Université de Toulouse

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)

Présentée et soutenue par :

Audrey MOUI

le 2 juillet 2013

Titre :

UN CADRICIEL POUR UNE SURVEILLANCE ADAPTATIVE
DE RESEAUX ET SYSTEMES COMPLEXES

École doctorale et discipline ou spécialité :

ED MITT : Domaine STIC : Réseaux, Télécoms, Systèmes et Architecture

Unité de recherche :

UMR 5505

Directeur(s) de Thèse :

Michelle SIBILLA
Thierry DESPRATS

Jury :

Noémie SIMONI, Professeur Telecom ParisTech (rapporteur)
Chantal TACONET, Maître de conférence HDR Telecom SudParis (rapporteur)
Christophe CHASSOT, Professeur LAAS-CNRS Toulouse (examinateur)
Jean-Pierre GAUBERT, Expert ingénieur Airbus (examinateur)
Thierry DESPRATS, Maître de conférence Université Paul Sabatier (encadrant)
Michelle SIBILLA, Professeur Université Paul Sabatier (directrice de thèse)

Résumé

Les nouveaux paradigmes de gestion intégrée de réseaux, de services, de systèmes complexes hétérogènes reposent sur davantage d'autonomie et de décentralisation décisionnelles. Leurs mises en œuvre doivent permettre de doter ces systèmes gérés d'un degré d'auto-adaptation élevé face à des perturbations survenues ou prédictibles. Ainsi, sont portées de plus en plus d'exigences quant à l'efficience de l'activité de surveillance de l'état et du comportement du système géré. Pouvoir efficacement et dynamiquement adapter l'activité de surveillance à la demande d'activités d'analyse constitue un facteur d'amélioration globale de la gestion (exigences fonctionnelles) ; une activité de surveillance doit également pouvoir s'adapter face aux variations de contraintes liées à son environnement d'exécution ou à ses impacts possibles sur le système géré, voire, de s'auto-optimiser (contraintes opérationnelles). Des travaux ont contribué à rendre la surveillance adaptative, mais chacune des approches étudiées ne traite que d'un aspect particulier, ou n'est dédiée qu'à un environnement spécifique.

L'objectif des travaux présentés dans ce mémoire est de proposer une solution logicielle (cadriciel) apte à supporter l'automatisation de l'adaptation d'une activité de surveillance. Une telle adaptation résulte en une modification de la manière de surveiller. Ainsi, il est nécessaire de définir un plan de contrôle visant à gérer l'ensemble des mécanismes sous-jacents utilisés pour surveiller effectivement le système géré. Afin d'apporter une réponse logicielle la plus générique, modulaire et réutilisable possible, nous avons identifié trois capacités à partir desquelles a été architecturé le cadriciel proposé :

- la configurabilité : la capacité d'initialiser et de modifier dynamiquement et sans interruption la portée et les valeurs des paramètres gouvernant le comportement des mécanismes de surveillance (par exemple *polling* ou *event reporting*),
- l'adaptabilité : la capacité de modifier dynamiquement et sans interruption le comportement d'une activité de surveillance, donc d'exécuter l'adaptation de la surveillance,
- la gouvernabilité : la capacité de détecter un besoin d'adaptation et de déclencher cette adaptation des mécanismes de surveillance.

Chacune de ces couches du cadriciel a été conceptualisée, puis implémentée, testée et évaluée. Pour la configurabilité, les mécanismes de surveillance ainsi que les éléments gérés sur lesquels ils opèrent (la portée) ont été spécifiés de manière semi-formelle afin d'identifier une liste de paramètres de configuration qui gouvernent le comportement respectif de ces mécanismes.

Pour l'adaptabilité, a été identifié et spécifié un ensemble d'opérateurs permettant d'ajouter, de modifier et de supprimer des mécanismes de surveillance de manière dynamique, mais également permettant de modifier leur état opérationnel. Ces opérateurs offrent un service, support à la reconfiguration d'une activité globale de surveillance, dont l'interface a été bien définie et les algorithmes sous-jacents décrits.

Concernant la gouvernabilité, le paradigme de gestion à base de politiques de type ECA a été retenu de par sa prédisposition générale à exprimer des logiques de pilotage à base d'événements.

Notre approche, ses modèles et leur formalisation sont indépendants de tout environnement technologique. Ils ont été implémentés au sein d'un prototype basé sur un environnement CIM/WBEM de gestion intégrée (OpenPegasus) et sur le *framework* orienté politiques Ponder2. Ce démonstrateur a permis de montrer la faisabilité de l'approche et d'évaluer, à travers un cas d'étude, sa capacité à répondre à la fois aux variations d'exigences fonctionnelles et de contraintes opérationnelles. Cette adaptation dynamique de la surveillance est réalisée sans interruption du système. Le démonstrateur a aussi permis d'estimer le surcoût temporel lié à la gestion de l'adaptation et d'analyser les impacts liés à l'usage des différents opérateurs.

Remerciements

Michelle et Thierry, je tiens à vous dire un grand merci, pour m'avoir acceptée dans votre équipe, accompagnée pendant cette période, et orientée dans mes travaux. Ce manuscrit est de fait le résultat de cette collaboration :

Mme Michelle Sibilla, Professeur à l'université Paul Sabatier de Toulouse (directrice de thèse)
M. Thierry Desprats, Maître de conférence à l'université Paul Sabatier de Toulouse (encadrant)

Je tiens aussi à remercier les relecteurs pour l'intérêt qu'ils ont porté à ce travail et pour les remarques constructives qu'ils ont fait sur mon projet de thèse, et les membres du jury d'avoir accepté de juger ce travail.

Mesdames les rapporteurs :

Mme Noémie Simoni, Professeur, Télécom ParisTech
Mme Chantal Taconet, Maître de conférence, Télécom SudParis

Messieurs les examinateurs :

M. Christophe Chassot, Professeur, LAAS-CNRS Toulouse
M. Jean-Pierre Gaubert, Expert, Airbus

Je vais bien entendu aussi remercier Ludi, Romain, Manu, Cédric, Tony (...) et tous ceux que j'ai cotoyés au laboratoire pendant cette période, et Daniel Marquié qui a toujours été de bons conseils pour la partie enseignement.

Et je ne vais pas oublier Didier, mon professeur de guitare, qui me permettait de décrocher de temps à autre ;-)

Table des matières

Table des figures

Liste des tableaux

Première partie

Introduction générale

Introduction générale

Cadre des travaux

Les nouveaux paradigmes de gestion intégrée de réseaux, de services, de systèmes complexes hétérogènes, embarqués et/ou mobiles, parfois virtualisés, reposent sur davantage d'autonomie et de décentralisation décisionnelles. Leurs mises en œuvre doivent permettre de doter ces systèmes gérés d'un degré d'auto-adaptation élevé – en phase opérationnelle – de leur structure et/ou de leur comportement face à des perturbations survenues ou prédictibles. Cette adaptation sera d'autant plus efficace qu'elle conduira à des reconfigurations décidées le plus pertinemment possible, opérées « *on line* », de tout ou partie du système géré.

Sur la base d'une « gestion intégrée » permettant de gérer de manière unifiée un ensemble d'éléments hétérogènes selon les aires fonctionnelles FCAPS, c'est autour de la boucle logique « MAPE » (*Monitoring* – *Analysis* – *Plan* – *Execute*) que s'organise classiquement le pilotage – autonome ou non – d'un système. La qualité du contrôle résulte certes de la qualité des mises en œuvre de chacune de ces quatre fonctions MAPE mais également de la richesse des échanges s'opérant entre elles. Plus particulièrement, la pertinence des conclusions d'une analyse (A) résulte en partie de la qualité de l'information qui lui est fournie par les mécanismes de surveillance (M).

Ainsi, la qualité de l'information collectée par la surveillance (M) se décline selon de multiples critères et ne peut, globalement, qu'être subjectivement évaluée. Néanmoins, elle peut être étudiée en regard des trois points de vue suivants :

1. selon le contexte « métier » dans lequel s'opère le processus d'analyse, il sera préféré, au cours de son déroulement, des informations tantôt plus ou moins exactes, précises, actuelles, complètes, fines… pour alimenter la connaissance momentanément nécessaire à la détection et l'interprétation de situations. Pouvoir efficacement et dynamiquement adapter l'activité de surveillance à la demande d'activités d'analyse constitue un facteur d'amélioration globale de la gestion (exigence fonctionnelle).
2. Par ailleurs, d'un point de vue opérationnel, l'exécution d'activités de surveillance automatisées consomme des ressources telles que du temps CPU, de l'espace mémoire, de l'énergie et parfois de la bande passante. Ainsi un environnement d'exécution peut contraindre l'activité de surveillance, soit directement, lorsque les ressources sont – partiellement ou totalement – temporairement indisponibles, ou indirectement lorsque des politiques de restriction de consommation de ressources sont à appliquer (par exemple : ne pas autoriser plus de 10 % de la bande passante globale pour du trafic de surveillance). Ainsi, une activité de surveillance a également besoin de s'adapter afin de considérer les variations des contraintes de son environnement d'exécution (contraintes opérationnelles).

3. Finalement, dans une vision autonomique d'une surveillance, des préoccupations d'auto-optimisation doivent être supportées par la fonction de surveillance afin d'augmenter son niveau d'efficience. Dans cette approche auto-adaptative, résultant d'une analyse introspective de ses propres performances, des décisions d'auto-ajustements peuvent être prises par la fonction de surveillance, mais ne pourront être appliquées qu'à la condition que la fonction de surveillance soit adaptable (auto-optimisation).

L'objectif de ces travaux est de proposer une solution logicielle (cadriciel) apte à supporter l'automatisation de l'adaptation d'une activité de surveillance au sein d'un système de gestion intégrée de réseaux et systèmes complexes communicants.

Etat de l'art

Dans le domaine de la gestion de réseaux et de systèmes, un certain nombre de travaux contribuent à rendre la surveillance adaptative, mais chacune des approches étudiées ne traite que d'un aspect particulier de la problématique, à savoir : comment rendre la surveillance adaptable dans un respect des contraintes opérationnelles, ou des objectifs métiers, voire dans un but d'auto-optimisation. Nous avons relevé deux limites majeures, à savoir :

- Manque de généricité : Tout d'abord, les solutions proposées sont souvent spécifiques, dans le sens où elles ne sont pas applicables dans un contexte général de gestion intégrée. Par exemple, Anemona offre des langages et des outils intéressants permettant de rendre la surveillance adaptable mais reste entièrement dédié pour une utilisation avec le protocole SNMP. RAP de son côté offre une solution permettant uniquement de gérer l'adaptabilité d'un seul mécanisme de surveillance (le *polling*) dans un but de performance uniquement. Ganglia, par contre, permet de reconfigurer tous les mécanismes en fonction de l'état courant d'un réseau mais ne porte pas sur d'autres types d'entités gérées que des réseaux de communication.
- Adaptation visant majoritairement la performance : Deuxième point à mettre en évidence, la plupart des approches tentent de répondre à la seule préoccupation d'améliorer la performance : particulièrement, « la surveillance ne peut pas surcharger le système surveillé » constitue l'objectif prédominant qui a guidé les travaux visant à considérer des demandes de reconfiguration d'une surveillance.

Cet état des lieux permet d'ores et déjà d'identifier grossièrement les caractéristiques d'un service de surveillance adaptative. Tout d'abord, un tel service doit s'inscrire dans un contexte de gestion intégrée, il doit être indépendant de toute architecture, de tout protocole. Le système sous-jacent n'a alors plus d'importance : tout type de composant, de protocoles doit pouvoir être pris en charge par le *framework* de surveillance adaptative. De ce fait, la solution proposée doit donc gérer tous les composants de manière unifiée tout en étant modulaire (il est possible de récupérer tout ou partie du *framework* selon les besoins définis), réutilisable (le *framework* doit pouvoir être intégré à n'importe quel cas d'utilisation, et doit pouvoir être adapté en fonction du nouveau contexte courant), dynamique (les règles de pilotage de l'adaptation de la surveillance ne doivent pas être figées dans des algorithmes, mais doivent pouvoir être modifiées en fonction des besoins qui sont, eux aussi, évolutifs)...

Contribution

La « surveillance adaptative » correspond à la capacité dont dispose une fonction de surveillance de décider et d'exécuter, sans interruption, l'ajustement de son comportement pour maintenir son efficacité dans le respect des exigences fonctionnelles et des contraintes opérationnelles, mais éventuellement aussi, pour améliorer son efficience selon des préoccupations d'auto-optimisation.

Pour répondre aux besoins dégagés dans l'état de l'art, une approche de type *bottom-up* a été suivie pour construire un cadriciel supportant l'automatisation d'adaptations d'une activité de surveillance.

Une analyse a révélé que pour influer sur la qualité de l'information collectée, ou bien pour répondre aux contraintes opérationnelles d'exécution ou encore pour gérer les performances de la surveillance elle-même, il est nécessaire d'agir tantôt sur la configuration individuelle des mécanismes de *polling* ou d'*event reporting*, tantôt sur l'ensemble de ces mécanismes en cours d'exécution. Ainsi, une adaptation résulte en une modification de la manière de surveiller. Automatiser l'adaptation de la surveillance nécessite de gérer et contrôler l'activité de surveillance. Ceci nous a conduit à définir un plan de contrôle visant à gérer l'ensemble des mécanismes sous-jacents utilisés pour surveiller effectivement le système géré.

A partir de la réception de deux types d'*inputs* (règles ou objectifs de haut niveau d'une part, contraintes environnementales ou opérationnelles d'autre part), une analyse est déclenchée au niveau du plan de contrôle de la surveillance et une décision d'adaptation de la surveillance peut être prise, de manière automatique. De ce fait, un ajustement du comportement de la surveillance peut être lancé pour améliorer l'efficacité de la surveillance, pour améliorer ses performances et pour ajuster la qualité des informations de gestion requises.

Afin d'apporter une réponse logicielle la plus générique, modulaire et réutilisable possible, pour la mise en œuvre de tels plans de contrôle, nous avons identifié trois capacités à partir desquelles a été architecturé notre cadriciel.

Ces trois couches « imbriquées » représentent les trois étapes successives de construction du *framework* adaptatif :

- CONFIGURABILITE : la capacité d'initialiser et de modifier dynamiquement et sans interruption la portée et les valeurs des paramètres gouvernant le comportement des mécanismes de surveillance,
- ADAPTABILITE : la capacité de modifier dynamiquement et sans interruption le comportement d'une activité de surveillance, donc d'exécuter l'adaptation de la surveillance,
- GOUVERNABILITE : la capacité de détecter un besoin d'adaptation et de déclencher cette adaptation (règles de déclenchement des opérateurs d'adaptation des mécanismes de surveillance).

Chacune de ces couches du *framework* a été définie formellement, conceptualisée, puis implémentée, testée et évaluée au sein d'un prototype basé sur un environnement CIM/WBEM de gestion intégrée.

Pour la configurabilité, les mécanismes de surveillance ainsi que les éléments gérés sur lesquels ils opèrent ont été spécifiés de manière semi-formelle afin d'identifier une liste de paramètres de configuration qui gouvernent les comportements respectifs des mécanismes de *polling* et d'*event reporting*. Ces paramètres ont ensuite été modélisés en CIM, puis instrumentés dans un serveur CIM de l'outil Open Pegasus.

Des algorithmes aptes à considérer ces configurations initiales ainsi que des demandes d'ajustement dynamique de ces mécanismes ont été conçus. Pour validation, deux programmes Java ont été développés comme des clients CIM et testés : le *poller* adaptable, et le *listener* adaptable. Leur but est de récupérer la configuration sur un serveur CIM et de l'appliquer pour que leur comportement soit conforme à la configuration enregistrée.

Pour l'adaptabilité, nous avons identifié et spécifié un ensemble d'opérateurs permettant d'ajouter, de modifier et de supprimer des mécanismes de surveillance (par exemple *polling* ou *event reporting*) de manière dynamique et sans entraver le comportement global de l'activité de surveillance, mais également permettant de modifier leur état opérationnel. Ces opérateurs offrent un service support à la reconfiguration d'une activité globale de surveillance.

L'interface d'adaptabilité a été implémentée dans l'environnement CIM. Des tests ont été réalisés pour montrer la faisabilité de l'approche mais également pour estimer le surcoût temporel lié à la gestion de l'adaptation et analyser les impacts liés à l'usage des différents opérateurs. Le temps de traitement des modèles CIM a été précisément mesuré.

Concernant la gouvernabilité, nous avons opté pour le paradigme de gestion à base de politiques de par sa prédisposition générale à exprimer des logiques de pilotage. Suite à une étude comparative menée sur une dizaine de langages de politiques et des outils associés, le *framework* Ponder a été retenu pour évaluation. En effet, d'abord le langage de politiques qu'il propose supporte l'expression d'évaluation de règles sur occurrence d'événement, ensuite il offre la possibilité d'ajuster une politique en cours d'exécution par la modification des règles la constituant, et enfin, il permet une implémentation en Java des actions d'une politique grâce à son API Java Ponder2. Ainsi, un ensemble de politiques Ponder2 a pu être développé et intégré au sein du prototype afin de montrer la faisabilité d'une gouvernabilité à base de politiques, mais également celle de l'approche dans sa globalité.

Un cas d'étude a été mis en place pour observer le comportement de la surveillance en fonction des trois types de variations de contraintes identifiées :

- une certaine fraîcheur des données peut être imposée (contrainte métier),
- le système géré peut être modifié par l'ajout d'un équipement qu'il faut aussi surveiller (contrainte opérationnelle),
- le surveillance peut avoir à répondre dans certains cas à un besoin d'adaptation rapide ou bien d'optimisation de son comportement (contrainte d'optimisation).

