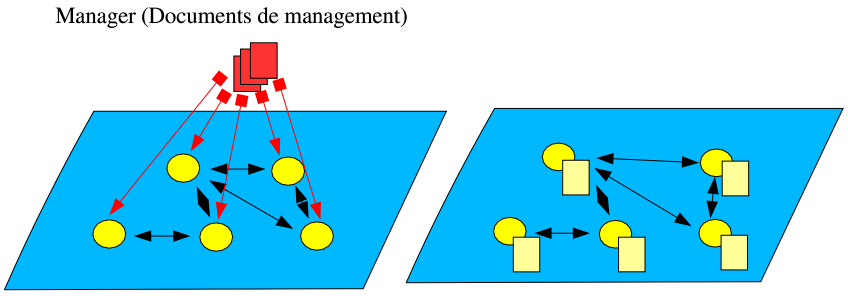

FIG. 3.3 – Documents de management spécifiques (gauche) et documents de management communs (droite)

Bien que l'on puisse considérer la deuxième possibilité comme un cas particulier de la première, nous verrons qu'il s'agit bien de deux approches différentes pouvant devenir complémentaires dans certains cas.

En fait, les deux approches diffèrent sur le plan de la provision des peers en documents de management. Nous allons détailler ces deux approches et énumérer leurs avantages et inconvénients respectifs.

Document spécifique

Dans cette approche (Cf. Fig 3.4), le manager se charge d'amorcer le système de management en configurant chaque peer à l'aide d'un document de supervision et d'un document de description des Web Services associés. Ensuite, chaque peer peut fonctionner de manière autonome : il suit les consignes décrites dans le document de management et se charge ainsi d'échanger des données avec ses frères et avec des Services Web externes.

Au départ, les peers ne contiennent aucun document de management.

Cette approche permet de procurer au plan de management une très grande dynamicité dans la mesure où le manager n'a aucune contrainte : il peut fournir aux peers autant de fichiers de management qu'il le souhaite. Dans ce cas, les peers peuvent participer à plusieurs tâches de management en même temps. De la même façon, le manager peut fournir à chaque peer un document de description des WS associés à ses documents de management. Ainsi il peut non seulement définir quelles informations le peer va partager, mais aussi lui dicter des traitements à effectuer (ex : filtre sur les informations à partager, opérations de maintenance des documents...).

Cependant cette approche ne possède pas que des avantages : en effet, le mécanisme de provision des peers n'est pas trivial. Il faut en effet pouvoir déployer "à chaud" des documents de

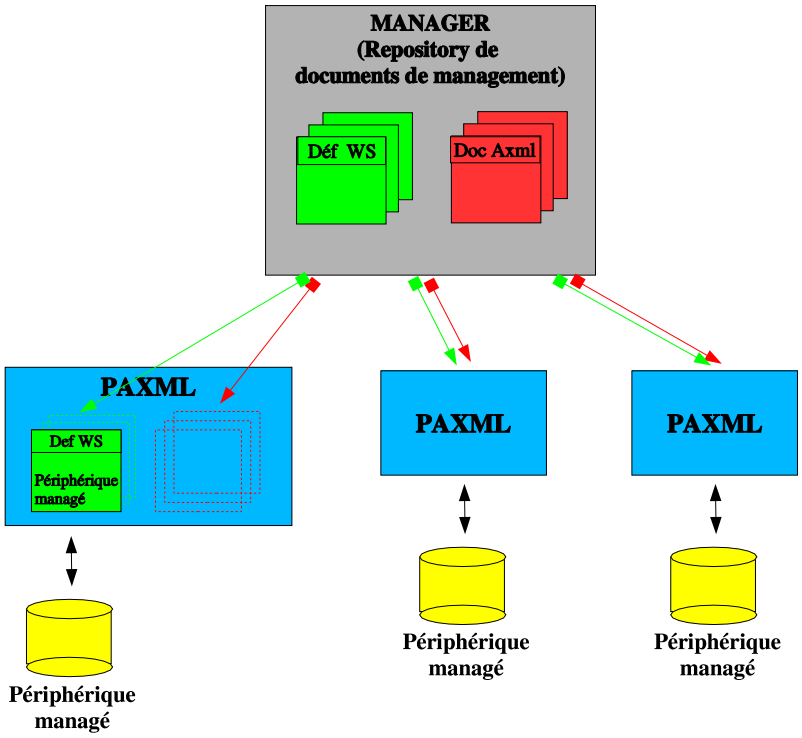

FIG. 3.4 – Distribution de documents de management spécifiques

management dans les peers AXML. Si conceptuellement ce type de mécanisme est envisageable, il reste relativement difficile à implémenter.

Documents communs génériques

Dans cette approche (Cf. Fig 3.5), chaque peer AXML possède à son lancement un document de management générique ainsi qu'un document de définition des web services associés, lui aussi générique.

Chaque peer doit offrir deux fonctionnalités primaires :
- récupération auprès du manager de sa marche à suivre,
- permettre aux autres peers d'accéder à ses données.

Chacune de ces deux fonctionnalités est traitée par l'un des deux documents génériques du peer. Ainsi, nous proposons les deux documents génériques suivants :

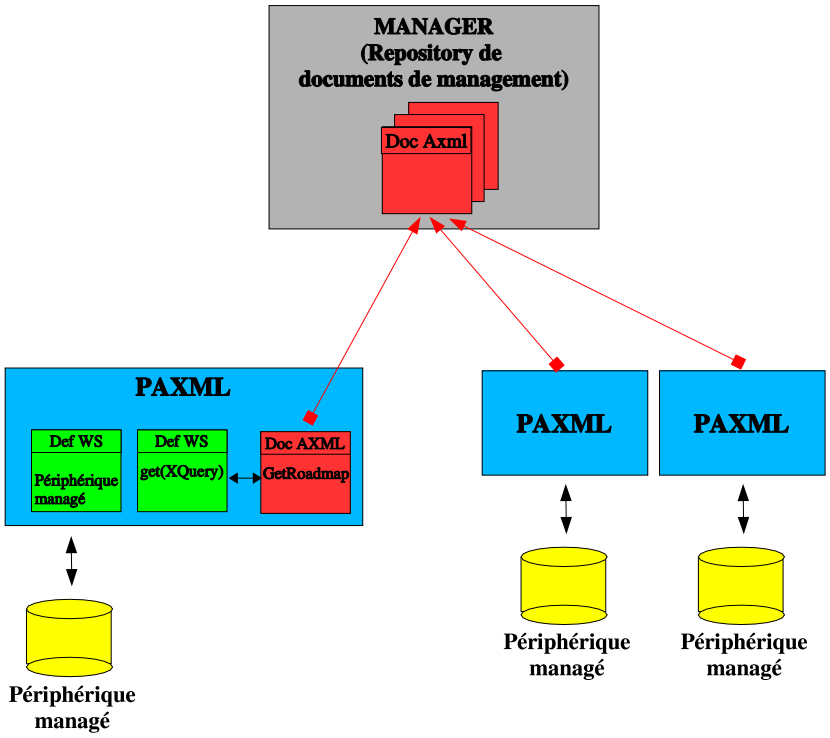

FIG. 3.5 – Documents de management génériques

Document de management générique :

```
<?xml version='1.0' encoding='UTF-8'?>
<ManagementDocument xmlns:axml='http://www-rocq.inria.fr/verso/AXML'>
  <axml:sc methodName='getRoadMap' serviceNameSpace='urn:axmlManager'
   serviceURL='...' valid='last' frequency='0' mode='immediate'>
    <axml:params>
      <axml:param name='peerID'>10</axml:param>
    </axml:params>
  </axml:sc>
</ManagementDocument>
```

Avec un tel modèle, les peers vont automatiquement rechercher auprès du manager leur marche à suivre ("roadmap") lorsqu'ils sont activés. Bien sûr, il faut au préalable configurer l'adresse du manager au sein du document générique.

Document de définition de WS générique :

```
let sc get($xquery) be
  $xquery
```

De plus, ce document peut contenir la définition d'autres web services permettant par exemple de rafraîchir la roadmap d'un peer, de démarrer ou arrêter un peer (suppression de sa roadmap).

D'autre part, on peut imaginer effectuer du routage applicatif dynamique pour provisionner les peer avec leur "roadmap". Dans ce cas, le document de management est vide au départ. Il faut obligatoirement faire appel à un service "refresh" du peer qui se chargera de compléter le document de management avec l'appel au service "roadmap" comme il a été décrit précédemment. Cette petite nuance permet d'envisager un enchaînement dans la provision des peers. Ainsi le document de management fourni par le manager au premier peer sur lequel on exécute la fonction refresh contient en premier un appel aux fonctions refresh des peers avec lesquelles il va communiquer. On obtient ainsi un mécanisme de provision qui s'enchaîne éliminant ainsi les incohérences pouvant survenir dans le cas où les peers démarrent tandis que tous les peers impliqués dans la politique de management n'ont pas encore été configurés.

Comparatif des deux approches

Les deux approches précédentes présentent chacune des avantages et des inconvénients.

Documents spécifiques	
Avantages	Inconvénients
Souplesse totale	Provision de tous les peers en documents de définition de web services

Documents communs	
Avantages	Inconvénients
Pas de problème de provision	1 seul document de management transmission de la requête XQuery à chaque get

Au vu de ces avantages et inconvénients, il convient de dire qu'aucune de ces deux architectures est en elle-même adapatée à tous les cas. En effet, la première approche apporte une souplesse totale quant à la programmation d'une politique de management mais souffre du coup d'un excès de configuration. En effet, il est nécessaire de fournir à tous les peers à la fois un document AXML (roadmap) ainsi qu'un document de définition de web services. À l'inverse la seconde approche évite cet excès en transmettant uniquement la roadmap, mais limite la souplesse (1 seul document de management par peer).

Cependant, nous pouvons tirer le meilleur de ces deux approches en les combinants de la façon suivante : la deuxième approche doit permettre de construire des modules de management génériques fréquemment utilisée (ex : monitorage d'un périphérique SNMP), tandis que la première approche doit être utilisée pour la configuration des peers servant à superviser d'autres peers. De cette façon, les inconvénients de chaques approches sont limités. La seconde approche permet alors d'optimiser le traitement des fonctions de management de base, tandis que la première approche préserve la très grande souplesse de l'ensemble du plan de management.

3.1.4 Intégration du modèle de management avec les périphériques managés

Comment coupler les Peers Active XML du plan de management aux périphériques du plan d'action ?

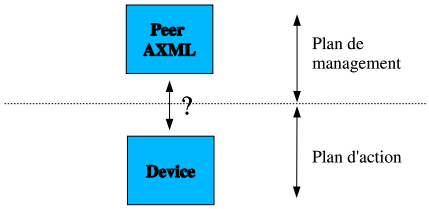

FIG. 3.6 – Intégration Active XML / Périphérique managé

Nous envisageons trois possibilités permettant cette interopérabilité :

- Solution à base de Web Services :
  Chaque périphérique dispose d'une interface de management réalisée avec des Web Services. Dans ce cas, l'intégration est très simple puisque les fonctions de management sont directement accessibles par un Peer AXML de management.

- Solution à base de passerelles :
  Beaucoup de périphériques existant ne disposent pas d'une interface de management interopérables avec l'architecture que nous proposons à base de Web Services. Une solution permettant de résoudre ce problème est d'intercaler entre un Peer AXML (Web Services) et le périphérique managé une passerelle (ex : passerelle SNMP/SOAP).
  La réalisation de ce genre de passerelles est d'autant plus aisée que certain périphériques disposent déjà de protocoles de management proches des Web Services : par exemple, JUNOScript étant basé sur un mécanisme proche de XML-RPC, une passerelle vers les Web Services peut être réalisée facilement. De plus, des travaux de recherches ont d'ores et déjà été effectués dans ce domaine (cf Etat de l'art : Intégration XML/SNMP).

- Intégration directe :
  Il est possible d'intégrer directement le Peer AXML dans le périphérique. Dans ce cas, les données de management du périphérique sont directement accessibles par le Peer AXML sous forme XML.

Au vu de ces différentes solutions, il est clair que l'intégration de l'approche que nous proposons avec les périphériques du plan d'action est possible autant avec de nouveaux périphériques spécialement conçus à cet effet, qu'avec les périphériques existants.

Remarque : Nous avons réalisé dans ce cadre un prototype simple de passerelle SNMP/Web Service permettant de faire du monitorage (cf. Annexe E). Il est à noter que seules les opérations de base de SNMP ont été traduites, les notifications n'étant pas reportées.

3.1.5 Synthèse

Dans ce chapitre nous avons proposé un plan de management dans lequel les entités sont entièrement autonomes et dont le traitement peut être reconfiguré à volonté. L'avantage d'un tel modèle est sa plus grande souplesse qui permet de spécialiser simplement le plan de management en fonction des nouveaux besoins en un temps très court.