Le cas d'étude a non seulement démontré la faisabilité de l'approche, mais aussi sa simplicité de mise en œuvre. Pour intégrer une nouvelle variation de contrainte, il suffit d'implémenter une politique Ponder2 lui correspondant, à savoir, l'événement déclencheur, la condition (variation de contrainte), et les actions résultantes. Ces actions correspondent à un ensemble d'un ou plusieurs opérateurs d'adaptations (déjà développés) qu'il suffit d'invoquer avec les arguments corrects pour lancer l'adaptation correspondante.

Organisation du document

Le reste de ce manuscrit est organisé comme suit. La première partie est générale et vise a définir la surveillance intégrée et les problématiques associées à sa mise en place dans les systèmes complexes actuels. Un état de l'art est également dressé pour mettre en évidence comment est

actuellement rendue adaptable la surveillance. Enfin, est présenté notre vision du *framework* idéal permettant de piloter l'adaptation de la surveillance dynamiquement. Ce framework ayant trois capacités particulières ayant chacune fait l'objet d'une contribution pour rendre la surveillance adaptative, chacune de ces trois capacités fait l'objet d'une partie. Définition, spécification semi-formelle et présentation d'algorithmes, exemple d'implémentation en CIM, et résultats (exemples de fonctionnement et éventuellement des mesures) rythment chacune de ces parties.

Deuxième partie

Autour de la surveillance intégrée : contexte, problématique, état de l'art

D'une surveillance intégrée vers son adaptation

1

Le but de ce chapitre est de positionner le contexte de travail (la gestion intégrée) et de définir la surveillance dite « intégrée ». La problématique du document peut ensuite être explicitée.

1.1 Définitions et positionnement du contexte général

1.1.1 Qu'est-ce que la gestion de réseaux et de systèmes ?

La gestion d'un système est définie dans [ISO10040] et concerne l'ensemble des modèles relatifs à ce système :

- les modèles organisationnels tout d'abord correspondent aux concepts d'agents et de *managers*. Agent et *manager* sont utilisés pour échanger des informations. Un agent est positionné sur un élément à surveiller de sorte à envoyer les informations qu'un *manager* lui demande (mode *polling*) ou qu'il juge nécessaire de lui envoyer (mode *event reporting*). Le *manager* est positionné quant à lui sur une station de gestion, et récupère les informations de surveillance pour les analyser.
- les modèles protocolaires concernent l'ensemble des protocoles de communication pouvant être utilisés pour faire communiquer entre eux les éléments du système. Il peut s'agir de SNMP, Netflow…
- les modèles informationnels enfin regroupent l'ensemble des informations concernant les éléments du système géré : il peut s'agir de statistiques, de vues particulières sur ces éléments, de configurations…

Pour gérer un tel système, il est nécessaire d'agir sur chacune de ses entités en les surveillant, contrôlant et coordonnant de sorte à supporter les cinq aires fonctionnelles appelées « FCAPS » [ISO10164] :

- (F) la « gestion des anomalies » (*fault management*) permet de détecter, localiser et réparer des pannes, puis de rétablir ensuite le service interrompu par ces pannes,
- (C) la « gestion des configurations » (*configuration management*) permet de désigner et de paramétrer les équipements,

- (A) la « gestion des consommations » (*accounting management*) permet de connaître les charges des équipements et les coûts relatifs à la consommation,
- (P) la « gestion de la performance » (*performance management*) permet de collecter des données à des fins d'analyse statistique et d'optimisation du comportement,
- (S) la « gestion de la sécurité » (*security management*) permet de contrôler et distribuer les informations relatives à la sécurité (cela inclut les problématiques de cryptage et de droit d'accès).

Ainsi un ensemble d'activités a été mis en place de sorte à mettre en oeuvre cette activité de gestion. Ces activités entrent en jeu dans la boucle de gestion MAPE [IBM03] définissant les quatre étapes d'un processus de contrôle, à savoir :

- la « surveillance » (*Monitor*) du système collecte les données relatives à l'état et au comportement du système,
- les données collectées sont alors « analysées » (*Analyze*) pour identifier une situation, un état ou un comportement qui nécessiterait une adaptation du comportement du système géré,
- selon les résultats d'analyse mis en évidence, une « décision » réactive ou proactive est prise (*Plan*) dans le but d'adapter le comportement du système sous-jacent,
- si une décision a été prise, les actions d'ajustement correspondantes sont « exécutées » (*Execute*) sur le système pour améliorer son efficacité.

En somme, la boucle MAPE permet de collecter des données de gestion en provenance du système géré, de les analyser, et enfin de prendre une décision de contrôle pour lancer des actions (pouvant être de reconfiguration).

Mais dans le cas de systèmes complexes, la gestion devient problématique car les systèmes sont composites, hétérogènes. Deux solutions se présentent :

- l'une basée sur des systèmes de gestion spécifiques à chaque type de composant d'un système complexe (figure 1.1) : par exemple, un (ou plusieurs) gestionnaire(s) de réseau, un (ou plusieurs) gestionnaire(s) de système d'exploitation, un (ou plusieurs) gestionnaire(s) de services... Il est impératif alors de faire un gros effort pour la mise en oeuvre des FCAPS : les composants sont dépendants, donc la gestion du composite dépend de la gestion des composants. Il faut donc jongler entre différents gestionnaires hétérogènes, spécifiques, et utilisant des protocoles et des modèles d'information de gestion dédiés. Ainsi, pour bâtir de la gestion automatisée, l'effort est considérable car les gestionnaires sont cloisonnés.
- l'autre offrant un niveau d'intégration qui permet de pouvoir développer des FCAPS (une combinaison fonctionnelle est rendue possible) à partir d'une représentation unifiée des données de gestion, d'un protocole unificateur et d'une couche qui permet d'intégrer des gestionnaires spécifiques existants pour chaque type de composants du système complexe géré. Nous parlons alors de gestion intégrée.

1.1.2 Qu'est-ce que la gestion intégrée ?

Les systèmes informatisés actuels sont dits « complexes ». Réseaux, systèmes, services doivent gérer l'ensemble des spécificités de chacun des composants du système global : ceux-ci peuvent être hétérogènes (de nature différente : un *switch* n'est pas appréhendé comme un serveur Web), de complexité différente (il est plus aisé de configurer un *switch* qu'un *firewall*), mais aussi de taille différente (un composant peut être un segment de réseau complet). Ajoutés à cela, certains

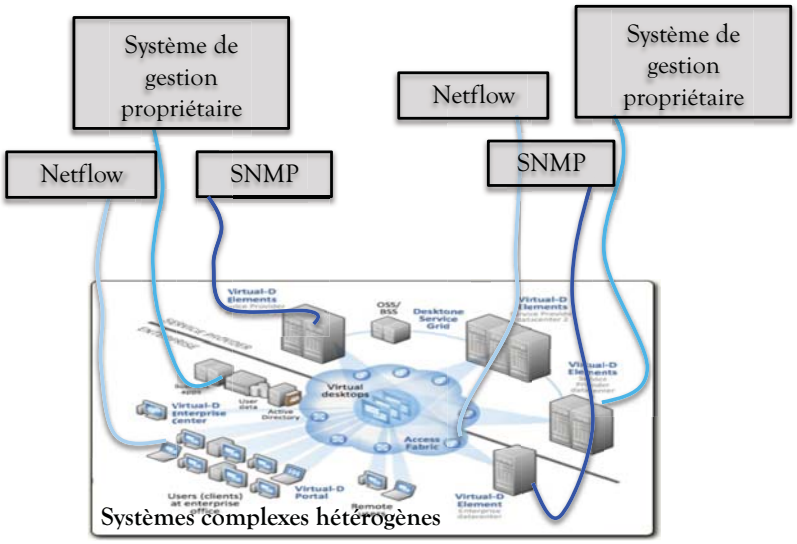

FIGURE 1.1: Les systèmes de gestion spécifiques

éléments peuvent être dépendants les uns des autres (un service est hébergé sur un serveur), mobiles, embarqués ou encore virtuels. Compte tenu de la multiplicité de ces éléments, la quantité d'informations en circulation augmente elle aussi.

Un premier problème résulte de cette complexité : **l'augmentation de la quantité de données**, par exemple, peut tendre à diminuer la performance [1] du système global et son efficacité. Alors, comment vérifier que le système réalise bien toutes les actions que l'on attend de lui dans un tel environnement, où chaque composant semble devoir être géré selon ses propres spécificités ?

Un second problème apparaît compte tenu qu'un système a *a priori* pour exigence de **répondre de manière globale à un (ou plusieurs) objectif(s) métier**. Comment déterminer les opérations de ce système global pour qu'il réponde aux objectifs métier ?

De plus, hétérogénéité des composants, complexité, grande échelle, dépendance, mobilité ou encore virtualisation sont des caractéristiques qui complexifient fortement la gestion intégrée. Or, cette gestion est nécessaire pour que le système géré puisse livrer dans de bonnes conditions les données et services attendus.

Ainsi, une piste de solution serait de s'abstraire de toutes ces spécificités en raisonnant sur une

1. La performance d'un système est la mesure chiffrée du fonctionnement de ce dernier pendant la réalisation des services que l'on attend de lui [Lahmadi08]. Elle est mesurée au moyen de six métriques :
- le temps de réponse (durée entre l'invocation d'un service et la fin de sa réalisation),
- le débit (quantité de trafic supporté par le système sur un intervalle de temps défini),
- l'utilisation des ressources consommées pendant la réalisation du service,
- la fiabilité (probabilité qu'une erreur se produise lors de la réalisation du service),
- la disponibilité (fraction de temps durant laquelle le système est disponible pour répondre aux requêtes des utilisateurs),
- la capacité de passage à l'échelle (capacité à rendre les services de façon continue et sans dégradation).

base unifiée et indépendante des technologies.

Définition

La « gestion intégrée » permet de gérer un ensemble d'éléments complexes et hétérogènes comme une seule et même entité.

1.1.3 Qu'est-ce que la surveillance intégrée ?

Une approche unifiée doit être adoptée pour prendre en compte toutes les entités du système géré : une couche d'intégration est ainsi ajoutée entre le système géré et la surveillance. Le système géré, quel qu'il soit, doit être vu comme une « boîte noire ».

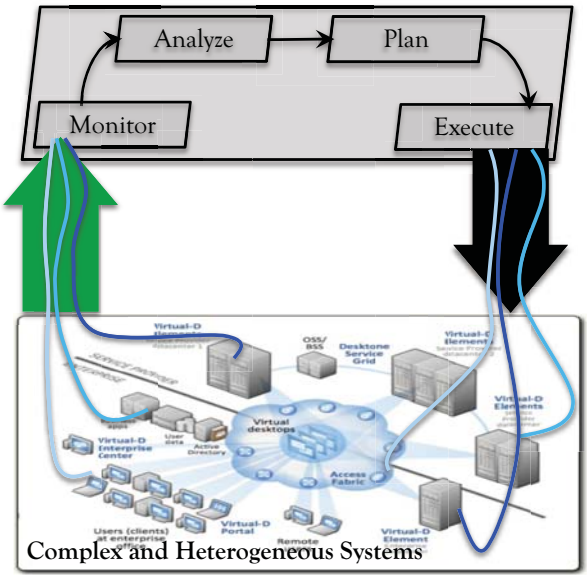

FIGURE 1.2: La boucle de gestion MAPE

La surveillance se place au sein de la boucle de contrôle MAPE (figure 1.2) en tant que première activité d'observation du système géré pour indirectement contribuer à déterminer s'il fonctionne correctement. Elle se définit comme suit [Samaan09] :

Définition

La « surveillance » (ou *monitoring*) est une activité de gestion permettant d'obtenir en temps réel, par collecte et/ou calcul, une vision claire et précise de l'état et du fonctionnement du système géré.

Positionnée dans un contexte de gestion intégrée, elle permet notamment de surveiller des composants hétérogènes et dépendants entre eux, et supporte des protocoles hétérogènes de gestion (il s'agit de la flèche verte sur la figure 1.2).

Voyons deux exemples pour montrer l'intérêt de la surveillance et de la gestion intégrée :

- Un réseau avionique est un système embarqué composé de deux réseaux redondés, de *switchs* et de liens physiques et/ou virtuels. Le système géré doit hétérogène. Chacun de ses éléments doit être surveillé et géré pour observer, notamment, son état selon les différentes étapes du

cycle de vie du système (conception, mise au point, essais, opérationnel). Des architectures doivent être successivement conçues, testées, maintenues (dans ce dernier cas des processus de maintenance peuvent demander une conception spécifique). La prise en compte de chacun de ces éléments spécifiques à chacune des étapes du cycle de vie a un coût, et impose un délai de réalisation. La gestion intégrée permet alors de règler ce problème, en considérant le système géré dans sa globalité et de manière unifiée.

- Un robot est un élément autonomique qui doit être capable de prendre en compte un ensemble de protocoles de communications différents pour pouvoir se positionner dans un environnement particulier dont il doit être conscient. Le mode opératoire d'un tel robot doit s'appuyer particulièrement sur des analyses réalisées à partir d'informations apportées par la surveillance intégrée. Un robot est généralement équipé de plusieurs interfaces de communication. Ainsi, selon la disponibilité et l'état courant des liaisons de communication associées à ces interfaces, selon leur consommation en terme d'énergie pour assurer une opération de communication, selon la capacité énergétique du robot, des décisions d'usage de telle ou telle interface de communication s'appuieront sur un ensemble d'informations hétérogènes et relatives à différents modules (communication, énergie, mission, etc.). Ainsi, une approche de surveillance intégrée apporte une unification des éléments d'informations respectifs et permet le développement de solutions globales de prises de décision.

Pour surveiller ces différents systèmes, il faut pouvoir faire abstraction de leur nature, de leur grande taille, de leur mobilité, de l'hétérogénéité de leurs composants et protocoles de communication.

1.2 Pourquoi et comment adapter la surveillance ?

Nos travaux ont porté essentiellement sur l'activité de surveillance : pourquoi a-t-on besoin d'adapter la surveillance ? comment peut-on envisager cette adaptation ?

1.2.1 Besoins

La surveillance permet d'obtenir une vision de l'état et du comportement courant du système géré. Mais pourquoi a-t-on besoin de cette « photographie » du système géré et à qui ou quoi cela sert-il ?

Ces éléments sont tous recueillis par le module d'analyse, qui va, comme son nom l'indique, analyser ces données collectées et/ou calculées pour tenter de répondre aux trois questions suivantes :

- le système géré fonctionne-t-il correctement ?
- si non : quelle est l'origine de son dysfonctionnement ?
- si oui : le système géré est-il performant ? Nécessiterait-il une reconfiguration pour mieux réaliser les opérations que l'on attend de lui ?

Toutefois un paradoxe apparaît ici : doit-on avoir une vision globale du système géré, pour savoir s'il fonctionne bien dans son ensemble ? ou doit-on se focaliser sur certains éléments particuliers de ce système (pour détecter une panne) ? De même comment déterminer que la surveillance apporte au module d'analyse des informations pertinentes ?

Les données collectées et/ou calculées par le module de surveillance et envoyées au module d'analyse sont définies par les éléments suivants :

- La « **portée de la surveillance** » correspond à la quantité d'éléments surveillés : Que surveille-t-on ? tout ? un sous-ensemble du système géré ? certains éléments défaillants ? Il s'agit ici d'agir sur les flux de collecte.
- La « **granularité de la surveillance** » correspond au « degré de détail » de la surveillance : Veut-on une vision plutôt globale, générale ou véritablement précise ? La granularité est fortement liée à la « finesse » qui permet de déterminer à quel degré cette vision est correcte et reflète la réalité.
- Enfin, la « **qualité de l'information de surveillance** » est aussi importante : elle correspond à un ensemble de critères permettant de déterminer si les données recueillies permettent d'obtenir une vision correcte de l'état du système. Par exemple :
  - la « fréquence temporelle » utilisée pour surveiller les éléments gérés : si la collecte se fait toutes les minutes, la donnée sera moins fraîche que si elle est opérée toutes les dix secondes,
  - la « finesse d'une donnée » permet de déterminer à quel degré cette donnée est correcte et reflète la réalité,
  - tandis que la « précision d'une donnée » permet de déterminer le degré d'exactitude de l'ensemble des mesures opérées,
  - enfin la « granularité de l'information » correspond quant à elle au degré de détail qu'il est possible d'appliquer pour mesurer une donnée.

Si le système géré fonctionne correctement, pourquoi chercher à avoir une granularité fine ? Il suffit de surveiller le système dans sa globalité. Mais lorsqu'une panne survient, il est préférable de réduire la portée et augmenter la finesse pour mieux détecter l'origine d'une panne. Un exemple d'aide au diagnostic peut éclaircir ce fait : imaginons que l'état d'une ressource soit surveillée de sorte à en offrir une vision « grossière » ou tout au moins globale. Puis un dysfonctionnement est détecté sur un système dans lequel est impliquée cette ressource en tant que composant : elle peut être une cause du problème. Une méthode d'identification de diagnostic peut se baser sur une observation plus fine du comportement de cette ressource. Pour cela, on souhaite pouvoir changer de granularité en collectant des informations d'état et de comportement de plus faible granularité.

Il apparaît que la qualité de l'analyse dépend fortement de la qualité des données collectées : les données vouées à être analysées se doivent de répondre à des contraintes de précision, de fraîcheur (voire parfois d'instantanéité), de pertinence (les informations utiles), de granularité. Comment la surveillance peut-elle savoir que les données collectées respectent ces contraintes ?