Les progrès actuels des technologies d'intégration telles que les Web Services et plus particulièrement Active XML permettent d'envisager la mise en œuvre de solutions de management importantes et complexes. En outre, un tel modèle permet de tester la validité et de superviser rapidement de nouveaux périphériques logiciels ou matériels : ce modèle offre ainsi plus de possibilités d'évaluation que celles offertes par une supervision classique en permettant le test dans un contexte réel.

Le deuxième avantage de ce concept est que les entités peuvent communiquer entre-elles et collaborer en formant des chaînes de management, ce qui permet de paralléliser le traitement d'opérations de supervision correspondant à des chemins critiques.

Le troisième avantage est que ce modèle ouvre la voie à des mécanismes de management autonome, notamment au niveau de la découverte de services (UDDI pour les Web Services).

Enfin, le modèle étant entièrement décentralisé, il permet la supervision d'un nombre plus important d'entités : son passage à l'échelle est ainsi assuré.

Chapitre 4

Prototype

Dans ce chapitre nous présentons le prototype fonctionnel que nous avons réalisé afin de valider l'architecture que nous proposons. Nous avons choisi de développer notre architecture sur un exemple concret appartenant au domaine de la gestion de configuration, à savoir la "réplication de règles de firewall".

4.1 Présentation du prototype

Nous considérons dans ce chapitre l'exemple suivant :

Un réseau local possède trois points d'entrée de l'extérieur. Chacun de ces points d'entrée est protégé par un Firewall. Chaque Firewall possède un certain nombre de règles de filtrage des paquets en transit. Sur le plan du management, on souhaite harmoniser les règles entre ces trois points d'entrée : lorsque le manager change une règle sur l'un d'entre eux, la modification est répercutée automatiquement sur les deux autres.

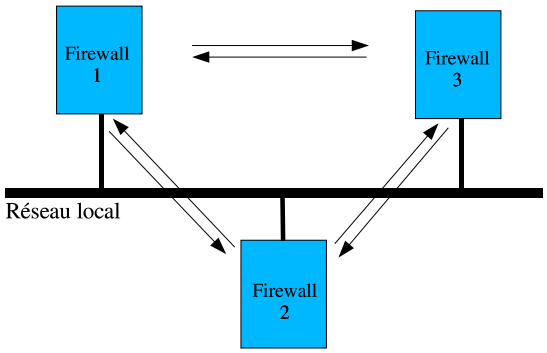

FIG. 4.1 – Example de réplication de politique de Firewall

Dans cet exemple, chaque Firewall est représenté par un Peer Active XML sur le plan du management. Le manager fournit à chacun de ces peers un document de management lui indiquant avec qui et comment échanger ses règles de filtrage. La politique d'approvisionnement choisie ici est la première proposée dans notre architecture. Ce choix s'explique par le fait que

l'implémentation d'Active XML dont nous disposons est encore un prototype est la mise à jour d'information sur un Peer n'est pas encore opérationnelle (cf. XQuery Update).

4.2 Implémentation

Le prototype que nous avons réalisé fonctionne sous Linux. Il a été écrit en Java et utilise la plateforme de Web Services Axis [3]. Par ailleurs, l'intégration des règles de firewalling est réalisée grâce à une librairie externe : DaxFi [4].

4.2.1 Axis

Apache AXIS est une implémentation open-source sur la plate-forme JAVA des spécifications de la norme SOAP 1.1, et d'une partie des fonctionnalités de la nouvelle version de la norme SOAP 1.2. AXIS est développé par la fondation Apache (Apache Software Foundation).

Apache AXIS est à la fois un environnement d'hébergement de Web Services, et un toolkit complet de développement pour la création ainsi que l'accès client à des services tiers. Ce toolkit comporte des fonctionnalités avancées de génération de code automatique basées sur l'analyse de descriptions WSDL de Web Services et le mapping automatique à double sens entre classes Java et description WSDL de services.

4.2.2 DaxFi

Notre prototype se base sur la librairie DaxFi (Dynamic XML Firewall), qui constitue un outil simple de configuration des firewalls les plus courants à partir de règles exprimées en XML. Les firewalls supportés par cette librairie sont les suivants :

- iptables
- ipfilter
- ipchain
- ipfwadm

Voici un exemple de règle de firewall énoncée avec la syntaxe DaxFi : cette dernière accepte et journalise tous les paquets TCP provenant du sous réseau "192.196.1.0/24", arrivant sur l'interface "le1" et dirigés vers le port 80.

```
<?xml version='1.0'?>
<append>
  <rule direction='in' source-ip='192.196.1.0/24' interface='le1'>
    <tcp destination-port='80' />
    <accept />
    <log />
  </rule>
</append>
```

[3] http://ws.apache.org/axis/

[4] http://daxfi.sourceforge.net/

4.2.3 Schéma général de fonctionnement

Le prototype fonctionne de la manière suivante : les Peers AXML représentant chacun un firewall possèdent au départ un document de management (Document AXML, cf. Annexe C), ainsi qu'un document de définition des Web Services qu'ils exposent (Document Def. WS, cf. Annexe D).

Le document de management a deux rôles :
- il impose la politique d'approvisionnement en règles du firewall auprès de ses voisins grâce à des appels à leurs Web Services.
- il met à jour le Firewall qu'il manage lorsque les règles ont été échangées grâce à l'appel du Web Service correspondant de ce dernier.

Le document de définition des Web Services exposés comprend la définition d'un Web Service permettant de récupérer les règles de firewalling qu'il connaît.

Le principe général est représenté sur la figure 4.2. Dans cet exemple, si le manager modifie une règle sur le firewall 3, elle sera automatiquement répercutée successivement sur le firewall 2 puis sur le firewall 1 : en effet, d'après la politique d'échange que nous avons définie, les peers AXML s'appellent entre eux périodiquement pour s'échanger leurs règles de firewalling respectives.
De la même façon, les Peers envoient périodiquement les règles qu'ils se sont échangés au firewall qu'ils managent grâce à des appels de Web Services.

Il est à noter que pour faciliter l'implémentation, nous utilisons un fichier temporaire intermédiaire en guise de middleware entre le Web Service DaxFi développé et la librairie DaxFi.

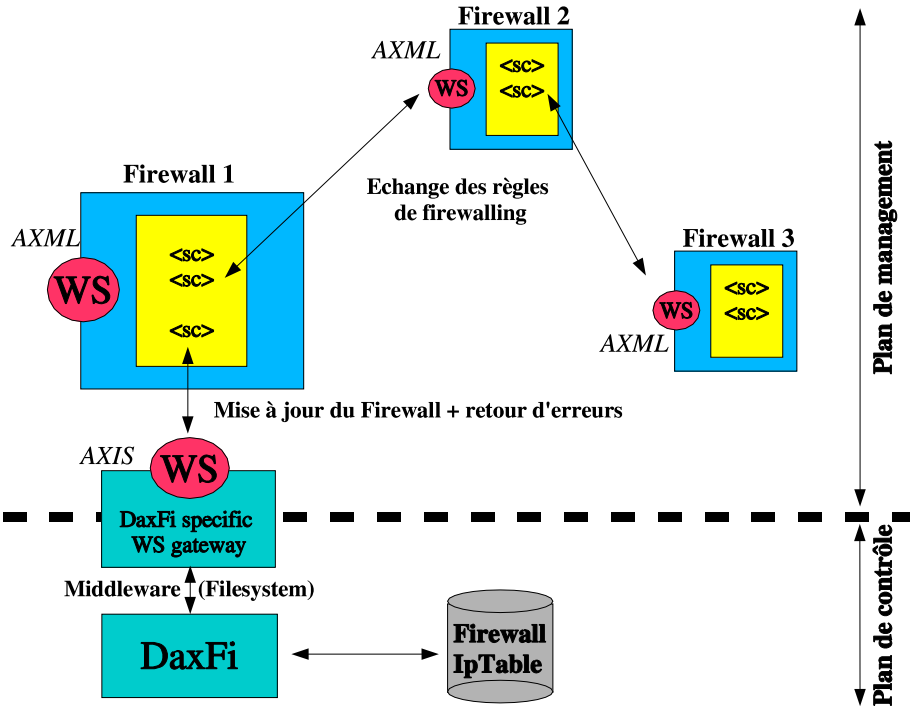

FIG. 4.2 – Schéma de fonctionnement du prototype

4.2.4 Problèmes identifiés

Le prototype nous a permis de déceler une faiblesse de notre approche, à savoir le besoin d'un mécanisme d'estampillage des messages échangés entre les peers. En effet, dans le modèle proposé par AXML, les règles nouvellement reçues sont fusionnées avec les anciennes règles ayant le même identifiant. Imaginons le scénario suivant : un administrateur modifie une règle sur le firewall 1 tandis qu'au même moment un autre administrateur modifie la même règle sur le firewall 2. Le problème est qu'il n'est pas possible de prévoir l'ordre d'arrivée des messages de règles sur chaque peer AXML du fait de leur transfert sur un réseau commun. Il n'est donc pas possible de contrôler la fusion entre les anciennes règles déjà connues et les règles nouvellement reçues, les nouvelles règles écrasant toujours les anciennes.
Bien sûr, il est possible d'envisager une solution qui consiste à fournir un identifiant unique à chaque règle (ex : numeroDuFirewall + NuméroRègle) , mais une telle solution élimine la fusion des règles identiques entre les firewalls.

D'autre part, le modèle ActiveXML ne permet d'effectuer que des appels de type asynchrone. Or dans bien des cas, il serait utile de pouvoir lancer un appel, non pas à un instant précis, mais en fonction d'un évènement précis, de façon synchrone. Typiquement, dans le cas du prototype présenté, il serait utile de pouvoir simplement mettre à jour le firewall dès la réception de toutes les règles.

4.2.5 Synthèse

Grâce à Active XML, le manager possède plusieurs moyens de contrôler la politique de management des firewalls : il peut par exemple contrôler le taux d'échange de données entre les firewalls en choisissant le firewall le plus approprié avec qui échanger des données, ou effectuer des traitements spécifiques au niveau des règles (ex : contrôle de la validité des règles provenants des autres Firewalls). De plus, il lui est possible de contrôler le dialogue des entités entre-elles : en effet, le manager peut, par exemple, soit demander au Firewall 1 de récupérer les règles du Firewall 2 et celles du Firewall 3, mais il peut aussi demander au Firewall 2 de renvoyer au Firewall 1, un lien vers les règles du Firewall 3 de façon à ce qu'il puisse les rechercher.

Il en résulte que le plan de management possède une grande souplesse et peut très facilement être modifié. De plus, l'ensemble des règles de firewalling étant écrites en XML, elles sont non seulement lisibles et facilement compréhensibles, mais peuvent aussi être très facilement utilisées et intégrées à d'autres outils (ex : Interface graphique de génération des règles...).

Par ailleurs, cet exemple nous a également permis d'identifier certaines faiblesses de notre approche : en effet, dans l'état actuel Active XML ne propose pas à la base un mécanisme d'estampillage des messages reçus qui permettrait de régir la fusion des informations reçues. Cependant, il va de soi que ce type de problème n'est pas nouveau et est bien connu dans le monde distribué ; des travaux futurs pourront donc palier à ce problème.

La grande capacité d'interopérabilité offerte par les Web services tend à faire grandir l'architecture de façon exponentielle. De plus, cette première implémentation se base elle-même sur un prototype de la technologie Active XML, encore en phase de développement. Aussi, certaines fonctionnalités offertes par Active XML telles que le service continu ou les requêtes de mise à jour n'ont pas pu être exploitées.

Conclusion

L'objectif de notre étude était de proposer de nouveaux modèles de supervision et de fournir une première étude sur la généralisation de l'approche Web services à toutes les fonctions de gestion.

Afin de répondre à cet objectif nous avons proposé un modèle de supervision novateur, basé sur Active XML, qui permet de bénéficier à la fois des avantages d'intégration de XML, d'interopérabilité des Web services, de dynamicité et de passage à l'échelle du modèle pair à pair. Le modèle proposé offre au plan du management des fonctions de supervision négociables et évolutives permettant de favoriser une plus grande autonomie des ressources managées et de maîtriser leur hétérogénéité. Au cours de la recherche, nous avons pu mettre en évidence que ce modèle facilitait, entre autre, la mise au point des différentes politiques de gestion dans un contexte réel.

Nous avons implanté un premier prototype fonctionnel dans le cadre de la gestion de configuration pour valider notre architecture. Mais nous nous sommes heurtés à des difficultés nouvelles : en choisissant Active XML, nous connaissions les problèmes inhérents à cette technologie (sécurité des Web services, terminaison des boucles, ...) ; de plus, du fait de son utilisation dans le contexte particulier de notre approche, nous avons pu mettre en évidence d'autres problèmes (estampillage, appel par évènements ...).

Si toutes les fonctions de gestion peuvent à priori bénéficier des avantages du modèle de management proposé, il n'en demeure pas moins que l'architecture n'a encore été évaluée dans tous les domaines, notamment celui du monitoring.