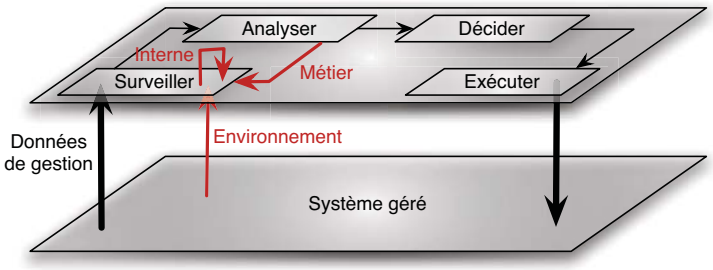

FIGURE 1.3: Les interactions avec le module de surveillance

Les interactions de la figure 1.3 nous apportent une partie des éléments nécessaires pour

déterminer s'il faut adapter la surveillance. Dans quels cas a-t-on besoin d'adapter la surveillance ?
Trois types de variations peuvent nécessiter un ajustement du comportement de la surveillance :

- une **variation des objectifs de l'analyse** : des **besoins métiers** ou objectifs de haut niveau peuvent être intégrés dynamiquement au module d'analyse, et la surveillance doit donc en être informée et être ajustée pour toujours répondre aux objectifs de l'analyse, quels qu'ils soient,
- une **variation des contraintes opérationnelles** : une entité du système géré est en panne et il faut réduire la portée de la surveillance pour détecter l'origine de la panne, ou encore surveiller de nouvelles entités dans le système...
- un besoin de **« *self-optimization* »** (autrement dit d'optimiser la surveillance) : les données collectées par la surveillance répondent aux deux premières contraintes de variations, mais elle est tellement intrusive dans le système géré que celui-ci ne fonctionne pas aussi bien qu'il le pourrait : la surveillance doit être optimisée pour être efficace mais sans perturber le bon fonctionnement du système sous-jacent.

Ces trois types de contraintes peuvent correspondre à un ensemble de bornes positionnées pour déterminer le domaine d'action de la surveillance. Par exemple, la fraîcheur des données récoltées par la surveillance doit être de moins de 10 ms (il s'agit d'une quantification de la qualité de l'information) ; la consommation de la surveillance ne doit pas dépasser 10 requêtes/ms (il s'agit ici de répondre à un besoin de qualité de service) ; enfin, un objectif métier pourrait être d'obliger la surveillance à interroger tous les équipements du système ou du réseau dans sa globalité.

Ainsi, il devient logique de mettre en place des actions pour permettre de modifier le comportement de la surveillance pour respecter ces trois types de variations de contraintes. Le but est donc de rendre la surveillance adaptative. Nous avons défini cette surveillance adaptative telle que :

Définition

La « surveillance adaptative » correspond à la capacité dont dispose une fonction de surveillance de décider et d'exécuter, sans interruption, l'ajustement de son comportement pour maintenir son efficacité dans le respect des exigences fonctionnelles et des contraintes opérationnelles mais éventuellement aussi, pour améliorer son efficience selon des préoccupations d'auto-optimisation.

1.2.2 Automatiser pour adapter la surveillance

Cet ajustement du comportement de la surveillance correspond à ce que l'on appelle l'« adaptation de la surveillance » en fonction des variations de contraintes opérationnelles, aux variations d'objectifs de l'analyse, et à des besoins éventuels d'auto-optimisation. Des actions d'adaptation sont nécessaires : elles sont généralement réalisées par un opérateur humain (en modifiant la configuration de l'outil utilisé ou en modifiant et/ou relançant l'exécution de divers scripts), ce qui engendre un coût supplémentaire venant se greffer au coût de base de la surveillance, qui peut alors devenir non négligeable lorsque le système géré est particulièrement étendu et complexe. De même, pour un contexte embarqué, des algorithmes doivent être réécrits pour réaliser ces actions d'adaptation, ce qui implique une augmentation du coût et du temps d'adaptation. Pour réduire ce coût et rendre cette adaptation performante, un fort besoin d'automatisation devient nécessaire.

Les deux concepts d'« adaptation » et d'« automatisation » posent simplement les bases de la problématique générale de ce manuscrit :

Problématique

Comment adapter dynamiquement la surveillance dans un contexte de gestion intégrée et dans un souci de prise en compte des trois types de variations préalablement établis ?

Il est donc nécessaire de faire un lien entre les actions d'adaptation à réaliser et les propriétés que devrait avoir une solution visant à supporter l'adaptation de la surveillance dans un cadre de gestion intégrée. Les chapitres suivants ont donc pour fonction :

- de déterminer d'une part quelles doivent être les propriétés nécessaires pour répondre à la problématique, et d'autre part de vérifier si les approches existantes y répondent totalement, partiellement ou pas du tout (voir le chapitre 2),
- de proposer et décrire une solution globale permettant de répondre totalement aux propriétés qui ont été définies et à la problématique du manuscrit (voir le chapitre 3).

Adapter la surveillance : état de l'art

2

Le but de ce chapitre est de définir les différentes propriétés auxquelles doit répondre une solution de surveillance adaptative, et de dresser un état de l'art des divers moyens pour surveiller un système mais aussi pour évoluer vers des solutions de surveillance adaptative.

2.1 Comment surveiller ?

Selon [Dilman01] la surveillance a deux finalités :

- La « surveillance statistique » est simplement utilisée comme observation. Elle permet de visualiser l'état du système géré, à un instant t, afin de vérifier son comportement.
- La « surveillance réactive » est plutôt utilisée dans un objectif préventif ou correctif. Tout symptôme, détecté sous la forme d'un comportement anormal et pouvant être à l'origine d'un dysfonctionnement de tout ou partie du système, doit être identifié pour aboutir à une réparation en temps réel, ou tout au moins le plus rapidement possible.

Quelle que soit cette finalité, toute observation ou détection d'anomalies passe par la collecte d'information de gestion. Cette information, qui est composée de données concrètes, représentant, par exemple, l'état d'une entité du système ou une quantification des paquets échangés, est transmise au *manager* de la station de gestion pour être traitée (analyse, décision et exécution).

Deux méthodes de collecte, illustrées sur la figure 2.1, permettent d'obtenir de l'information de gestion [Samaan09] :

- La « **collecte active** », ou ***polling***, permet au *manager* de la station de gestion de consulter périodiquement les données des agents. Cette consultation est généralement opérée de manière régulière, que le système fonctionne ou non correctement, et augmente inévitablement le trafic réseau. Mais, la vision de l'état du système est actualisée et il est possible d'obtenir des statistiques précises sur le comportement du système (ce qui est utile pour déterminer s'il répond bien à toutes les exigences de qualité de service imposées).
- La « **collecte passive** », ou ***event reporting***, permet au *manager* de la station de gestion de recevoir des notifications d'événements en provenance des agents disséminés sur les entités du système géré. La station de gestion reçoit ainsi des notifications uniquement lorsque les agents

détectent une anomalie ou une panne, un incident, ou une dérive quelconque sur le système surveillé, de sorte que la collecte passive permet de ne pas surcharger le trafic par l'envoi massif de données de surveillance.

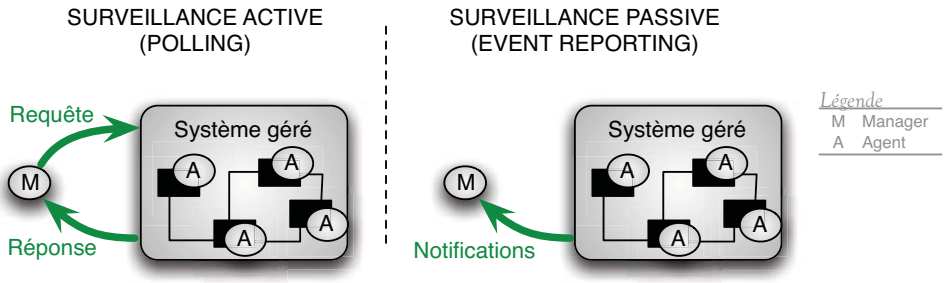

FIGURE 2.1: Méthodes de collecte d'information de gestion

Les deux mécanismes de surveillance peuvent être utilisés de manière disjointe ou non, dans un souci de performance, comme indiqué dans [Durai11] ou [Yang09]. Mais pour [Jiao00], les deux méthodes de collecte peuvent être utilisées conjointement, dans une tentative d'optimisation de leur utilisation : le but est d'avoir la meilleure vision possible de l'état du système en limitant le plus possible la quantité de données de surveillance échangée. A titre d'exemple, dans un but de non intrusivité, il peut être défini que les opérations de supervision ne doivent pas dépasser 10% de l'utilisation totale des ressources [Lahmadi08]. Dans un tel contexte, les agents informent le *manager* par notifications de tout dysfonctionnement détecté sur le système, et le *polling* est, quant à lui, utilisé de manière plus ciblée, pour engendrer le moins de trafic possible.

2.2 Propriétés d'une surveillance adaptative

Après une étude bibliographique, nous avons défini un ensemble de propriétés pour répondre à la problématique de l'automatisation de la surveillance, de sorte à répondre à un ensemble de **besoins liés au fonctionnel, à l'ouverture, à l'utilisabilité et à l'évolutivité logicielle**.

- L'adaptation doit être **automatisée** : ceci implique une nécessité de rendre l'adaptation **dynamique** et **non interruptive**, de sorte que l'adaptation ne perturbe pas le fonctionnement normal du système dans sa globalité.
- Etant positionné dans un contexte de gestion intégrée, il apparaît évident de continuer à conserver une propriété d'**ouverture**. Ainsi, il faut tenter d'offrir une solution **intégrée** s'adaptant parfaitement au contexte de la gestion intégrée, mais aussi **générique** de sorte qu'elle puisse être utilisée dans tout système quel qu'il soit.
- L'**utilisabilité** de la solution est également importante : si elle ne peut pas être **réutilisée** au sein d'autres systèmes, son utilité en est fortement réduite.
- L'**évolutivité logicielle** de la solution enfin est importante : elle doit être **modulaire** de sorte à pouvoir utiliser tout ou partie du logiciel, mais aussi **expressive** pour gérer l'adaptation à tous les niveaux d'abstraction. Ainsi, il est impératif que la solution soit ***well-defined***.

La solution finale doit donc pouvoir répondre aux propriétés répertoriées dans le tableau 2.1.

INTEGREE	La solution doit être spécifiée et implémentée dans un contexte de gestion intégrée pour répondre à un besoin d'unification.
GENERIQUE	La solution ne doit pas être dépendante d'une architecture, d'une technologie ou d'un protocole particulier.
WELL-DEFINED	La solution doit présenter une interface correctement définie et spécifiée, permettant d'éviter des ambiguïtés d'interprétation.
MODULAIRE	La solution doit être implémentée au moyen de briques pouvant être interchangées et spécialisées si besoin.
REUTILISABLE	La solution, ou une partie de la solution, doit pouvoir être réutilisée en l'état ou spécialisée.
DYNAMIQUE	La solution doit permettre de réaliser une adaptation automatisée et en cours d'exécution de la surveillance.
NON INTERRUPTIVE	La solution doit permettre d'adapter la surveillance sans qu'elle ou le système géré ne soit interrompus.
EXPRESSIVE	La solution doit permettre d'exprimer une adaptation à tous les niveaux d'abstraction que son interface a définis.

TABLE 2.1: Propriétés d'une solution de surveillance adaptative

Il convient désormais de dresser un état de l'art des solutions existantes et communément employées pour surveiller un système, de sorte à vérifier si l'une d'elle répond à l'ensemble de nos propriétés.

2.3 Etat de l'art

Les solutions analysées relèvent majoritairement de la gestion de réseaux, mais peuvent aussi concerner la gestion de système. Ainsi, il sera indiqué clairement si une des approches permet d'être étendue à de la gestion intégrée (propriété 1).

2.3.1 Collecte de données de surveillance

La collecte de données de surveillance est généralement réalisée de manière statique et conformément à l'une des quatre propriétés suivantes : la portée, l'aspect temporel, la granularité et la distribution.

2.3.1.1 Distribution de la surveillance

Comme cela est illustré sur la figure 2.2, la surveillance peut être soit centralisée, soit distribuée [Samaan09] :

- La « surveillance centralisée » (à gauche) consiste à confier les fonctionnalités de surveillance à un seul et unique *manager*, positionné sur la station de gestion du système. Ce *manager* est en charge d'effectuer toute demande de consultation nécessaire (*polling*) et de recevoir toute notification (*event reporting*) envoyée par les agents. Globalement, un seul *manager* traite l'ensemble de l'information de gestion.
- La « surveillance distribuée » (à droite) consiste à confier tout ou partie des fonctionnalités de surveillance à plusieurs *managers*. Chaque *manager* a la charge d'un segment du système, qu'il

doit surveiller au moyen de sondes. Les *managers* peuvent interagir entre eux pour obtenir une vision globale du réseau.

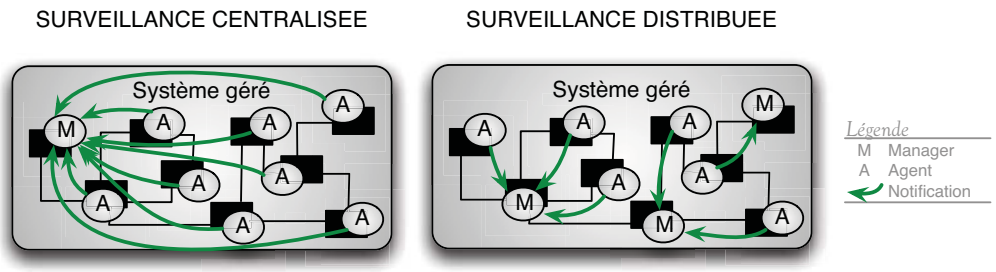

FIGURE 2.2: Distribution de la surveillance

Quelle méthode de distribution est la plus utilisée ? Les systèmes étant de plus en plus importants, les données réceptionnées par un seul et unique *manager* peuvent être faussées et refléter un état du système qui a, entre temps, déjà été modifié (manque de fraîcheur). Le temps de réception engendre un réel risque de décalage entre l'état réel et la représentation de l'état du système. La tendance actuelle est donc de privilégier la surveillance distribuée, de sorte à minimiser cet écart entre l'état réel et son reflet. Dans un premier temps, des petits segments du système sont surveillés indépendamment les uns des autres ; dans un second temps, l'état global est reconstitué par composition des états de tous les segments constitutifs du système.

Sont présentées à la suite plusieurs architectures basées sur cette approche de distribution.

2.3.1.1.1 Les MIBs à la demande

Chaparadza et al. [Chaparadza06] présentent le concept de surveillance à la demande, basé sur cette idée de la distribution de la surveillance. Des sondes positionnées sur le réseau reçoivent des notifications et peuvent décider de collecter des informations de surveillance dans des MIBs à la demande (*on-demand MIBs*). Ces MIBs, qui ont une durée de vie prédéfinie par un TTL (*Time To Live*), permettent de stocker temporairement de l'information. Lorsque cette information n'est plus utile, la MIB est simplement détruite. Ainsi, chaque sonde peut visualiser l'état de son segment du réseau. Si une situation apparaît comme problématique, elle informe par notification la station de gestion, comme le montre la figure 2.3.

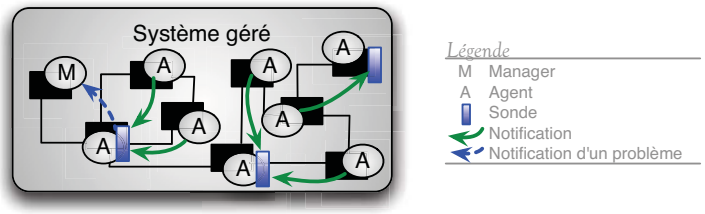

FIGURE 2.3: La surveillance à la demande

Cette approche tente de minimiser l'utilisation de toutes les ressources :

- le *manager* ne recevant que des notifications concernant un comportement jugé problématique, la bande passante est économisée et les ressources de la station de gestion sont

utilisées au minimum (traitement simple sans filtrage d'information),

- les ressources sur lesquelles sont positionnées les sondes voient également leur consommation réduite du fait d'une mise à jour ponctuelle de l'état (quand cela est nécessaire uniquement), et de la libération des MIBs à la demande par destruction lorsque la surveillance n'est plus nécessaire.

Cette approche est réutilisable, dans le sens où il suffit d'installer les sondes sur le réseau que l'on souhaite surveiller. De plus, l'utilisation de ces sondes est dynamique et non interruptible car sur réception d'une notification, elles créent une MIB pouvant être interrogée à tout moment par la station de gestion du réseau.

2.3.1.1.2 Le système A-GAP

Prieto et al. [Prieto07] tendent à rendre une vision axée sur les processus de gestion. Chaque processus de gestion devient un nœud dans un arbre d'agrégats : ce nœud contient les informations de surveillance locale et un agrégat des informations de surveillance des nœuds descendants (une somme, une moyenne, la valeur minimum, la valeur maximum). Les calculs propres à la surveillance du réseau sont ainsi distribués. La figure 2.4 représente un exemple d'arbre d'agrégation A-GAP portant sur la fonction somme (`SUM`).

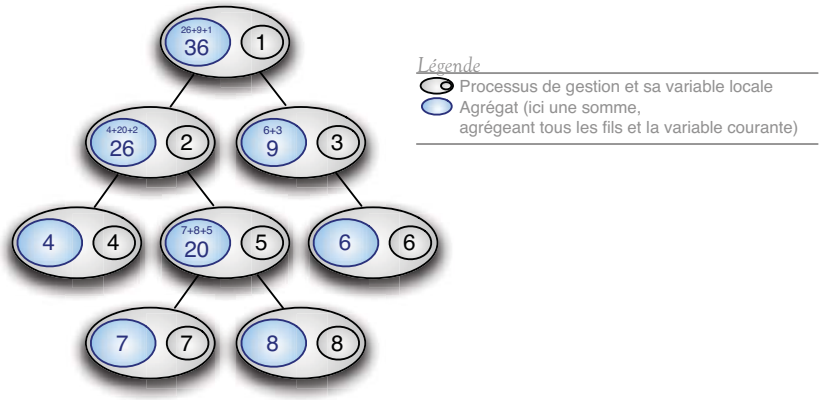

FIGURE 2.4: Agrégation A-GAP de type somme

Chaque donnée contenue dans les nœuds de l'arbre peut être mise à jour sur réception d'un événement (détection d'un nouveau voisin, panne d'un voisin, simple mise à jour d'une variable de gestion).