La durée impartie pour ce type d'étude n'a pas permis de répondre à tous les problèmes liés à l'ambition de ce sujet difficile. Aussi faudrait-il ouvrir de nouvelles pistes de recherche pour affiner, développer et optimiser le modèle en vue d'étendre son utilisation à toutes les fonctions de gestion, en particulier le monitorage pour lequel il est nécessaire de mettre en place un mécanisme d'alerte. A ce sujet, la fonctionnalité de service continu offerte par Active XML semble être une piste particulièrement prometteuse. D'autre part, dans le domaine de la gestion de configuration, les spécifications WSDL et UDDI liés Web Services semblent ouvrir la voie à une future extension de notre approche en matière d'auto-configuration. Enfin, des travaux pourront être menés pour répondre aux problèmes résiduels en matière de sécurité liée à l'utilisation des Web Services. Ces perspectives de recherche seront étudiées lors d'un futur stage de 3 mois dans l'équipe du Professeur Aiko Pras à l'Université de Twente aux Pays-Bas.

Glossaire

SNMP (Simple Network Management Protocol) : il s'agit du protocole de gestion de réseau. Dans SNMP, des agents qui peuvent être matériels ou logiciels, contrôlent l'activité de nombreux périphériques sur le réseau et envoient leur rapport au manager du réseau. Les informations de contrôle de chaque périphérique sont conservées dans une structure appelée base d'informations de gestion (Voir MIB).

MIB (Management Information Base) : il s'agit d'une structure de données normalisée par l'IETF contenant les informations de gestion des éléments d'un réseau. Toutes ces informations sont répertoriées dans une architecture hiérarchisée par un identifiant en notation pointée relatif à la branche de l'arbre (ex : 1.3.6.1.2.1.1.3.0 représente la donnée SysUpTime).

ASN.1 (Abstract Syntaxe Notation number One) : ASN.1 est une notation formelle qui permet de spécifier très facilement les informations manipulées par les protocoles de télécommunications, indépendamment des systèmes informatiques, des logiciels et des modes de transfert des données.

SMI (Structure of Management Information) : SMI est une structuration, définie à partir d'ASN.1 (Voir ASN.1) par l'ISO, des informations de gestion afin de les consolider et de les échanger dans un système d'administration commun.

RPC (Remote Procedure Call) : il s'agit d'un mécanisme d'appel de procédure à distance.

XML-RPC : il s'agit d'un mécanisme d'appel de procédure à distance défini en XML :
Ex : <methodCall>
<methodName>example.method</methodName>
<params><param><value><param>42</param></value></param></params>
</methodCall>

SOAP (Simple Object Access Protocol) : SOAP est un protocole léger pour l'échange d'informations dans un environnement décentralisé et distribué. C'est un protocole basé sur XML qui se compose de trois parties : une enveloppe qui définit un cadre pour décrire ce qui est dans un message et comment le traiter, un ensemble de règles de codage pour exprimer des instances d'application définies, et une convention pour représenter des procédures distantes, leurs appels et réponses. SOAP peut potentiellement être utilisé en combinaison avec d'autres protocoles.

MIME (Multipurpose or MultiMedia Internet Mail Extensions) : MIME est un format de transmission de données utilisé par le courrier électronique, qui permet l'échange de données non textuelles, comme des graphiques, du son et de la vidéo.

IDL (Interface Description Language) : IDL est un langage de définition d'interface élaboré par l'OMG qui définit les types des objets en spécifiant leurs interfaces. Une interface consiste en un ensemble d'opérations nommées et leurs paramètres.

CORBA (Common Object Request Broker Architecture) : CORBA est une norme définissant l'architecture que doivent adopter les systèmes d'exploitation et les applications, pour

rendre possible la communication entre des objets provenant d'environnements différents.

DCOM (Distributed COM) : DCOM est une architecture définie par Microsoft permettant à une application d'en contrôler une autre en manipulant ses objets COM distants à travers un réseau comme si elle lui appartenait.

COM (Component Object Model) : COM est un standard de compatibilité binaire entre objets défini par Microsoft.

XML (eXtensible Markup Language) : XML est un format universel de stockage et d'échange de données. XML est un langage de balisage semi-structuré et extensible : il n'est pas sémantiquement figé comme HTML. Il permet de définir ses propres balises, ce qui le rend adaptable et donc à même de stocker tous types d'informations. La force de la norme XML est de séparer les données et leur présentation (le fond et la forme).

XPath : XPath est un langage de requêtes qui permet d'adresser, de désigner, des objets structurels contenus dans un document XML. Il est conçu pour être utilisé par des recommendations telles que XSLT ou XML Query, qui ajouteront à cette recommandation partagée les fonctionnalités propres qui leurs sont nécessaires. XPath est donc une recommandation de base, externalisée pour être partagée par un ensemble de recommandations de plus haut niveau.

XQuery : XQuery est un langage de requêtes sur un document XML (à la manière de SQL pour interroger des bases de données). Il permet, par exemple, de rechercher simplement tous les éléments ayant un nom donné.

DTD (Document Type Definition) : DTD établit la structure d'un document XML, c'est-à-dire l'arborescence (ordre et imbrication) et le caractère obligatoire ou facultatif des éléments qui composent un document. De plus, elle décrit chaque élément.

XML Schema : XML Schema est un langage de description de la structure et du contenu d'une certaine classe de documents XML : il permet de valider les éléments et les attributs d'un document XML en vérifiant leurs types. Il a la particularité d'être fortement orienté objet. Pour cela, il permet de définir des types complexes par agrégation ou dérivation.

XSL (Langage de feuille de style extensible) : XSL est un langage de présentation compatible avec CSS, dont il est un sur-ensemble. Une feuille de style XSL permet de mettre en page et de reformater (ou réordonner) le document XML auquel elle est associée.

Namespace (Espace de noms) : Ces espaces de noms permettent de créer des sous-ensembles parmi les éléments qui composent un document XML. Tous les éléments appartenant à un namespace sont préfixés par son identifiant suivi du signe deux-points (:). Ce système permet de lever toute ambiguïté en fournissant des identifiants uniques. Il correspond à la notion de paquetage des langages orientés-objets.

DOM (Document Object Model) : DOM est une norme définissant une interface de programmation (indépendante de tout langage) qui permet d'accéder à une représentation "orientée objet" des documents XML. En utilisant DOM, les développeurs peuvent manipuler et parcourir de façon arborescente un document constitué de nœuds.

LDAP (Lightweight Directory Access Protocol) : LDAP est un protocole de gestion d'annuaires, permettant à des clients Internet d'accéder automatiquement à des services d'annuaires en ligne sur TCP.

Bibliographie

[1] Cisco configuration registrar. http ://www.cisco.com/en/US/products/sw/netmgtsw/ps4618/.

[2] Omg corba specification 3.0. http ://www.corba.org/.

[3] Syncml data management protocol, version 1.0. http ://www.openmobilealliance.org/syncml/ downloads.html.

[4] Syncml sync protocol, version 1.0. http ://www.openmobilealliance.org/syncml/ downloads.html.

[5] Omg interface definition language (idl). Standard 14750, ISO, March 1999.

[6] Xml path language (xpath) version 2.0. Working-draft, W3C, April 2002.

[7] Xquery 1.0 : An xml query language. Working-draft, W3C, April 2002.

[8] S. Abiteboul, O. Benjelloun, I. Manolescu, T. Milo, and R. Weber. Active xml : A data-centric perspective on web services. Evry, France, October 2002. Bases de Données Avancées (BDA'2002). http ://www-rocq.inria.fr/verso/Gemo/Projects/axml/.

[9] S. Abiteboul, O. Benjelloun, I. Manolescu, T. Milo, and R. Weber. Active xml : Peer-to-peer data and web services integration. Hong Kong, China, August 2002. Very Large Data Bases (VLDB'2002). http ://www-rocq.inria.fr/verso/Gemo/Projects/axml/.

[10] S. Abiteboul, A. Bonifati, G. Cobéna, I. Manolescu, and T. Milo. Dynamic xml documents with distribution and replication. San Diego, California, USA, June 2003. ACM SIGMOD. http ://www-rocq.inria.fr/verso/Gemo/Projects/axml/.

[11] G. Kakivaya D. Box, D. Ehnebuske and A. Layman. Simple object access protocol (soap) 1.1. Note, W3C, May 2000. http ://www.w3.org/TR/SOAP/.

[12] L. Garrett D. Thompson, C. Exton, A.S.M. Sajeev, and D. Watkins. Distributed component object model (dcom).

[13] G. Meredith E. Christensen, F. Curbera and S. Weerawarana. Web service description language (wsdl) 1.1. Note, W3C, March 2001. http ://www.w3.org/TR/wsdl.

[14] T. Goddard. Towards xml based management and configuration. Network working group internet-draft, IETF, June 2002.

[15] S. Han Y. Oh J. Yoon H. Lee H. Ju, M. Choi and J.W. Hong. An embeded web server architecture for xml-based network management. In M. Ulema R. Stadler, editor, *Integrated Network Management*, pages 5–18, Florence, Italy, April 2002.

[16] M. Schoffstall J.Case, M.Fedor and J. Davin (Eds.). A simple network management protocol (snmp). Technical Report RFC 1157, IETF, May 1990.

[17] BJ. Lee and TS. Choi. Cli-based mediation mechanism using xml for configuration management. Network working group internet-draft, IETF, October 2003.

[18] J.P. Martin-Flatin. Push vs. pull in web-based network management. In S. Mazumdar M. Sloman and E. Lupu, editors, *Integrated Network Management*, volume VI, pages 3–18, Boston, MA, USA, May 1999. IFIP/IEEE International Symposium on Integrated Network Management (IM'99), IEEE Press.

[19] J.P. Martin-Flatin. *Web-Based Management of IP Networks and Systems*. PhD thesis, Ecole Polytechnique de Lausanne, 2000.

[20] N. Borenstein N. Freed. Multipurpose internet mail extensions (mime). RFC 2045-2049, IETF, November 1996.

[21] N. Ben Youssef O. Festor. Wbem. Technical Report 3927, INRIA, Avril 2000.

[22] Y. Atarashi S. Miyake M.Kitani F. Baker M. Wasserman R. Atarashi, T. Shimojo. Xml configuration architecture. Network working group internet-draft, IETF, April 2003.

[23] Juniper Network R.Enns. Xmlconf configuration protocol. Network working group internet-draft, IETF, February 2003.

[24] Avaya Labs Research. Xml based management interface for snmp enabled devices, 2001. http ://www.research.avayalabs.com/user/mazum/Projects/XML/.

[25] J. Schönwälder. Specific simple network management tools (scli). pages 109–119. LISA 2001, December 2001. http ://www.ibr.cs.tu-bs.de/projects/scli/.

[26] P. Shafer and R. Enns. JUNOScript : An xml-based network management api. Network working group internet-draft, IETF, August 2002.

[27] F. Strauss and T. Klie. Towards xml oriented internet management. In G. Goldszmidt and J. Schönwälder, editors, *Integrated Network Management*, volume VIII, pages 505–518, Colorado Springs, USA, March 2003. IFIP/IEEE International Symposium on Integrated Network Management (IM'2003), KLUWER ACADEMIC PUBLISHERS.

[28] D. Ehnebuske A. Hately T. Bellwood, L. Clément. Universal description, discovery and integration specification. Technical report, OASIS, July 2002.

[29] C. M. Sperberg-McQueen E. Maler T. Bray, J. Paoli. Extensible markup language (xml) 1.0. Recommendation, W3C, February 1998. http ://www.w3.org/XML/.

[30] Igor Tatarinov, Zachary G. Ives, Alon Y. Halevy, and Daniel S. Weld. Updating XML. Santa Barbara, California, May 2001. ACM Sigmod.

[31] M. Wasserman. Concepts and requirements for xml network configuration. Network working group internet-draft, IETF, June 2002.

[32] M. Choi Y. Oh, H. Ju and J.W. Hong. Interaction translation methods for xml/snmp gateway. volume 2506, Montreal, Canada, October 2002. 13th IFIP/IEEE International Workshop on Distributed Systems : Operations and Management (DSOM'2002), Springer.

Annexes

Annexe A

Passerelles Postech

L'équipe Postech propose trois méthodes pour réaliser une passerelle XML/SNMP.

– La première approche (Cf. Figure A.1) consiste à implémenter une interface XML DOM permettant de traduire des appels de fonctions DOM en opérations SNMP. Les résultats des opérations SNMP exécutées sont ensuite répercutés, d'abord sur les noeuds XML et ensuite sur l'application de management basée sur DOM. Dans cette approche la passerelle est fusionnée à l'application de management à travers l'API DOM. A.1.