Cette approche permet de visualiser les processus du réseau, mais présente dans le même temps l'inconvénient de les figer : une fois qu'ils ont été définis et positionnés dans l'arbre d'agrégats, ils ne peuvent plus être modifiés sans reconstruire entièrement l'arbre. Il faut donc essentiellement retenir ici comme avantage la distribution des calculs portant sur la surveillance, la flexibilité de l'approche, et la possibilité de passage à l'échelle du réseau (l'arbre d'agrégat peut en effet être complètement indépendant de l'architecture du système : sa vision est conceptuelle).

Les avantages principaux de cette approche sont la généricité et la modularité. L'arbre d'agrégation permet de prendre en compte tout processus métier, quel qu'il soit.

2.3.1.1.3 *Do You See What I See ?*

Miao et al. [Miao07] offrent la possibilité de surveiller le réseau de manière distribuée en interconnectant un ensemble de nœuds DYSWIS (*Do You See What I See ?*) en mode P2P (*peer to peer*). Sur chacun de ces nœuds sont positionnés des agents de surveillance chargés de détecter des pannes d'éléments (et de les historiser), et de communiquer toutes les informations concernant la panne avec tout autre nœud DYSWIS du système, comme l'illustre la figure 2.5. Ajouté à cela, en cas de dysfonctionnement du réseau, chaque nœud a un comportement spécifique déterminé par un ensemble de règles qui lui sont propres. Ces règles permettent de déterminer si le nœud doit déclencher une requête vers les autres nœuds DYSWIS pour obtenir plus de détails sur la panne, ou pour savoir si un autre nœud, ou lui-même, doit se charger de la panne, ou s'il doit attendre d'autres notifications.

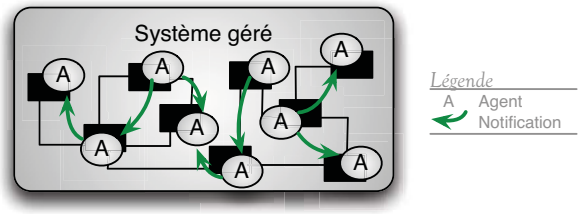

FIGURE 2.5: Comportement des nœuds DYSWIS

La station de gestion étant inexistante dans cette architecture, celle-ci offre l'avantage de réellement distribuer la surveillance et tend vers l'adaptabilité de la surveillance grâce à l'introduction de règles permettant de décider quel comportement les nœuds doivent adopter selon la panne détectée (il est par ailleurs à noter que cette approche restreint la surveillance à la seule détection de pannes et n'inclut aucune mesure d'efficacité ou de performance).

Les nœuds DYSWIS permettent donc de surveiller le réseau de manière dynamique et non interruptible, grâce à l'introduction des règles positionnées sur chacun des nœuds. Chacun des agents peut être réutilisés sur un autre réseau P2P en fonction des besoins, et avec les règles qui lui correspondent.

2.3.1.1.4 AutoMon

Le projet AutoMon [Binzenhofer06] est basé sur des principes semblables : ici encore, différents agents sont distribués sur les *peers* d'un réseau P2P, afin que chaque nœud puisse réaliser des traitements concernant les informations de surveillance collectées. Notamment, tous les agents réalisent, indépendamment les uns des autres, des tests concernant la gestion des pannes détectées sur leur segment de réseau (détection de pannes et tests de fonctionnement). La différence avec l'approche DYSWIS réside en l'existence d'un moniteur central (assimilable à une station de gestion) chargé, après réception des rapports de pannes, de coordonner les dépannages en envoyant ses instructions aux agents distribués (figure 2.6).

Un moniteur central étant chargé de coordonner les dépannages, cette approche distribue de manière plus limitée la surveillance que la précédente. Toutefois, cette approche ne peut être considérée comme centralisée, car les données réceptionnées par le moniteur sont issues de différents agents chargés de tester leur segment particulier de réseau. Il y a eu pré-traitement, afin de

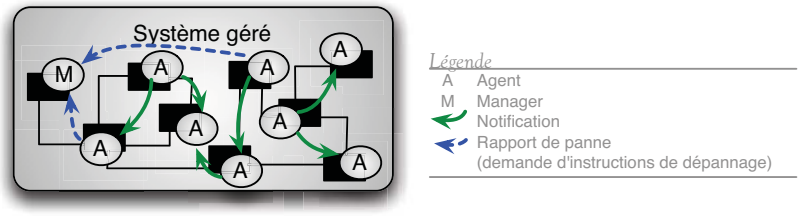

FIGURE 2.6: Comportement des nœuds AutoMon

rendre les données plus précises avant demande de dépannage. Les actions de surveillance restent dynamiques et non interruptibles.

2.3.1.1.5 Ganglia

Une autre perspective est proposée par [Massie04]. Chacun des nœuds a ici la possibilité de surveiller l'ensemble des ressources qui lui sont assignées. Si une situation importante ou significative d'un problème lui est notifiée, le nœud envoie en *multicast* à tous les autres nœuds toutes ses données de surveillance de manière dynamique et non interruptible. De plus, chaque nœud possède sa propre interface programmable afin qu'il puisse reconfigurer ses paramètres et/ou mécanismes de surveillance, comme cela est illustré sur la figure 2.7.

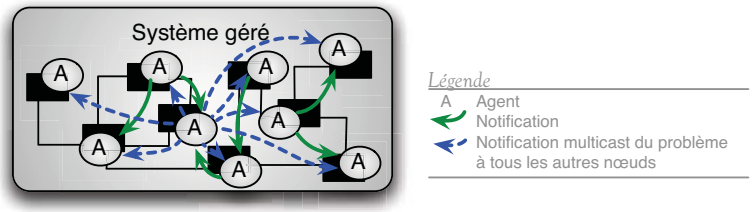

FIGURE 2.7: L'approche Ganglia

Des exemples de reconfiguration de la surveillance peuvent être cités :

- fréquence d'interrogations par *polling*,
- seuils positionnés sur les variables mesurées,
- arrêt d'utilisation du *polling* au profit des seules notifications d'événements,
- possibilité de surveiller plus ou moins finement les éléments gérés,
- possibilité de forcer la réalisation d'une occurrence de surveillance de type *polling* sur les différents nœuds pour vérifier le (dys)fonctionnement d'un équipement ou encore obtenir des statistiques supplémentaires sur une situation particulière survenue à un moment précis.

Ce système est réellement distribué et permet une reconfigurabilité de la surveillance intéressante, de par la diversité des fonctionnalités qu'il propose : bien défini, il est également modulaire et permet d'être réutilisé sur divers réseaux. Toutefois, cette décision de reconfiguration n'est pas automatisée et nécessite toujours la présence d'un opérateur humain.

2.3.1.2 Portée de la surveillance

La plupart des approches proposées par la communauté des chercheurs portent sur la distribution des agents de surveillance sur différents nœuds. Chacun de ces agents doit surveiller un ensemble de ressources qui lui sont locales, puis les traiter (envoi de notifications à d'autres agents ou à un ou plusieurs *managers*, reconfiguration locale de la surveillance, attente d'autres notifications...). Dans un tel contexte, il apparaît important de pouvoir définir la portée de chacun des agents sur le réseau, afin que ce dernier soit pris en compte dans sa globalité dans le processus de surveillance.

Il a notamment été démontré dans [Cantieni06] que, selon l'état du réseau, il peut être intéressant de surveiller plus en détail un segment bien particulier, en délégant à un nœud (ou plusieurs) la surveillance plus pointue de cette partie du réseau ; de même, à l'inverse, cet affinage de la surveillance peut être supprimé si l'état du réseau ne le nécessite pas. Pour ce faire, leur algorithme permet d'optimiser dynamiquement le placement des surveillants. Cet algorithme est basé sur les fonctions d'utilité : celui-ci prend en entrée la topologie du réseau et son état à un moment t, puis renvoie en sortie les nœuds qui sont les mieux placés pour effectuer la surveillance, ainsi que leur taux d'échantillonnage.

Grâce à l'utilisation de cet algorithme, les mesures de surveillance sont précises et la consommation des ressources est minimisée. Toutefois, bien qu'il puisse être réutilisé, l'algorithme d'optimisation n'intègre pas encore le traitement des objectifs métiers et l'analyse de la performance du réseau.

2.3.1.3 Aspect temporel de la surveillance

La plupart des entités chargées de surveiller un réseau réalisent leurs opérations de surveillance sur des intervalles de temps prédéfinis et non modifiables [Samaan09]. Il apparaît ainsi réellement intéressant de faire varier dynamiquement les intervalles de temps durant lesquels doivent être réalisées les opérations de surveillance. De plus, s'il était possible de réaliser des opérations à la demande plutôt que de les réaliser de manière périodique (cas du *polling*), il serait possible de réduire le trafic et la consommation des ressources.

2.3.1.3.1 Absence de temporalité

Pour mieux comprendre l'intérêt de pouvoir faire varier l'aspect temporel, il faut reprendre deux des approches précédemment évoquées : la surveillance à la demande [Chaparadza06] et le système Ganglia [Massie04] ne prennent pas en compte cet aspect. Elles sont basées sur l'occurrence asynchrone d'événements et ne déclenchent des opérations de surveillance que lorsque c'est nécessaire. Bien que ces approches passives permettent de gérer la consommation des ressources de manière optimale, deux inconvénients majeurs apparaissent :

- Il est impossible d'avoir une vision toujours actualisée du réseau, car les notifications ne sont émises que lorsqu'une situation particulière est détectée. En outre, si aucune notification n'est reçue, il peut être supposé que tout fonctionne correctement, alors qu'un composant du réseau est tombé et n'est donc pas en mesure d'émettre sa notification. Une consultation régulière par *polling* aurait, elle, révélé ce genre d'anomalie.
- Les statistiques sont inexactes car les seules informations de surveillance obtenues sont issues de situations problématiques ou particulières. Si le système fonctionne normalement, aucune

donnée de surveillance ne remontera, et nous n'aurons aucune visibilité sur le système.

2.3.1.3.2 Variations de temporalité

Moghé et al. [Moghe98] ont défini RAP (*Rate Adaptive Polling*), un algorithme permettant d'adapter dynamiquement le nombre de *polling* en cours d'exécution en fonction du taux d'utilisation du réseau et des ressources par les utilisateurs. Lorsque le trafic est trop important, les intervalles de temps entre deux demandes de consultation (*polling*) sont augmentés, de manière à réduire l'engorgement du réseau. *A contrario*, lorsque le réseau ne présente que peu de trafic, les intervalles de temps entre deux demandes de consultation sont réduits pour obtenir une vision plus fine de son état.

Cet algorithme ne réduit pas la performance du réseau car il permet de réduire l'impact de la surveillance sur le fonctionnement du réseau et offre une vision relativement fine de son état, lorsqu'il le peut.

2.3.1.4 Granularité de la surveillance

Selon [Samaan09], il est possible de faire varier différents aspects temporels, comme les temps de retard *maxima*, le taux d'échantillonnage des données (de quelques secondes à plusieurs heures) ou encore de réaliser des collectes de données sur différents niveaux de granularité.

2.3.1.4.1 Améliorer la qualité de l'information

Les deux approches présentées ici relèvent de la gestion de service.

Roxburgh et al. [Roxburgh11] adoptent une approche pragmatique basée sur la qualité de service pour offrir aux applications de gestion une interface orientée service. Leur service de surveillance leur permet de contrôler dynamiquement et de passer à l'échelle le déploiement de l'activité de surveillance intégrée. La surveillance est basée sur deux produits commerciaux, et de ce fait, pour gérer son intégration en tant que service, des adaptateurs *ad hoc* doivent être implémentés : aucune information de gestion commune n'a été définie pour supporter leur viabilité de manière unifiée.

Le Duc et al. [LeDuc10] présentent ADAMO, une solution de surveillance adaptative, inspirée de [Abid09], permettant « de récupérer différentes requêtes de données respectant une qualité d'information dans des flux de données dynamiques, de les transformer en paramètres de configuration sondés en fonction des contraintes sur les ressources ». Une approche à résolution de contraintes a été choisie pour la prise de décision d'adaptation, et particulièrement pour le calcul de la valeur de la fréquence de demandes sur la source de données. Par conséquent, la modélisation d'adaptation proposée est fortement influencée par les problèmes de satisfaction de contraintes. Aucun modèle d'information de gestion de granularité correcte n'a été clarifié.

2.3.1.5 Configurer globalement la surveillance

Un outil de supervision permet de (re)configurer dynamiquement la surveillance en jouant sur chacun des aspects précédemment cités : l'outil Nagios [Nagios]. Ce serveur, intégré à un système, permet de superviser un système dans son ensemble, tout en apportant les fonctionnalités suivantes :

- une interface graphique permettant d'obtenir une vue globale de l'état du système,

- un contrôle des états opérationnels des éléments du système :
  - le contrôle actif (*polling*) permet de réaliser des requêtes ponctuelles sur les équipements du système pour obtenir des détails particuliers,
  - le contrôle passif (*event reporting*) permet la réception de notifications prévenant de tout événement pertinent qu'il est important de mettre en évidence,
- la (re)configuration en cours d'exécution des contrôles actif et passif (cette « auto-adaptation » reste limitée à quelques paramètres si le logiciel n'a pas été spécialisé),
- une intégration de bonnes pratiques et de procédures utilisateurs pour la surveillance du système,
- une aide au diagnostic permettant de mettre à jour la cause d'une panne plus facilement et plus rapidement.

Une amorce a été engagée par les créateurs de Nagios pour permettre une certaine adaptabilité de la surveillance au niveau du logiciel en laissant le soin aux administrateurs de modifier certains paramètres de configuration des mécanismes de surveillance afin d'optimiser cette dernière en fonction des données collectées observées. Il est donc légitime de s'interroger sur la possibilité de généraliser cette adaptabilité à l'ensemble des paramètres de configuration de Nagios, tout en sachant que la configuration de la surveillance est contenue dans des fichiers texte.

Nagios est donc well-defined et modulaire, tout en permettant l'adaptation dynamique et sans interruption de la surveillance. Reste à améliorer l'expressivité de l'adaptation de la surveillance au niveau du logiciel, qui n'est utilisable que pour certains paramètres de configuration de la surveillance.

2.3.2 Traitement de l'information de surveillance

Les approches précédentes permettent de mettre en place la surveillance et d'adapter ponctuellement la configuration de la surveillance, sans prendre en compte le traitement effectué par les entités après collecte des informations de gestion. Deux cas de figures sont observables : soit les agents disséminés sur le réseau traitent directement l'information, soit ils relaient cette information vers le *manager* de la station de gestion qui est alors chargé de la traiter de lui-même.

2.3.2.1 La surveillance réflexe

La surveillance réflexe [Sterritt05] vise à utiliser un mécanisme de surveillance distribué essentiellement basé sur les événements et qui s'inspire de réactions naturelles : les réflexes. Un exemple très simple : si un homme pose sa main une plaque de cuisson brûlante, il la retire sans réfléchir. Ce mécanisme de réflexe est ici reproduit au niveau des agents : lorsqu'un événement particulier est détecté par un agent, celui-ci a le réflexe d'avertir immédiatement son *manager* (par le biais d'une notification) de l'événement en question, tout en lui incluant dans le message un résumé de ses indicateurs d'état. Cette action est dynamique et non interruptible.

Cette approche basée sur les réflexes vise donc à accélérer la propagation de l'information, mais peut toutefois augmenter de manière considérable le nombre d'informations échangées entre les agents et leur *manager* car les événements ne sont pas agrégés mais envoyés tels quels. Il est donc essentiel de définir correctement la granularité des informations collectées, afin de réduire la quantité de données échangées et ainsi améliorer la performance du réseau en ce qui concerne

l'utilisation des ressources.

Il faut tout de même noter que ce mécanisme ne vise qu'à avertir le *manager* des événements qui se sont produits sur le réseau. C'est ensuite au *manager* de traiter l'information, par le biais d'un algorithme de surveillance tel que défini ci-après.

2.3.2.2 L'agorithme de surveillance

Un algorithme de surveillance [Jiao00] permet de traiter les données collectées efficacement et en temps réel. Il prend en compte :

- Les valeurs collectées dans le passé, ce qui lui permet d'effectuer une comparaison précise entre l'état actuel et les états passés. Il est alors possible d'évaluer si la surveillance du réseau est plus, autant ou moins efficace qu'auparavant, en termes de temps de réponse, de débit ou d'utilisation des ressources [Lahmadi08] : en cas de perte d'efficacité, des réajustements sont nécessaires.
- Les contraintes de variations ont été positionnées sur les paramètres de surveillance. Ceci permet de vérifier que les données observées par la surveillance respectent bien les seuils et bornes spécifiés : si ce n'est pas le cas, un réajustement est nécessaire.

Un algorithme de surveillance qui peut détecter et prendre en compte dynamiquement toutes les conditions d'alarme est dit « correct ». Par ailleurs, cet algorithme devient « optimal » lorsque, en plus, il n'engendre pas un surcoût de surveillance excessif. Evidemment, il faut tendre à rendre un algorithme de surveillance le plus optimal et performant possible, en minimisant le surcoût de la surveillance.

L'algorithme de surveillance est utilisé conjointement avec un ensemble de règles de surveillance, généralement des politiques, visant à déterminer le traitement à réaliser le plus approprié à l'état actuel du réseau.

2.3.2.3 Les règles de surveillance

Les règles de surveillance utilisées par les algorithmes sus-cités sont généralement des politiques. En effet, selon [Han04], les politiques peuvent être utilisées pour paramétrer la configuration de la surveillance basées sur l'état du réseau et des applications en cours d'exécution : dans ce cas, les politiques sont activées lors de l'exécution pour ajuster divers paramètres, comme le taux d'échantillonnage, ou encore des intervalles de temps entre deux demandes de consultation.

Si l'ajustement à réaliser est plus important qu'une simple modification d'un paramètre donné, les politiques peuvent sélectionner un algorithme de surveillance complet et approprié à l'état courant du réseau afin d'ajuster de manière plus approfondie la surveillance en cours d'exécution. Ceci implique alors de mettre en place un moyen d'identification et de choix d'algorithmes.