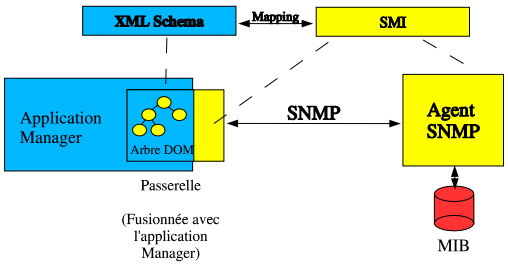

FIG. A.1 – Première approche d'intégration de passerelle de l'équipe Postech

– Dans la seconde approche (Cf. Figure A.2), une passerelle indépendante accepte, en guise de protocole de management, des requêtes HTTP de la part des stations de management. Ces dernières sont basées sur des URIs pouvant contenir des expressions XPath ou XQuery pour adresser des objets spécifiques des agents. La passerelle traduit la requête en opération SNMP destinées aux agents : HTTP GET pour l'obtention d'une donnée de management, HTTP POST pour la mise à jour. Les réponses SNMP sont utilisées pour créer un document XML de réponse qui est ensuite envoyé au manager en réponse à sa requête. Enfin, les traps SNMP destinés à la passerelle peuvent être redistribués à travers le même mécanisme grâce à une requête HTTP POST.

– Dans la troisième approche (Cf. Figure A.3), la passerelle implémente un service RPC SOAP en guise de protocole de management avec le manager. Á la réception d'un message SOAP d'un manager, ce dernier est traduit en opérations SNMP. De la même manière, les réponses SNMP sont traduites en message SOAP sortant. Cette approche introduit un excès de lourdeur au niveau du traitement et du protocole, mais elle apporte aussi les

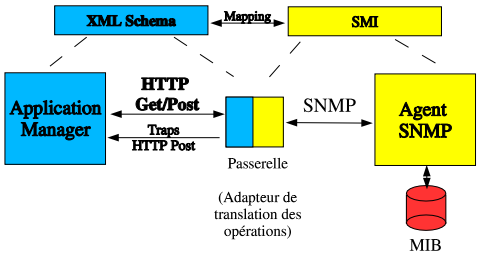

FIG. A.2 – Deuxième approche d'intégration de passerelle de l'équipe Postech

meilleurs possibilités d'extension de la passerelle (ex : ajout d'opération de planification de récupération de valeur). Dans cette approche, l'identification est également effectuée grâce à XPath.

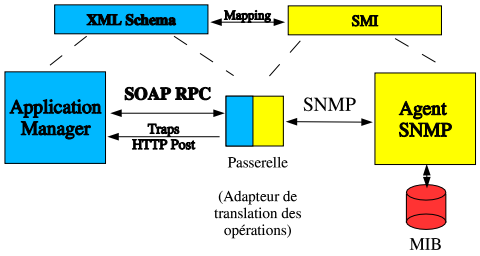

FIG. A.3 – Troisième approche d'intégration de passerelle de l'équipe Postech

Annexe B

Passerelle DaxFi/Web Services (Prototype)

Listing B.1 – DaxfiWSImpl.java

```
//
// java packages
//
import java.net.*;
import java.io.*;
import java.util.*;
import java.text.*;

//AXIS
import org.apache.axis.MessageContext;

// DOM classes.
import org.w3c.dom.*;

//JAXP 1.1
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
import javax.xml.transform.dom.*;

public class DaxfiWsImpl {
    private OutputStream fwRulesOut=null;
    private OutputStream defaultFwRulesOut=null;
    private PrintWriter debugOut=null;

    //Error handler
    private String nextStepDesc="";
    private boolean errorOccured;

    public Document doSubmission(Document inDoc){

    Document result=null;

    try{
        //-------------------------------------------------------------------------------------------
        //define default firewallRules.xml output
        //-------------------------------------------------------------------------------------------

        defaultFwRulesOut=new FileOutputStream("firewallRules.xml");
        if(fwRulesOut==null)
        fwRulesOut=defaultFwRulesOut;

        nextStepDesc="Build concrete firewallRules.xml output";

        //-------------------------------------------------------------------------------------------
        //Build concrete firewallRules.xml output
        //-------------------------------------------------------------------------------------------

        printDebugMsg("Firewall rules generation in progress ...");

        DocumentBuilderFactory factory=DocumentBuilderFactory.newInstance();
        DocumentBuilder builder=factory.newDocumentBuilder();
        Document fwRulesDoc=builder.newDocument();

        Element rootFwRulesDoc = (Element) fwRulesDoc.createElement("ruleset");
        fwRulesDoc.appendChild(rootFwRulesDoc);

        String[] tags={"flush","append","insert","replace","delete"};
```

```
        for(int j=0;j<tags.length;j++){
        NodeList inElements=inDoc.getElementsByTagName(tags[j]);

        for(int i=0;i<inElements.getLength();i++){
            rootFwRulesDoc.appendChild(fwRulesDoc.importNode(inElements.item(i),true));
        }
        }

        nextStepDesc="Serialization of firewallRules.xml";

        // Serialisation through Tranform.
        DOMSource domSource = new DOMSource(fwRulesDoc);
        StreamResult streamResult = new StreamResult(fwRulesOut);
        TransformerFactory tf = TransformerFactory.newInstance();
        Transformer serializer = tf.newTransformer();
        serializer.setOutputProperty(OutputKeys.ENCODING,"ISO-8859-1");
        //       serializer.setOutputProperty(OutputKeys.DOCTYPE_SYSTEM,"users.dtd");
        serializer.setOutputProperty(OutputKeys.INDENT,"yes");
        printDebugMsg("Firewall rules generated");
        serializer.transform(domSource, streamResult);



        nextStepDesc="Call daxfixmlfile for iptables update with firewallRules.xml as parameter";

        //-----------------------------------------------------------------------------------------------
        //Call daxfixmlfile for iptables update with firewallRules.xml as parameter
        //-----------------------------------------------------------------------------------------------
        printDebugMsg("Call daxfixmlfile for iptables update");

        if(fwRulesOut==defaultFwRulesOut){
        Process p = Runtime.getRuntime().exec("daxfixmlfile -f iptables -x firewallRules.xml");
        p.waitFor();
        printDebugMsg("Firewall updated !!!");
        }
        else{
        printDebugMsg("Firewall NOT updated !!! (OutputStream redirected)");
        }

        nextStepDesc="Build answer";

    }
    catch(Exception ex){
        printDebugMsg("### ERROR ###");
        if(debugOut!=null)
        ex.printStackTrace(debugOut);
        else
        ex.printStackTrace();
        errorOccured=true;
    }

    try{
        //-----------------------------------------------------------------------------------------------
        //Build answer
        //-----------------------------------------------------------------------------------------------

        printDebugMsg("Preparing the answer");

        DocumentBuilderFactory factory=DocumentBuilderFactory.newInstance();
        DocumentBuilder builder=factory.newDocumentBuilder();
        result=builder.newDocument();

        if(!errorOccured){
        // Prepare result attributes
        InetAddress myIP=InetAddress.getLocalHost();
        SimpleDateFormat sdf = new SimpleDateFormat("dd-MMM-yy hh:mm:ss a");
        Date date = new Date();
        String myDate = sdf.format(date);

        Element el=result.createElement("FirewallUpdated");
        el.setAttribute("firewallIp",myIP.getHostAddress());
        el.setAttribute("firewallUpdDate",myDate);
        result.appendChild(el);
        }
        else{
        //Error occured during firewall rules update
        Element el=result.createElement("FirewallNotUpdated");
        el.setAttribute("errorDesc","ERROR : "+nextStepDesc);
        result.appendChild(el);
        }

    }
    catch(Exception ex){
        if(debugOut!=null)
        ex.printStackTrace(debugOut);
        else
        ex.printStackTrace();

    }

    printDebugMsg("Send the answer");
    return result;
```

```
    }

    public void redirectFWRulesOut(OutputStream out){
    fwRulesOut=out;
    }

    public void debugOutput(OutputStream out){
    debugOut=new PrintWriter(out,true);
    debugOut.println("DEBUG␣MODE");
    }

    private void printDebugMsg(String msg){
    if(debugOut!=null)
        debugOut.println("***␣"+msg+"␣***");
    }

}
```