De manière générale, la gestion à base de politiques représente un très bon moyen de piloter l'adaptation de la surveillance dans le sens où chaque objectif de gestion peut être spécifié et traduit en un format de politique. Dans les domaines de la gestion de réseaux et de systèmes, de nombreux travaux traitent de la surveillance adaptative dans sa globalité et proposent des solutions basées sur les règles de surveillance et politiques. Les paragraphes suivants font état de ces solutions.

2.3.2.3.1 QMON

Un exemple d'utilisation des politiques est l'approche de surveillance QMON [Agarwala06]. Cette approche, centrée sur la qualité des services rendus par le réseau surveillé (QoS), utilise un ensemble de politiques prédéfinies pour assurer l'adaptation de la surveillance à l'état du réseau et aux objectifs de l'entreprise, lorsqu'il est en cours d'exécution. Cette approche présente plusieurs avantages. En effet, QMON est :

- flexible : la plupart des paramètres de surveillance peuvent être modifiés,
- configurable : le comportement de la surveillance est adapté en temps réel aux besoins de gestion définis initialement,
- robuste : la surveillance ne provoque pas de perturbations sur le réseau pendant qu'il rend les services attendus,
- optimale : l'efficacité de la surveillance tend à être maximisée en permanence.

Toutefois, même si elle est dynamique, réutilisable et *well-defined*, QMON n'est pas une approche générique et reste dédiée aux réseaux.

2.3.2.3.2 ANEMONA

Le *framework* ANEMONA [Duarte08] propose un langage permettant de spécifier et déployer des politiques pour la surveillance d'un réseau en utilisant pour celle-ci le standard SNMP dans sa troisième version. Le langage utilise deux types de MIBs SNMP (*Event* et *Expression*) pour collecter l'information, calculer/évaluer des expressions (conditions) et stocker les actions à exécuter sur évaluation des conditions. Les objets des MIBs sont donc modifiés dynamiquement de sorte à avoir l'information la plus fraîche possible. De plus, ANEMONA est expressive dans le sens où elle apporte une langage et une programmabilité intéressante pour adapter la surveillance.

ANEMONA est une approche intéressante basée sur les politiques et les événements permettant de programmer des applications de surveillance de réseau configurable tout en étant simple d'utilisation (l'approche ne nécessite qu'une connaissance relative des MIBs et commandes SNMP), mais ne supporte pas réellement le contrôle dynamique de l'adaptation de la surveillance. De plus, l'approche est exclusivement dédiée à un environnement SNMP, ce qui ne permet pas d'utiliser d'autres protocoles de collecte de données, comme cela est pourtant requis en gestion intégrée.

2.3.2.3.3 Des politiques pour configurer une infrastructure de surveillance

Ouda et al. [Ouda10] étudient la possibilité, quand une approche de gestion à base de politiques est adoptée, d'extraire d'un ensemble de politiques toute l'information utile à la configuration de l'infrastructure de surveillance sous-jacente. L'initialisation et la reconfiguration dynamique de leurs agents de surveillance sont réalisées *via* un code commun : aucune interface précise et aucun paramètre de configuration ne sont explicitement mentionnés.

2.4 Bilan

Le tableau 2.2 dresse un récapitulatif de travaux et de solutions visant à rendre la surveillance adaptable.

Aucune ne correspond exactement à notre vision du problème. Tout d'abord, les solutions proposées ne sont pas assez génériques, ne fonctionnent qu'avec un protocole ou un type

	intégrée	*générique*	*well-defined*	*modulaire*	*réutilisable*	*dynamique*	*non interruptible*	*expressive*
MIBs à la demande	☹	☹	☹	☹	☺	☺	☺	☹
A-GAP	☹	☺	☹	☺	☺	☹	☹	😐
DYSWIS	☹	☹	☹	☹	☺	☺	☺	😐
AutoMon	☹	☹	☹	☹	☺	☺	☺	😐
Ganglia	☹	☹	☺	☺	☺	☺	☺	😐
Cantieni06	☹	☹	😐	☹	☺	☺	☺	☹
RAP	☹	☹	☹	☹	☺	☺	☺	☹
Roxburgh11	😐	😐	☺	😐	😐	☺	☺	😐
ADAMO	😐	😐	😐	😐	☺	☺	☺	😐
Nagios	☹	😐	☺	☺	☺	☺	☺	😐
Surveillance réflexe	☹	☹	☹	☹	☺	☺	☺	☹
Algo de surveillance	☹	😐	☺	☹	☺	☺	☺	😐
QMON	☹	☹	☺	☹	☺	☺	☺	☹
ANEMONA	☹	☹	😐	😐	☺	😐	☺	☺
Ouda10	☹	☹	☹	☹	☺	☺	☺	☹

TABLE 2.2: Comparaisons des outils/techniques de surveillance

d'environnement particulier. Ensuite, l'ensemble des solutions proposées reposent sur l'utilisation d'algorithmes ou de règles qui sont figés et non adaptables dans le temps. Or, les contraintes et objectifs peuvent évoluer. Avec de telles solutions, faudrait-il arrêter le système complètement, recoder les algorithmes et tout relancer ?

Enfin, elles ne visent que des problèmatiques de performance, sauf dans les deux cas suivants :

- L'approche visant à mettre en place des algorithmes de surveillance [Jiao00] est conceptuellement intéressante car elle vise à optimiser la surveillance pour la rendre la moins intrusive possible, grâce à l'ajout de règles de surveillance,
- QMON [Agarwala06] est une approche à base de politiques permettant de prendre en compte les besoins de gestion mais uniquement initialement. En cours d'exécution, il n'est pas possible de modifier les objectifs de gestion, car les politiques permettant de les implémenter ne sont pas adaptables. Or, les objectifs de gestion peuvent eux aussi évoluer.

Ainsi, nous nous sommes attachés à étudier et spécifier un cadriciel pour la mise en œuvre d'une surveillance adaptive de réseaux et de systèmes complexes, qui soit :

- indépendant de toute technologie, ou de toute solution de gestion particulière,
- non contraint à l'utilisation du seul protocole SNMP,
- capable de prendre en compte toutes les complexités ou difficultés liés à ce domaine d'application : hétérogénéité des composants, dépendences potentielles...

Enfin, la surveillance doit être appliquée dans un contexte de gestion intégrée : les approches proposées sont globalement appliquées dans un cadre de gestion de réseaux.

De sorte qu'il devient nécessaire de concevoir un *framework* prenant en compte l'ensemble des propriétés définies, pour pouvoir piloter l'adaptation de la surveillance de manière efficace.

Un *framework* pour piloter l'adaptation de la surveillance

3

L'objectif de ce chapitre est de dresser un cadre de surveillance adaptable à toute variation de contraintes opérationnelles, environnementales ou de performance. Un *framework* a été conceptualisé dans ce sens et fait l'objet de ce chapitre.

Pour répondre aux besoins d'adaptation de la surveillance, nous avons conduit une approche de type *bottom-up*. Notre analyse a révélé que pour influer sur la qualité de l'information, il faut agir sur la configuration des mécanismes de *polling* et *event reporting*, tandis que pour répondre à la variation des besoins métiers et adapter la performance de la surveillance, il faut agir sur l'ensemble des mécanismes en cours d'exécution (réduction ou augmentation de la portée).

Ainsi, il devient logique de mettre en place des actions pour permettre de modifier individuellement la configuration des mécanismes de surveillance, aussi bien que l'ensemble des mécanismes en cours d'utilisation, afin de jouer sur les différentes propriétés précédemment établies.

3.1 Une boucle de contrôle pour adapter la surveillance

La démarche proposée vise à créer un *framework* supportant une adaptation de l'activité de surveillance. Cette adaptation résulte en une modification de la manière de surveiller, ce qui implique une « gestion » de la surveillance elle-même. Pour ce faire, a été introduit un plan de contrôle de la surveillance, virtuellement représenté à partir d'une seconde boucle MAPE (visible en rouge sur la figure 3.1) pour l'adaptation de l'activité de surveillance adaptative. Chacune des données utilisées par les différentes activités des boucles MAPE sont stockées dans une base de connaissance commune. Chaque module peut interroger cette base de connaissance pour obtenir les données qui lui sont nécessaires à chaque traitement.

La seconde boucle de contrôle, présente dans la partie supérieure du schéma, représente une abstraction du fonctionnement interne du module de surveillance adaptative : dans cette approche, les mécanismes de surveillance et la collecte des informations de surveillance doivent ainsi pouvoir être ajustés selon l'état du système surveillé ou toute variation de contraintes, et ce en temps réel. La figure décompose le processus en trois étapes :

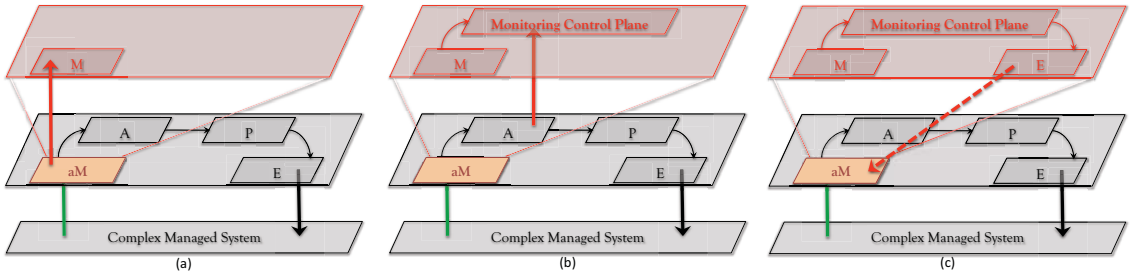

FIGURE 3.1: La surveillance adaptative

- sur l'étape (a) de la figure 3.1, le module de surveillance [M] de la surveillance adaptative [aM] récupère les données de gestion collectées issues du système géré lui-même,
- sur l'étape (b) de la figure 3.1, les résultats issus de l'analyse sont à leur tour pris en compte, tout comme l'ensemble des variations de contraintes fonctionnelles, environnementales ou d'optimisation ayant été dégagées précédemment. Ainsi, en combinant ces variations avec les données de gestion collectées en (a), le plan de contrôle de la surveillance adaptative [aM] peut déterminer quels mécanismes de surveillance doivent être modifiés afin d'ajuster la surveillance. Cette étape peut elle-même être décomposée en deux phases :
  - la phase d'analyse évalue l'ensemble des stratégies possibles d'adaptation de la surveillance,
  - la phase de décision permet ensuite de choisir quelle stratégie adopter pour ajuster la surveillance.

  Pour faire ce choix, le plan de contrôle de la surveillance a été pensé et défini en s'inspirant de la fonction *self-configuring* du concept de l'*Autonomic Computing* [Kephart03], et passe de ce fait par l'utilisation de règles de haut niveau.
- sur l'étape (c) de la figure 3.1, enfin, la stratégie d'adaptation de la surveillance est exécutée : elle applique un ensemble d'opérations de reconfigurations simples (modification d'une valeur seuil ou d'une période temporelle) ou plus complexes (modification du mode d'exécution ou de la portée de la surveillance).

En conclusion, à partir de la prise en compte des deux types d'entrées (règles ou objectifs de haut niveau d'une part, contraintes environnementales ou opérationnelles d'autre part), une analyse est déclenchée au niveau du plan de contrôle de la surveillance et une décision d'adaptation de la surveillance peut être prise de manière automatique. De ce fait, une adaptation du comportement de la surveillance peut être lancée pour améliorer l'efficacité de la surveillance, pour améliorer ses performances et pour ajuster la qualité des informations de gestion requises.

Pour être fonctionnelle, l'infrastructure du *framework* doit posséder trois capacités en langages et/ou outils pour mettre en place cette adaptation. Ces trois couches interconnectées – qui ont été spécifiées, conceptualisées et implémentées successivement – représentent les trois étapes de construction du *framework* adaptatif :

- la capacité à initier puis modifier dynamiquement sans interruption le comportement individuel d'un mécanisme de surveillance ((re)configuration des mécanismes de surveillance), à savoir les modalités opératoires d'un *polling* ou d'un *event reporting*,
- la capacité d'opérer l'adaptation d'une activité de surveillance en cours, grâce à la spécification d'opérateurs d'exécution de l'adaptation,

- la capacité **gouvernabilité**, à savoir de détecter un besoin d'adaptation et de déclencher cette adaptation (règles de déclenchement des opérateurs d'adaptation des mécanismes de surveillance).

Une analyse *bottom-up* a donc été employée pour spécifier et implémenter chacune de ces capacités, en partant du mécanisme de base, qui doit être rendu (re)configurable, puis en poursuivant au niveau de l'activité qui doit être rendue adaptable. Enfin, la gouvernabilité de l'adaptation de la surveillance peut être abordée.

3.2 Comment rendre la surveillance adaptative

Automatiser l'adaptation de la surveillance nécessite de gérer et contrôler l'activité de surveillance. Ceci mène à l'introduction d'un plan de contrôle visant à gérer l'ensemble des mécanismes sous-jacents utilisés pour surveiller effectivement le système géré. Le système contrôlé est l'activité de surveillance elle-même. Pour répondre à cette problématique, les trois capacités interdépendantes de la figure 3.2 ont été définies.

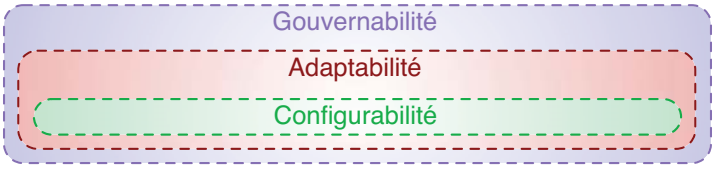

FIGURE 3.2: Couches du *framework*

3.2.1 Configurabilité

La « configurabilité », qui est la couche « basse » du *framework* proposé, est la capacité à ajuster initialement et en cours d'exécution le comportement des mécanismes de surveillance : pour ce faire, les mécanismes doivent être rendus configurables. Il devient nécessaire de décomposer la configuration de chacun des mécanismes de surveillance (*polling* et *event reporting*) pour identifier les paramètres de configuration modifiables et non modifiables nécessaires dans le cadre d'une surveillance (re)configurable.

Ainsi le chapitre 1 de la partie III propose une définition de la configurabilité de la surveillance, visant à respecter son utilité première et ses caractéristiques, et tendant à répondre au maximum aux propriétés fonctionnelles (dynamicité, non interruptivité, généricité, gestion intégrée), d'utilisabilité, et d'évolutivité (modularité, expressivité) dégagées précédemment. Cette contribution a été conceptualisée, implémentée et testée.

3.2.2 Adaptabilité

L'« adaptabilité » de la surveillance s'appuie sur la couche configurabilité du *framework* proposé et permet de modifier la configuration des mécanismes de surveillance ainsi que leur portée.

L'adaptabilité est ainsi la capacité à exécuter la réalisation de l'adaptation de la surveillance. Ainsi, cette couche intermédiaire permet d'agir sur l'état d'un ensemble de mécanismes de surveillance

et sur le cycle de vie de chacun de ses éléments (ajout, suppression, modification, suspension, reprise). Des opérateurs d'adaptation ont été spécifiés et implémentés de sorte à mener des actions de contrôle sur le cycle de vie des mécanismes.

La couche d'adaptabilité a été définie, conceptualisée, partiellement implémentée et testée. Elle fait l'objet du chapitre 2 de la partie III.

3.2.3 Gouvernabilité

La « gouvernabilité » de la surveillance s'appuie sur la couche adaptabilité du *framework* proposé. Elle permet de rendre l'adaptation de la surveillance autonome, en fonction du contexte courant du système.

Il s'agit de la capacité de l'infrastructure à pouvoir détecter un besoin d'adaptation de la surveillance et à décider de la « nature » de l'adaptation. Celle-ci est au cœur du plan de contrôle de la surveillance, dont le but est de décider si et comment l'activité de surveillance doit être ajustée.

Afin d'automatiser le pilotage de l'adaptation de la surveillance, il faut qu'il y ait eu détection d'un besoin d'adapter la surveillance. Ce déclenchement peut être causé par deux facteurs résultant de :

- l'analyse des informations de surveillance du système géré ainsi que des besoins métiers,
- l'analyse de la performance de la surveillance elle-même, afin de tendre vers une optimisation de son fonctionnement. Cette analyse vise à adapter d'une part la surveillance au contexte relatif aux disponibilités des ressources d'exécution et à augmenter d'autre part l'efficacité de la surveillance.

Il est donc nécessaire de pouvoir gérer des conditions de déclenchement de l'adaptation et leurs exécutions relatives. Le chapitre 3 de la partie propose un état de l'art des langages disponibles pour exprimer ces logiques et une première implémentation permettant de piloter l'adaptation de la surveillance à l'aide d'un de ces langages.

3.3 Architecture du *framework*

3.3.1 Architecture générique

La figure 3.3 présente, à un haut niveau d'abstraction, l'architecture d'un *framework* permettant de supporter une surveillance adaptative basé sur les trois capacités définies dans la section précédente. La figure détaille plus particulièrement le plan de contrôle, tout en mettant en évidence les dépendances hiérarchiques entre la configurabilité, l'adaptabilité et la gouvernabilité.

D'un point de vue architectural et indépendamment de toute technologie, la surveillance adaptative est composée de deux plans :

- Le plan opérationnel correspond à la gestion de l'exécution des mécanismes de surveillance, conformément à leur configuration respective. Cette configuration est récupérée et appliquée de sorte à piloter le comportement des mécanismes en cours d'exécution.
- Le plan de contrôle de la surveillance adaptative est visible en rouge sur la figure 3.1. Il intègre les trois capacités précédemment établies :
  - le niveau « configurabilité » correspond entre autres à l'ensemble des paramètres de configuration définissant le comportement des mécanismes de surveillance, ainsi que diverses informations telles qu'un ensemble de statistiques,

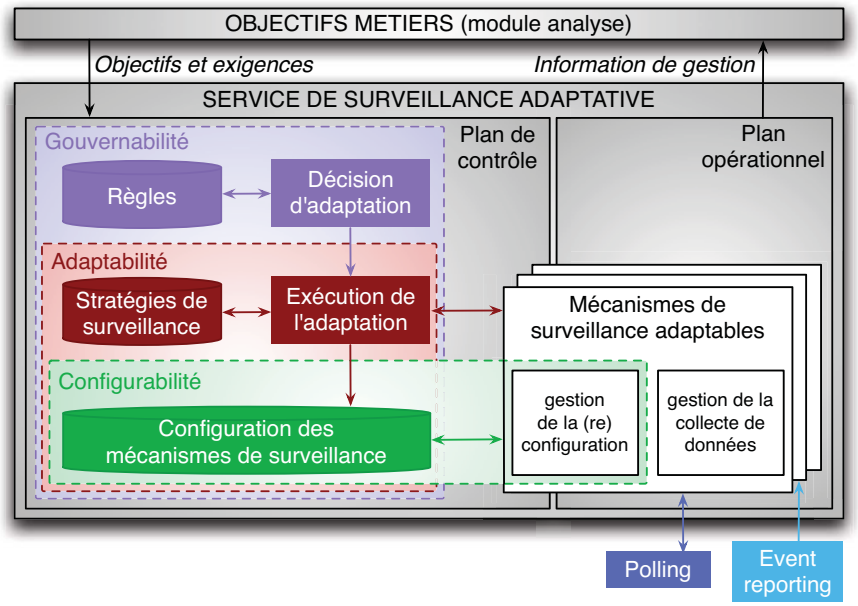

FIGURE 3.3: Architecture générique d'un service de surveillance adaptative

- le niveau « adaptabilité » correspond à un ensemble de stratégies de surveillance permettant d'effectuer un ajustement de la surveillance au moyen d'opérations « basiques » d'adaptation agissant sur des paramètres de configuration ou sur la portée des mécanismes de surveillance,
- le niveau « gouvernabilité », enfin, correspond à un ensemble de règles permettant de prendre la décision d'adapter effectivement la surveillance en appliquant une stratégie de surveillance respectant les contraintes fonctionnelles, environnementales et besoins d'optimisation de haut niveau.

3.3.2 Implémentation en environnement spécifique

Dans un souci de validation, un prototype a été réalisé dans un environnement supportant des standards du DMTF offrant une solution unificatrice d'un point de vue informationnel avec CIM, et protocolaire. L'utilisation de CIM [CIM12] a été choisi car nous souhaitons nous positionner en gestion intégrée et nous obtenons donc une architecture indépendante de toute technologie. De ce fait, l'environnement CIM est ici présenté, son choix est justifié, et l'architecture précédemment définie est adaptée dans l'environnement.

3.3.2.1 Choix technique d'implémentation

3.3.2.1.1 Qu'est-ce que CIM ?

CIM (*Common Information Model*) est un standard : c'est un modèle d'information de gestion générique orienté objet permettant de décrire dans sa globalité tout système géré. Défini par le DMTF [DMTF09], le méta modèle CIM s'appuie sur les concepts objet de classes associatives (héritage et composition) et définit des concepts :

- de correspondance avec des modèles hétérogènes de MIB (qualificatif « *mapping string* »),
- d'indication (notion d'événement) et des mécanismes d'abonnement (Modèle CIM Indication),

- de *patterns*, qui seront explicités et réutilisés plus tard.

Le modèle CIM 3.0 se décompose selon trois niveaux d'abstraction basés autour d'un métamodèle standardisé, permettant l'ajout de concepts provenant du monde de la gestion :

- le *CIM core model* est le cœur du modèle CIM : générique et indépendant de toute technologie, il est à la base de toute conception,
- le *CIM common model* spécialise le *CIM core model* en y ajoutant un ensemble de domaines de gestion : *Devices* caractérise un ensemble d'éléments physiques gérés (*switchs, providers...*), *Networks* définit des types de réseaux, *Users* caractérise des catégories d'utilisateurs...
- le dernier niveau (*Extensions*) correspond à toute extension ajoutée par tout utilisateur, concepteur, développeur souhaitant étendre le domaine en y spécialisant une application ou un domaine de gestion particulier. C'est à ce niveau d'abstraction que toute nouvelle application est modélisée.

L'ensemble des modèles proposés est générique en l'état, mais peut être spécialisé en fonction du besoin, dans le respect du standard et des modèles d'information proposés.

Tout modèle CIM est indissociable d'une architecture de type WBEM [WBEM12], visible sur la figure 3.4.

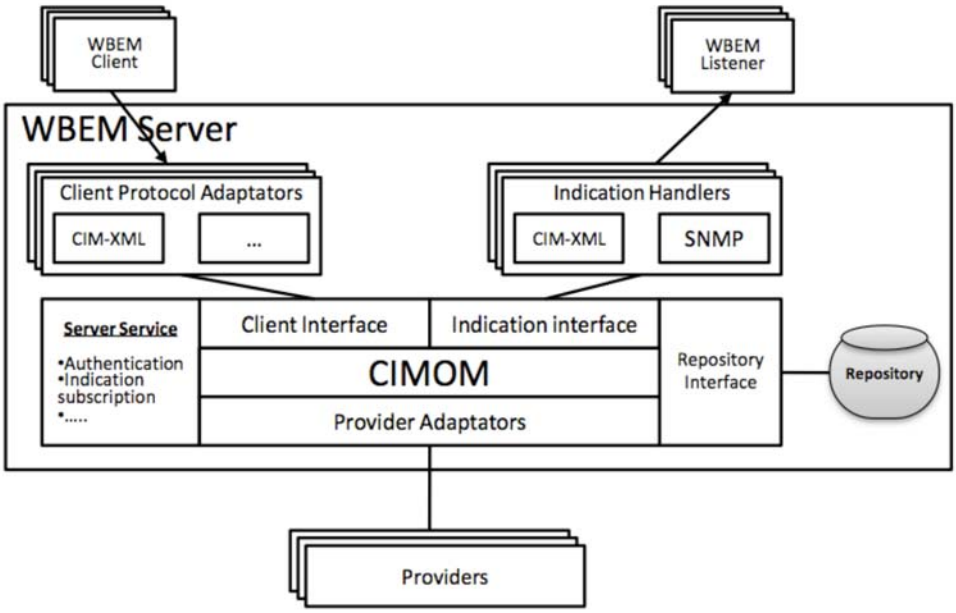

FIGURE 3.4: L'architecture WBEM

Cette architecture est composée des éléments suivants [Park06] :

- le *CIM repository* permet d'héberger l'information de gestion : le modèle d'information étendu ainsi que les instances du modèle étant contenus dans ce composant, il est possible de créer de nouveaux objets, de modifier ou supprimer ces objets, mais aussi d'instancier le modèle dynamiquement et en cours d'exécution,
- le *CIM object manager* (CIMOM) est une interface permettant à l'utilisateur de manipuler et instrumenter des objets CIM contenus dans le *CIM repository*,
- le *CIM provider* est une passerelle entre le CIMOM et les éléments gérés : il a pour rôle de

traduire en opérations CIM les informations en provenance de sources hétérogènes externes (par exemple, il peut traduire en *CIM indication* toute notification d'événements) ; le *CIM provider* fait office de couche d'intégration,
- le *WBEM listener* est un composant chargé de réceptionner des événements de type *CIM indications*,
- le *WBEM client* est l'interface au travers de laquelle des opérations WBEM peuvent être invoquées : il offre un moyen d'interagir avec le *CIM repository* en fournissant des opérations standards de consultation, de création, de modification et de suppression d'objets CIM.

3.3.2.1.2 Pourquoi CIM ?

Plusieurs raisons poussent à implémenter le *framework* dans un environnement de type CIM/WBEM.

Tout d'abord, et par définition, CIM est utilisé dans un but d'unification des éléments sous-jacents et est de ce fait indépendant des plateformes, des logiciels, des systèmes et de l'architecture. En conséquence, un tel modèle peut être utilisé dans le domaine de la gestion intégrée, qui vise à gérer de manière conjointe et unifiée des composants hétérogènes dans des domaines hétérogènes.

Ensuite, les trois couches du *framework* impliquent la nécessité de manipuler de l'information de gestion selon plusieurs niveaux d'abstractions. Un modèle d'information tel que CIM s'avère utile pour manipuler cette information sémantiquement : elle peut être enregistrée de manière structurée, selon des relations de dépendance, d'héritage ou de composition correspondant à leur nature propre. Les concepts objets peuvent ici être utilisés de manière efficace et adaptée à la vision globale du problème.

De plus, CIM est un standard apportant de fait un modèle d'information générique qu'il reste à étendre en le spécialisant avec les concepts constitutifs du *framework* mettant en place un service de surveillance adaptative. Un parallèle peut d'ores et déjà être observé entre la configuration des mécanismes de surveillance et les *patterns* CIM préexistants attendant d'être utilisé. De même une connexion peut être établie entre nos éléments gérés et les *managed system elements* de CIM, qu'il faut employer correctement et donc bien déterminer.

L'environnement CIM permet de se positionner à un haut degré d'abstraction, tout en apportant des moyens d'intégrer et gérer de manière générique des domaines, éléments, composants hétérogènes, mais aussi de réutiliser des modèles (*Application*, *Network*, *System*), les *patterns* et éventuellement les *Indications*. Ce sont pour ces raisons que le *framework* est (parfois seulement partiellement) spécifié et implémenté dans un environnement CIM/WBEM.

3.3.2.2 Architecture spécifique en environnement CIM/WBEM

La figure 3.5 montre comment l'architecture logique générale définie dans la figure 3.3 de façon indépendante à un environnement technologique d'implémentation est transposée dans un environnement CIM/WBEM.

Sur la partie gauche de l'architecture sont représentés les modèles sur lesquels repose la mise en œuvre des trois capacités précédemment définies (configurabilité, adaptabilité, gouvernabilité) et permettant la réalisation de l'adaptation d'un service de surveillance :
- en bas à gauche, le schéma CIM a été défini pour permettre la gestion de la configurabilité des mécanismes de *polling* et d'*event reporting* (cf. partie 1),

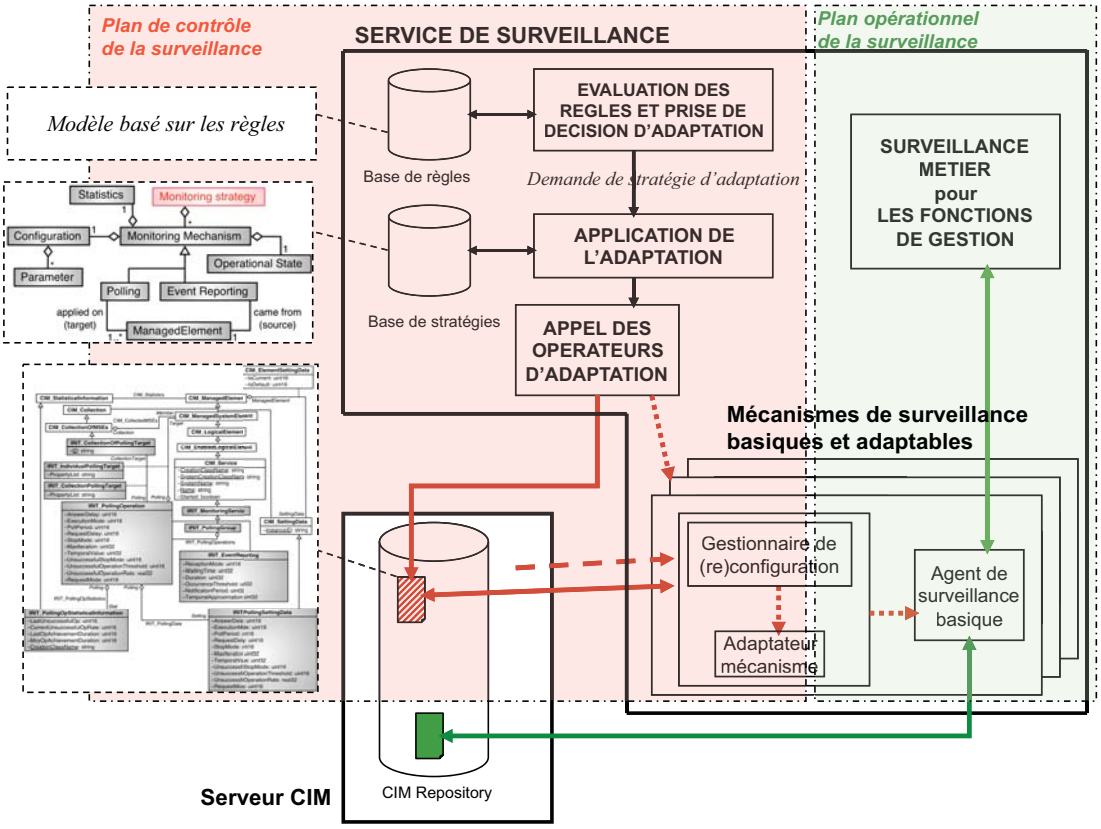

FIGURE 3.5: Architecture d'un service de surveillance adaptative en environnement CIM

- au-dessus, le diagramme de classes a permis de modéliser l'adaptabilité : celui-ci constitue une structuration informationnelle servant de base à la gestion de stratégies de surveillance et de leurs états (cf. partie 2),
- enfin, en haut, un modèle permettant l'expression de règles d'adaptation doit être conçu et utilisé pour mettre en œuvre la capacité de gouvernabilité (cf. partie 3).

Deux entités logicielles apparaissent au sein de cette architecture :

- Le serveur CIM/WBEM supporté par le logiciel libre Open Pegasus [OpenPegasus] offre un support à la reconfiguration en temps réel et sans interruption les mécanismes de base permettant la collecte d'information de surveillance (plan de contrôle). Les configurations qui y sont stockées sont accédées en parallèle par les *pollers* et *listeners* adaptables qui sont chargés d'ajuster leur comportement respectif selon ces informations de surveillance (plan opérationnel).
- Le service de surveillance adaptative assure les fonctionnalités de la surveillance (plan opérationnel) et le pilotage autonome de l'activité de surveillance (plan de contrôle). L'architecture interne de ce service est composé des entités suivantes :
  - Un fournisseur d'opérateurs d'adaptation dans un environnement CIM/WBEM offre la possibilité d'adapter la configuration d'une « stratégie de surveillance » (il s'agit d'une vue à un moment donné d'un ensemble de mécanismes de surveillance comprenant leurs cibles, leur configuration et leur état opérationnel) basée sur un ensemble de mécanismes de

surveillance par lancement des opérateurs d'adaptation ($\mathscr{AUDRS}$). L'exécution de chacun de ces opérateurs se traduit par la mise à jour des instances CIM du *repository* du serveur CIM.
- Une entité intermédiaire doit permettre de concrétiser l'ajustement du comportement du service de surveillance en gérant le changement de stratégies de surveillance. Ce niveau peut être implémenté indépendamment de l'environnement CIM/WBEM sous-jacent.
- Une entité doit enfin permettre l'évaluation des règles d'adaptation de la surveillance. Elle est alors responsable de la prise de décision de modification d'une stratégie de surveillance.

3.4 Conclusion

Chaque mécanisme de surveillance adaptable peut être vu comme un processus ou un service en cours d'exécution dont les objectifs opérationnels sont de collecter de l'information de gestion et de la délivrer ensuite à des fonctions de gestion de haut niveau (comme le module d'analyse de la boucle MAPE). De plus, chaque mécanisme doit être capable de s'auto-configurer initialement en récupérant les valeurs de ses paramètres de configuration afin de gouverner son propre comportement opérationnel. Enfin, chaque mécanisme doit être capable de prendre en compte tout changement ayant été opéré sur les valeurs de ses paramètres de configuration ou sa portée (ajout ou suppression de cibles interrogeables). Ces éléments correspondent à la configurabilité.

Un ensemble de mécanismes adaptables permet à l'activité de surveillance de remplir au mieux les exigences exprimées par les règles de gestion haut niveau. La gouvernabilité est la capacité qu'a le *framework* à prendre des décisions d'adaptation de manière dynamique à partir de sa base de connaissance et à lancer l'exécution des actions d'ajustement sur l'activité de surveillance. Gérer en cours d'exécution la réalisation opérationnelle de ces décisions d'adaptation correspond à l'adaptabilité.

Un *framework* a été réalisé pour expérimenter ces concepts. Le *framework* est tout d'abord capable de supporter la capacité de modifier dynamiquement l'ensemble des configurations et états opérationnels de chaque mécanisme de surveillance adaptable ; ensuite, est intégrée la capacité à exiger dynamiquement l'ajustement du comportement de chaque mécanisme, dont la réalisation est supportée par le niveau de configurabilité. La réalisation opérationnelle de cette adaptation passe par des actions particulières devant être consistantes et cohérentes. Ainsi, chaque couche du *framework* doit présenter une interface clairement définie grâce à une conception générique offrant toute possibilité d'être spécialisée au niveau implémentation sur n'importe quel type d'environnement technologique (même si l'exemple de l'environnement CIM est choisi). Ces éléments font l'objet du chapitre 1 de la partie III.

Des opérateurs doivent être définis pour permettre une modification de la configuration et de la portée de la surveillance, et des stratégies doivent être mises en place pour déterminer l'ensemble des opérations devant être réalisées conjointement pour ajustement le comportement des mécanismes de surveillance. L'adaptabilité fait l'objet du chapitre 2 de la partie III.

Enfin, la décision d'adaptation et le choix d'appliquer certaines stratégies revient à la capacité de gouvernabilité, présentée au chapitre 3 de la partie III.

Troisième partie

Contributions

La configurabilité ou comment (re)configurer la surveillance

1

Pour rendre les mécanismes de surveillance configurables, il est nécessaire d'une part de caractériser la configuration des mécanismes de surveillance dans leur globalité, mais aussi de définir les algorithmes permettant de prendre en compte ces caractéristiques initialement et en cours d'exécution. Une implémentation dans l'environnement technique choisi, CIM, et une première contribution par le biais d'un prototype de surveillance configurable peut alors être présentée.