Annexe C

Document de management (prototype)

Listing C.2 – Prototype3.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<firewall xmlns:axml="http://www-rocq.inria.fr/verso/AXML" xmlns:daxfiWs="urn:daxfiWs">

<data>

  <!-- Request Firewall 2 rules -->
  <axml:sc frequency="every 25000" id="6BB80CBB-9851-07C1-01C7-865BFD047B73" methodName="get">
   <serviceURL="http://localhost:8080/axml2/services/getFirewallRules" valid="last">
  </axml:sc>
  <rules axml:origin="6BB80CBB-9851-07C1-01C7-865BFD047B73">
    <flush>
      <rule direction="in"/>
    </flush>
    <append>
      <rule direction='in' source-ip='192.168.1.0/24'><tcp /><accept /></rule>
    </append>
  </rules>

  <!-- Request Firewall 3 rules -->
  <axml:sc frequency="every 30000" id="6BB80CC4-9851-07C1-0129-5FE82A2F7E40" methodName="get">
   <serviceURL="http://localhost:8080/axml3/services/getFirewallRules" valid="last">
  </axml:sc>
  <rules axml:origin="6BB80CC4-9851-07C1-0129-5FE82A2F7E40">
    <flush>
      <rule direction="in"/>
    </flush>
    <append>
      <rule direction='in' source-ip='192.168.1.0/24'><tcp /><accept /></rule>
      <rule direction='in' source-ip='192.168.2.0/24'><tcp /><accept /></rule>
    </append>
  </rules>
</data>

<control>
  <axml:sc frequency="every 60000" id="6BB80CCF-9851-07C1-0107-1C841B1D3794" methodName="doSubmission">
   <serviceURL="http://localhost:8080/axis/services/firewall" valid="last">
    <axml:params>
      <axml:param name="in1" xpath="//data"/>
    </axml:params>
  </axml:sc>
</control>

</firewall>
```

Annexe D

Document de définition des Web Services associés (prototype)

Listing D.3 – getFirewallRules.xml

```
<!-- A slightly more complex one, that uses a parameter. -->

<serviceDefinition type="query">

  <query><![CDATA[
  select r
  from r in firewallRules//rules;]]>
  </query>

</serviceDefinition>
```

Annexe E

Passerelle Snmp/Web Services

Listing E.4 – SnmpMonitoringWsInterface.java

```
public interface SnmpMonitoringWsInterface{
    public String get(String oid,String agentAddress);
}
```

Listing E.5 – SnmpMonitoringWsImpl.java

```
//
// SNMP related packages
//
import uk.co.westhawk.snmp.pdu.*;
import uk.co.westhawk.snmp.stack.*;

//
// java packages
//
import java.net.*;
import java.util.*;

public class SnmpMonitoringWsImpl implements SnmpMonitoringWsInterface,Observer {

    private String oid = null;
    private java.net.InetAddress agent = null;
    private String value = null;
    private String protocol="snmp";
    private String name;
    private SnmpContext snmpcontext;

    private BlockPdu pdu;

    private void debug(String msg){
    System.out.println("----------SNMP-MONITORING-WS----------");
    System.out.println(msg);
    System.out.println("--------------------------------------");
    }

    public void update(Observable getreq, Object arg){
    debug("Update");
    if(protocol.equals("snmp")){

        if(arg instanceof varbind){
            varbind onevb = (varbind)arg;
        value=onevb.getValue().toString();
        debug("Update ars="+arg.toString()+"\n       value="+value);
        snmpcontext.destroy();
        }
    }
    }

    public String get(String oid,String agentAddress){
    //Creation of the agent from its IP address or Dns name
    try{
         agent = java.net.InetAddress.getByName(agentAddress);
    }
    catch(Exception e){e.printStackTrace();}

    if(protocol.equals("snmp")){
        if(oid != null && agent != null){
        try{
            snmpcontext = new SnmpContext(agent.getHostAddress(), 161, SnmpContextFace.STANDARD_SOCKET);

            sendGetRequest(oid);
```

```
            //debug("snmpcontext : "+snmpcontext.toString());
            //OneGetPdu onegetn = new OneGetPdu(snmpcontext, oid, this);
            //debug("onegetn : "+onegetn.toString());
            }
        catch(Exception e){
            e.printStackTrace();
        }
        }
    }

    return value;
    }

    private void sendGetRequest(String oid){
    pdu = new BlockPdu(snmpcontext);
    pdu.setPduType(BlockPdu.GET);
    pdu.addOid(oid);
    sendRequest(pdu);
    }

    private void sendRequest(BlockPdu pdu){
    try {
        varbind var = pdu.getResponseVariableBinding();
        AsnObjectId oid = var.getOid();
        AsnObject res = var.getValue();
        if (res != null)
        value="<value oid="+oid.toString()+">"+res.toString()+"</value>\n";
        else
        value="error null result";
    }
    catch (PduException exc){
        // give the user feedback
        exc.printStackTrace();
    }
    catch (java.io.IOException exc){
        // give the user feedback
        exc.printStackTrace();
    }
    }


}
```