1.1 Définition

La configurabilité se base sur le principe qu'un mécanisme de surveillance, *polling* ou *event reporting*, doit prendre en compte, initialement et quand des changements sont requis, certaines valeurs de paramètres influençant leur manière d'opérer. Les mécanismes doivent être rendus configurables. Il devient nécessaire d'étudier la configuration de chacun des mécanismes de surveillance (*polling* et *event reporting*) pour mettre en évidence les paramètres de configuration modifiables et non modifiables utiles dans le cadre d'une surveillance (re)configurable.

Un comportement opérationnel particulier peut être déterminé intialement, puis réajusté en cours d'exécution. Ainsi, nous définissons la configurabilité comme suit :

Définition

La **configurabilité** est la capacité d'initialiser et de modifier dynamiquement et sans interruption la portée (autrement dit, l'ensemble des cibles gérées) et les valeurs des paramètres gouvernant le comportement d'un mécanisme de surveillance.

Cette partie propose une caractérisation de la configurabilité des mécanismes de surveillance. Cette contribution a été spécifiée, implémentée et testée pour démontrer la faisabilité de la capacité de configurabilité.

1.2 Caractéristiques des mécanismes de surveillance

1.2.1 Caractérisation de la configuration de la surveillance

Un « mécanisme de surveillance » est soit un mécanisme de « *polling* » appliqué sur une (ou plusieurs) cible(s), soit un mécanisme d'« *event reporting* » dont les notifications sont en provenance d'une (ou plusieurs) source(s). Dans un souci de généricité et de simplification de l'écriture, « cible » et « source » sont regroupés sous l'appellation « éléments gérés » (définis ci-après). Chaque mécanisme de surveillance possède quant à lui une « configuration », cette dernière étant composée d'un ensemble de « paramètres ».

Tous ces éléments – les mécanismes de surveillance, leur configuration et les éléments gérés – sont représentés conceptuellement sur le diagramme de classes de la figure 1.1.

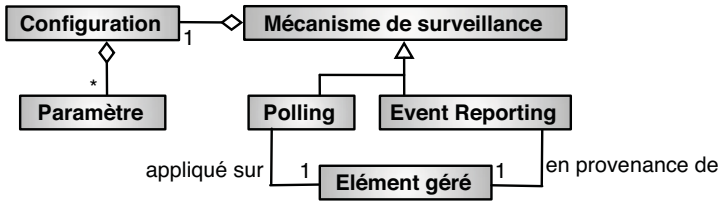

FIGURE 1.1: Configurabilité de la surveillance

Chacun des éléments visibles sur la figure sont passés en revue dans les sections suivantes pour spécifier précisément les caractéristiques de la surveillance.

1.2.1.1 Portée de la surveillance intégrée

Les mécanismes de surveillance sont appliqués sur un ensemble de composants en fonctionnement du système géré/surveillé, dont l'information de gestion est la « cible » (désignée par T) de la surveillance intégrée. Ces cibles représentent des objets de consultation pouvant être soit le composant dans sa globalité (une vision générale d'un composant) ou de l'information de gestion concernant ce composant (une vision à très fine échelle d'un composant : un ensemble de propriétés représentatives de l'élément). Le terme générique de « cible » est utilisé pour désigner toute information pouvant être récupérée par un mécanisme de surveillance, quelle qu'elle soit.

A un haut niveau d'abstraction, le système est ainsi composé d'un ensemble non nul de cibles. Il s'agit d'une vision orientée gestion d'un composant réel :

$$\forall\, n \in \mathbb{N}^{\star},\ T = \langle t_1, t_2, ..., t_n \rangle$$

Dans une approche de modèle d'information de gestion orienté objet, qui permet de représenter de manière unifiée ce type de concepts, une distinction peut être établie en ce qui concerne la nature de cette information pour chaque mécanisme :

- en ce qui concerne le mécanisme de *polling* (qui correspond à un mode de collecte *pull*, les cibles peuvent être :
  - un objet géré :

$$\forall\, t_k \in T,\ t_k = Object$$

- une collection d'objets gérés :
$$\forall\, t_k \in T,\ t_k = \{Object\}$$
- une propriété d'un objet géré :
$$\forall\, t_k \in T,\ t_k = Object.property$$
- un ensemble de propriétés d'un objet géré :
$$\forall\, t_k \in T,\ t_k = \{Object.property\}$$

- en ce qui concerne le mécanisme d'*event reporting*, les cibles de la surveillance sont plutôt les sources d'où provient l'information (alertes, avertissements, données particulères, information préalablement filtrée ou agrégée...). Il s'agit d'un mode de collecte « *push* ».

Les cibles ayant été définies, il faut désormais spécifier la façon de les surveiller, au moyen des mécanismes de surveillance.

1.2.1.2 Configuration opérationnelle des mécanismes de surveillance

A un haut niveau d'abstraction, les mécanismes de *polling* et d'*event reporting* sont regroupés sous la dénomination commune de « mécanismes de surveillance ». Cet ensemble, désigné par M, appliqué sur l'ensemble des cibles T, est alors la réunion non vide des mécanismes de *polling* P et d'*event reporting* E :
$$M_T = P_T \cup E_T \neq \emptyset$$

Notons que les écritures $M_T = P_T$ et $M_T = E_T$ sont aussi possibles. Il est possible de bâtir des systèmes de surveillance n'utilisant qu'un type de mécanisme. Leur utilisation conjointe n'est pas une obligation en soi.

Chaque mécanisme de surveillance possède une configuration C_f définissant son propre comportement. Cette configuration est composée de paramètres modifiables Pa_{mdf} et de paramètres non modifiables $Pa_{\neg mdf}$. Pour cela, soit inst la fonction permettant d'extraire la valeur correspondant à un paramètre :
$$C_f = \{\text{inst}(Pa_{mdf})\} \cup \{\text{inst}(Pa_{\neg mdf})\}$$

Divers paramètres ont été et peuvent encore être définis pour caractériser comment sont surveillées les éléments gérés, ou cibles, précédemment décrits. Les paramètres étant différents selon le mécanisme de surveillance, il faut désormais définir à la fois les mécanismes et leurs paramètres respectifs.

1.2.1.2.1 Configuration du *polling*

Selon un point de vue orienté contrôle et à un haut degré d'abstraction, le mécanisme de *polling* P peut être défini comme l'ensemble de i opérations successives de consultation C sur une cible t_k :
$$\forall\, i \in \mathbb{N}^\star,\ \forall\, t_k \in T,\ P_{t_k} = \langle C_1, C_2, ..., C_i \rangle \text{ tel que } startTC_i < startTC_{i+1}$$

Sur la figure 1.2, quatre occurrences de consultation C sont visibles. C_1, C_2 et C_3 sont constituées de deux messages : une demande de consultation et une réponse. Ainsi, il est possible de dire

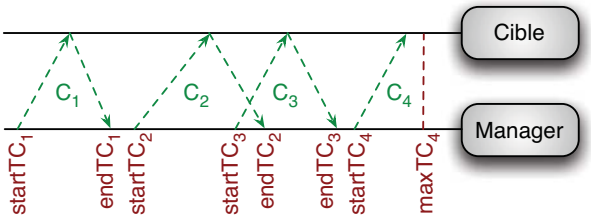

FIGURE 1.2: Le mécanisme de *polling*

que C est l'ensemble des opérations réalisées entre le début de l'émission de la demande de consultation (*startTC*) et la réception de la réponse (*endTC*).

Notons qu'un temps d'attente maximum *maxTC* peut également être positionné. Ce temps d'attente maximum, s'il est dépassé, indique que le message de réponse n'a pu être reçu suffisamment tôt pour être pris en compte (une réponse ne peut en effet être attendue de manière infinie, car cette attente pourrait bloquer le système). Ce cas est illustré par l'occurrence C_4 sur la figure 1.2.

Etre capable de gérer un mécanisme de *polling* signifie qu'il est possible d'avoir une influence sur son exécution. Plusieurs paramètres ont été identifiés, de sorte à déterminer notamment comment le *polling* est exécuté, comment l'opération de consultation est réalisée, quand et comment le *polling* s'arrête, et quelle doit être la productivité des opérations de consultation.

1.2.1.2.1.1 Mode d'exécution

Le mode d'exécution (`ExecutionMode`) définit l'ordonnancement temporel des opérations de consultation du *polling* (figure 1.3).

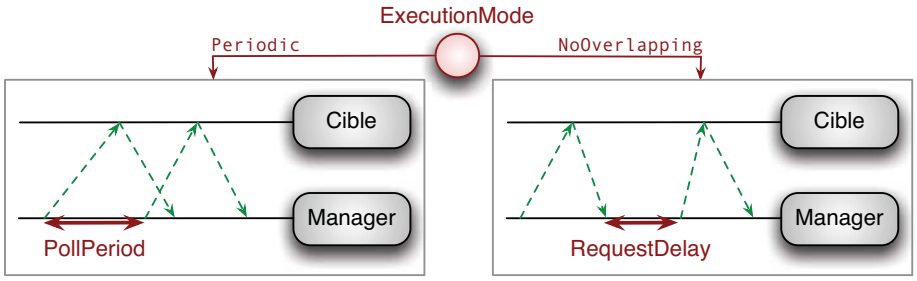

FIGURE 1.3: Mode d'exécution du *polling*

Ce paramètre peut prendre deux valeurs :

- `Periodic` : les opérations de consultation sont lancées en fonction de la période du *polling* (`PollPeriod`). Ce paramètre, représenté par $\Delta startTC_{(i,i+1)}$, est l'intervalle de temps séparant le début de deux opérations de consultation consécutives :

$$\Delta startTC_{(i,i+1)} = startTC_{i+1} - startTC_i$$

- `NoOverlapping` : dans ce cas, il faut considérer l'intervalle de temps (`RequestDelay`), représenté $interC_{(i,i+1)}$, qui fixe la durée entre la fin d'une consultation et le début de la suivante :

$$interC_{(i,i+1)} = startTC_{i+1} - endTC_i > 0$$

Rien n'empêche de commuter d'un mode d'exécution à un autre en cours d'exécution : il peut en effet être utile de modifier la fréquence des consultations, d'empêcher un entrelacement des occurrences pour, par exemple, alléger la charge CPU pour des problèmes de contraintes opérationnelles, et de revenir en mode périodique plus tard. De même les valeurs de la période de *polling* et d'intervalle de temps peuvent également être modifiées pour interroger des cibles de manière plus intensive sur une période donnée ou pour relâcher la période d'interrogation.

1.2.1.2.1.2 Mode de terminaison

Le mode de terminaison (`StopMode`) détermine la manière dont se termine le *polling* (figure 1.4).

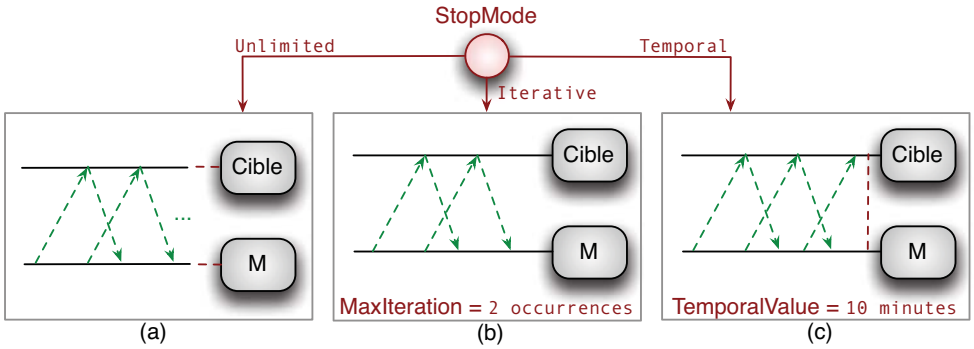

FIGURE 1.4: Mode de terminaison du *polling*

Ce paramètre sélecteur peut prendre l'une des trois valeurs suivantes :

- `UnlimitedPolling` : on ne souhaite pas prédéterminer la fin du *polling*. Puisqu'aucune borne d'arrêt n'est précisée, il est alors considéré comme « illimité ». Il s'agit du cas (a) de la figure 1.4.
- `IterativelyBounded` : le *polling* est limité et s'arrête lorsque son nombre d'occurrences atteint le seuil maximum d'itération (`MaxIteration`). Dans le cas (b) de la figure 1.4, le seuil est fixé à deux occurrences de consultation.
- `TemporallyBounded` : le *polling* est limité et s'arrête au bout d'un certain temps (`TemporalValue`). Dans le cas (c) de la figure 1.4, le *polling* doit s'arrêter dix minutes après le début de la première opération.

Bien que le nombre maximum d'itérations et la borne temporelle puissent voir leur valeur modifiée, le mode d'arrêt lui est un paramètre que l'on définit comme non modifiable. En effet, il semble illogique de modifier en cours d'exécution la condition d'arrêt sur nombre d'occurrences vers un arrêt temporel ou encore vers la poursuite du *polling* à l'infini, ou *vice versa*. Chaque mode correspond ici à une condition d'arrêt particulière définie initialement.

1.2.1.2.1.3 Temps d'attente

Le paramètre `AnswerDelay` correspond à $MaxTC_i$, le temps d'attente maximum du message de réponse à une demande de consultation (figure 1.5).

Deux cas de figures se présentent :

- `ProductiveRequest` : la réponse a été reçue dans le temps qui lui a été imparti. La requête est dite « productive ». Il s'agit du cas (a) de la figure 1.5.
- `UnproductiveRequest` : la requête est dite « improductive » dans plusieurs cas :

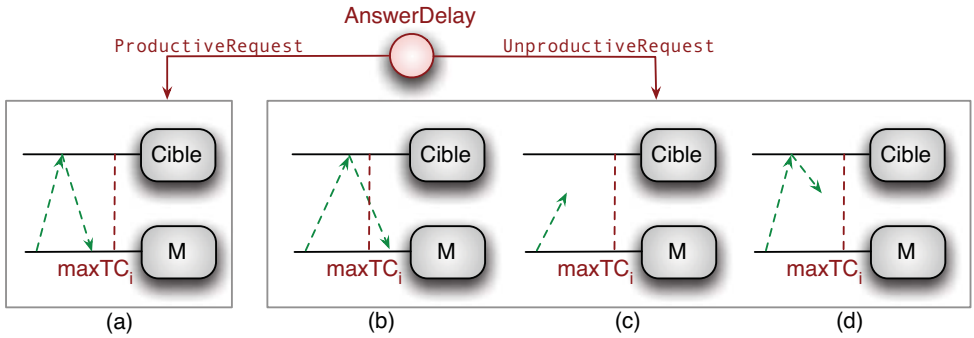

FIGURE 1.5: Temps d'attente maximum de la réponse du *polling*

- la réponse a été reçue trop tard : le temps de latence correspond au cas (b) de la figure 1.5,
- le message de demande de l'occurrence de consultation est perdue (n'est jamais reçue par la cible) : il s'agit du cas (c) de la figure 1.5,
- le message de réponse de l'occurrence de consultation est perdue (la cible a reçu la demande mais le *manager* n'a jamais reçu la réponse) : il s'agit du cas (d) de la figure 1.5.

La valeur de ce paramètre est directement liée à l'exécution des opérations de consultation et il semble intuitif de pouvoir la modifier dynamiquement en fonction des nécessités de la surveillance : par exemple, un temps d'attente maximum plus long peut être accordé lors d'un encombrement du système géré, car on sait qu'un encombrement ralentit l'acheminement des flux d'information.

1.2.1.2.1.4 Mode de terminaison autonome

Le mode de terminaison autonome (`UnsuccessfulStopMode`) permet à un mécanisme de s'arrêter par lui-même dans le cas où le nombre de requêtes improductives [1] excède un seuil prédéfini (figure 1.6).

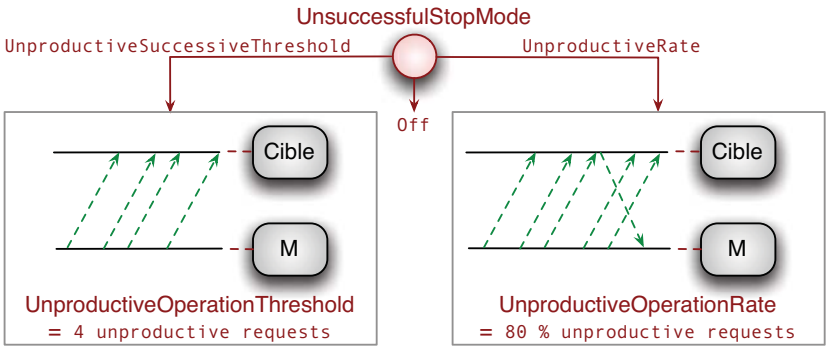

FIGURE 1.6: Mode de terminaison autonome du *polling*

Ce paramètre intervient à la fois dans le cas d'un *polling* illimité et dans le cas d'un *polling* limité itérativement ou temporellement, auquel cas, le mode de terminaison autonome devient prioritaire par rapport au mode de terminaison classique. Le paramètre sélecteur peut prendre trois valeurs :

1. Pour rappel, une requête est dite « improductive » dans le cas du *polling* dans les cas suivants : la réponse à la demande de consultation est reçue trop tard, ou l'un des messages de l'opération de *polling* est perdu (demande ou réponse).

- `Off` : le mode de terminaison autonome n'est pas activé.
- `UnproductiveRate` : dans ce cas de figure, deux paramètres internes sont nécessaires pour comptabiliser le nombre d'opérations et le nombre d'opérations improductives. Le premier est incrémenté à chaque début d'opération $startTC_i$, et le second est incrémenté à chaque fois qu'une opération de consultation C_i est jugée improductive. Si le ratio atteint la valeur définie par le sous-paramètre `UnproductiveOperationRate`, alors le processus de *polling* s'arrête. A gauche sur la figure 1.6, le *polling* s'arrête lorsque 80% des opérations de consultation sont improductives.
- `UnproductiveSuccessiveThreshold` : de manière similaire, un paramètre interne est nécessaire pour calculer le nombre de requêtes improductives successives. Si cette valeur atteint la valeur définie par le sous-paramètre `UnproductiveOperationThreshold`, alors le mécanisme s'arrête. A droite sur la figure 1.6, le *polling* s'arrête lorsqu'il y a eu quatre opérations de consultation successives improductives.

Comme pour le mode de terminaison, il semble illogique de vouloir modifier en cours d'exécution le mode de terminaison autonome d'un *polling*. En effet, soit celui-ci est positionné sur un taux d'opérations de consultations improductives ou sur un nombre d'opérations de consultation successives improductives. Le passage d'un mode à l'autre engendrerait des erreurs quant aux calculs relatifs à l'arrêt autonome de la terminaison. Toutefois, bien que ce paramètre sélecteur soit non modifiable, les valeurs de ses paramètres complémentaires (le seuil d'opérations de consultation successives improductives et le taux d'opérations de consultations improductives) sont, eux, modifiables dynamiquement.

1.2.1.2.1.5 Mode de réception

Lorsque l'information de gestion collectée est directement produite et fournie par l'agent placé sur la source de l'information de gestion, le *polling* est dit `direct`. Dans ce cas, il n'y a pas d'entité intermédiaire entre l'agent et le *manager*. Autrement, lorsque la collecte est réalisée sur une entité intermédiaire (qui n'est pas le producteur de l'information), le *polling* est défini comme `local` (figure 1.7). Dans ce cas, la fraîcheur des données contenues sur l'entité intermédiaire dépend du moment où la collecte a été réalisée sur la source.

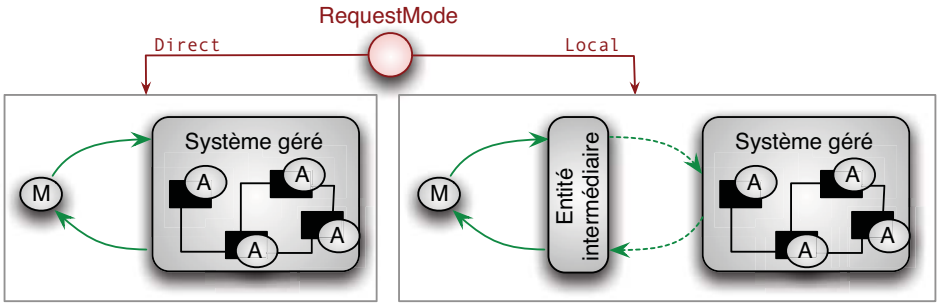

FIGURE 1.7: Mode de consultation du *polling*

Il apparaît ici évident que le mode de réception ne peut être modifié en cours d'exécution. En effet, soit une entité intermédiaire est positionnée initialement, soit non.

1.2.1.2.1.6 Récapitulatif L'ensemble des paramètres présentés ici pour le *polling* constitue une liste non exhaustive. Il est possible de définir de nouveaux paramètres de *polling* en fonction des besoins concernant la surveillance du système géré. Par exemple, concernant le mode de terminaison du *polling*, nous avons défini un mode d'arrêt autonome sur un seuil ou un taux d'opérations improductives : autrement dit, il s'agit ici d'un « mode d'arrêt sur échec ». Il est aussi possible d'envisager l'ajout d'un « mode d'arrêt sur réussite », qui pourrait être utile pour déterminer qu'un composant du système géré fonctionne à nouveau correctement (vu qu'il répond à une demande de consultation à laquelle il ne répondait pas avant). Toutefois, ce mode d'arrêt ne fait pas l'objet de notre étude.

Le tableau 1.1 liste les paramètres et leurs caractéristiques qui ont été présentés dans cette section. Certains paramètres sont modifiables dynamiquement (ils sont en noir) : ce sont des mécanismes décisionnels qui permettent de contrôler la configuration des opérations de *polling* et ainsi de faire varier son comportement. Le choix de cette « modifiabilité » de chacun de ces paramètres est justifié avec leurs définitions respectives.

La colonne de gauche correspond au paramètre sélecteur. Il s'agit d'un paramètre gouvernant le comportement général du mécanisme (par exemple le mode d'exécution). Selon la valeur de ce paramètre sélecteur (par exemple périodique), un sous-paramètre est introduit pour valuer le paramètre sélecteur (par exemple période de 10 ms).

Paramètre sélecteur	Valeur	Autre paramètre
		AnswerDelay (ulong)
Execution Mode (enum)	Periodic	PollPeriod (ulong)
	NoOverlapping	RequestDelay (ulong)
StopMode (enum)	Unlimited	
	Iterative	MaxIteration (ulong)
	Temporal	TemporalValue (ulong)
Unsuccessful StopMode (enum)	Off	
	UnproductiveSuccessiveThreshold	UnproductiveOperationThreshold (ushort)
	UnproductiveRate	UnproductiveOperationRate (ulong)
Request Mode (enum)	Source	
	Local	

TABLE 1.1: Paramètres de configuration du *polling*

1.2.1.2.2 Configuration de l'*event reporting*

Selon un point de vue orienté contrôle et à un haut degré d'abstraction, le mécanisme d'*event reporting* E peut être défini comme l'ensemble, vide ou non, de i opérations de notifications N en provenance d'une source t_k :

$$\forall\, i, x \in \mathbb{N}^2,\ \forall\, t_k \in T,\ E_{t_k} = \langle N_1, N_2, ..., N_i \rangle \text{ tel que } startTN_i < TN_{i+x}$$

Sur la figure 1.8, quatre opérations de notifications N sont visibles. Ainsi, TN_i est définie comme la réception d'une notification à un instant i et TN_{i+x} la réception d'une notification à un instant $i+x$, avec $x \in \mathbb{N}$. Pourquoi x ? Il faut ici prendre en compte le fait que la fréquence de réception des

notifications n'est pas contrôlable par le consommateur : les notifications sont émises lorsqu'une situation le nécessite et leur fréquence de réception ne peut pas toujours être déterminée par avance. La notation x nous permet ainsi de prendre en compte cette non périodicité des notifications.

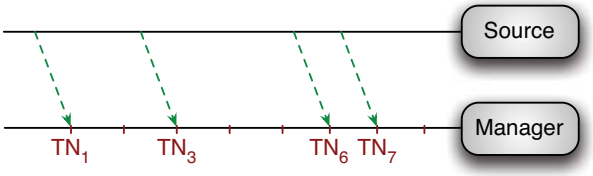

FIGURE 1.8: Le mécanisme d'*event reporting*

Nous décidons ici d'étudier comment sont reçues les notifications par le consommateur, en faisant abstraction de leur contenu : observer la fréquence de réception peut être significatif d'un problème particulier survenant sur le système géré. Par conséquent, le paramètre sélecteur `ReceptionMode` a été défini pour sélectionner l'intention de la surveillance de la fréquence afin d'identifier un comportement particulier.

Le tableau 1.2 liste les caractéristiques du mécanisme. Contrairement à la configuration du *polling*, les paramètres sont ici tous modifiables : il est possible, en cours d'exécution, de commuter d'un mode d'observation à un autre. Les notifications étant envoyées par les agents au consommateur en fonction d'événements significatifs de problèmes sur le système géré, l'intensité des notifications sera différente selon les périodes : parfois énormément de notifications, parfois aucune. Ainsi, l'intérêt de basculer d'un mode d'observation à l'autre permet de pouvoir détecter l'un ou l'autre de ces comportements lorsqu'il survient.

Paramètre sélecteur	Valeur	Autre paramètre
ReceptionMode (`enum`)	`Silence`	WaitingTime (`ulong`)
	`Burst`	Duration (`ulong`)
		OccurrenceThreshold (`ulong`)
	`Heartbeat`	NotificationPeriod (`ulong`)
		TemporalApproximation (`ulong`)

TABLE 1.2: Paramètres de configuration de l'*event reporting*

Trois modes d'observation ont été identifiés (figure 1.9) :

- `Silence` : Aucune notification n'est reçue pendant un intervalle de temps fixé. Cette situation peut être interprétée de deux façons : rien n'est à déclarer par les agents et le système fonctionne correctement ; ou bien une ou plusieurs notifications ne sont jamais arrivées à destination, impliquant alors potentiellement un problème sur le système géré (par exemple, le composant peut être en panne). Ainsi, si ce mode est sélectionné, un paramètre est nécessaire pour fixer le temps d'attente maximum (`WaitingTime`) d'une réception de notifications.
- `Burst` : Un surnombre de notifications sont reçues durant un intervalle de temps fixé. Ce comportement peut également être significatif d'un problème sur le système. Deux paramètres ont été définis pour gérer ce comportement : la durée (`Duration`) sur laquelle le comptage est

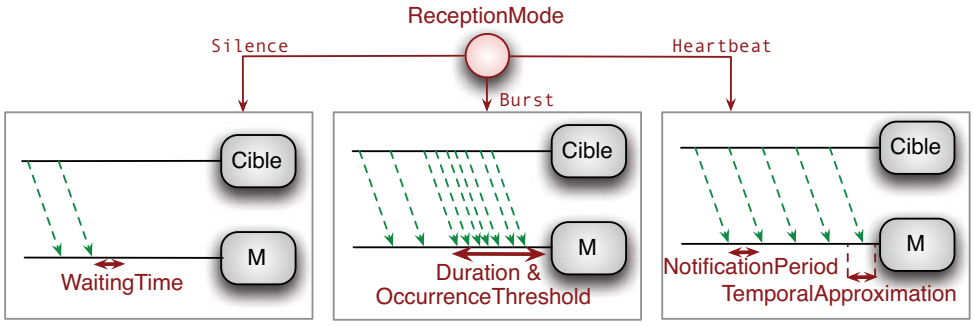

FIGURE 1.9: Le mode de réception de l'*event reporting*

réalisé et un seuil d'occurrences comptabilisés (`OccurrenceThreshold`) qui permet de conclure à l'identification d'un comportement sporadique.

- `Heartbeat` : La réception des opérations de notifications devrait être isochrone. La surveillance de cette situation implique l'utilisation de deux paramètres supplémentaires : la période de réception des notifications (`NotificationPeriod`) et un écart temporel autorisé (`TemporalApproximation`), utile notamment pour des raisons d'encombrement du trafic réseau.

La liste des paramètres de configuration d'un *event reporting* peut être complétée par d'autres caractéristiques utiles à l'observation ou la réception des notifications, voire même par la définition d'un mode d'arrêt autonome de l'observation. De plus, le mécanisme est ici considéré uniquement du côté « consommateur ». Côté « fournisseur », d'autres paramètres d'émission des notifications peuvent être intégrés : le nombre de notifications corrélées avant l'envoi d'une seule notification, la possibilité de filtrer un certain nombre de notifications. Nous avons ici pour l'instant fait abstraction de ces diverses caractéristiques, car on se place davantage du côté des consommateurs de l'information de gestion.

1.2.2 Algorithmes permettant la configurabilité

Les mécanismes de surveillance ainsi que les éléments gérés sur lesquels ils opèrent ont été spécifiés de manière semi-formelle afin d'apporter une liste de paramètres de configuration permettant de déterminer initialement et d'ajuster dynamiquement le comportement des mécanismes de *polling* et d'*event reporting*.

Il faut désormais intégrer ces caractéristiques des mécanismes de surveillance à des algorithmes permettant de contrôler le comportement des mécanismes de *polling* et *event reporting* en fonction des valeurs de paramètres courantes : globalement, ces algorithmes permettent de récupérer la configuration courante et de la mettre en application initialement, en cours d'exécution, puis de considérer une reconfiguration issue de modifications de paramètres de configuration.

1.2.2.1 Algorithmes concernant le mécanisme de *polling*

L'algorithme général correspondant à l'exécution d'un *polling* est visible sur la figure 1.10. Cet algorithme sert à déterminer quel doit être le comportement initial d'un programme implémentant un service de collecte d'information (un *poller*) paramétrable.

```
Poller
1. ArmerTemporisation(AnswerDelay);
2. OccurenceConsultation(cible, ModeDeConsultation)
   [Si Expiration(AnswerDelay) alors Traitement1()];
3. CompteurOperationsSuccessivesImproductives <- 0;
4. Continuer <- Poursuivre () ;
5. Si ExecutionMode = NoOverlapping alors
   5.1 TantQue Continuer = TRUE ET ProductiviteOK = TRUE faire
       5.1.1 Attendre(PollPeriod);
       5.1.2 ArmerTemporisation(AnswerDelay);
       5.1.3 NbOperationsInvoquees ++;
       5.1.4 OccurenceConsultation(cible, ModeDeConsultation)
             [Si Expiration(AnswerDelay) alors Traitement1() puis 5.1.6];
       5.1.5 CompteurOperationsSuccessivesImproductives <- 0;
       5.1.6 Continuer <- Poursuivre () ;
   5.2 FinTantQue
6. Si ExecutionMode = Periodic alors
   6.1 Chaque PeriodeDePolling faire
       6.1.1 Continuer <- Poursuivre () ;
       6.1.2 Si Continuer = TRUE ET ProductiviteOK = TRUE alors
             6.1.2.1 ArmerTemporisation(AnswerDelay);
             6.1.2.2 NbOperationsInvoquees ++;
             6.1.2.3 OccurenceConsultation(cible, ModeDeConsultation)
                     [Si Expiration(AnswerDelay) alors Traitement1() puis 6.1.1];
             6.1.2.4 CompteurOperationsSuccessivesImproductives <- 0;
       6.1.3 Sinon Stopper la boucle
   6.2 FinChaque

Traitement1()
1. NbOperationImproductive ++ ;
2. ProductiviteOK <- true ;
3. CptOpSuccImproductives ++ ;
4. Si UnsuccessfulStopMode = UnproductiveSuccessiveRate alors
5. Si NbOperationsImproductives / NbOperations > UnproductionOperationRate alors
   ProductiviteOK <- false ;
6. Si UnsuccessfulStopMode = UnproductiveSuccessiveThreshold alors
7. Si CptOpSuccImproductives > UnproductiveOperationThreshold alors
   ProductiviteOK <- false ;

Boolean Poursuivre ()
1. Poursuivre <- true ;
2. Si StopMode = iterative alors
2.1. Si NbOperations >= MaxIteration alors Poursuivre <- false ;
3. Si StopMode = temporal alors
3.1. Si Duree >= TemporalValue alors Poursuivre <- false ;
4. retourner(Poursuivre) ;
```

FIGURE 1.10: Algorithme général d'un *poller* paramétré

Les variations des paramètres de configuration impliquent d'étendre cet algorithme pour permettre leur prise en compte. Les algorithmes suivants sont exprimés à travers une description en SDL de la structure et du comportement que les processus doivent adopter lorsque les paramètres de configuration varient. Les types de données utilisés dans les algorithmes de cette section sont présentés en figure 1.11.

Le processus présenté en figure 1.12 permet l'initialisation et la mise à jour de la configuration des modalités d'exécution du mécanisme de *polling* :

- A l'initialisation du *polling*, il permet la récupération des valeurs des paramètres stockées ainsi que leur prise en compte en installant le comportement d'exécution attendu. En particulier, le *polling* est exécuté selon un mode d'exécution séquentiel ou concurrent.
- Durant le déroulement du *polling*, le processus est à l'écoute des demandes de variation des valeurs de paramètres de configuration, puis, à leur arrivée, les répercute de façon à ajuster le comportement du *polling*.

Dans le cas d'une reconfiguration, la spécification de la figure 1.12 fait apparaître deux cas différents :

- Le premier cas concerne un changement de mode d'exécution. Le comportement décrit indique

```
newtype T_ModeDExecution literals sequential, concurrent ; endnewtype T_ModeDExecution ;
newtype T_TypeDeBorne literals iterative, temporal, illimited ; endnewtype T_TypeDeBorne ;
newtype T_ModeDeConsultation literals refreshed, local ; endnewtype T_ModeDeConsultation ;
newtype T_ModeArretAutonome literals rateImp, thresholdSuccImp, off ; endnewtype T_ModeArretAutonome ;
newtype T_ContextPolling struct
  Duration DelaiAttenteMax ;
  Duration PeriodeDePolling ;
  T_TypeDeBorne TypeDeBorne ;
  Duration DureeAutorisee ;
  Integer NbOperationsMax ;
  T_ModeArretAutonome ModeArretAutonome ;
  Integer SeuilOPSuccImproductives ;
  Real TauxOPImproductives ;
  T_ModeDeConsultation ModeDeConsultation ;
  Integer CptOperationsInvoquees ;
  Integer CptOperationsImproductives ;
  Integer CptOperationsSuccvesImproductives ;
endnewtype T_ContextPolling;
newtype T_Target = String (type Charstring, literal emptystring) ; endnewtype T_Target;
```

FIGURE 1.11: Types de données

une première étape consistant à signaler la suspension au processus contrôleur couramment actif. Une seconde transition permet, à la réception d'un signal d'acquittement – comprenant les valeurs actuelles des compteurs d'opérations (le contexte opérationnel) – d'opérer, *via* l'envoi d'un signal, l'activation de l'autre processus contrôleur tout en lui transmettant le contexte et la configuration courants. Il s'agit de la phase de transfert de contexte.

- Le second cas regroupe le changement de valeur d'un des six autres paramètres de configuration. Cette reconfiguration s'opère par l'envoi d'un message contenant ces nouvelles valeurs au processus contrôleur actuellement actif.

Les variations à prendre en compte dans la reconfiguration sont les suivantes :

- Les contrôleurs de *polling* séquentiel et concurrent permettent respectivement de piloter un *polling* dont le mode d'exécution choisi est l'un des deux modes définis. De façon similaire, ils assurent le cadencement spécifique des occurrences de consultations ainsi que la gestion de l'arrêt de la série. L'un ou l'autre interagit avec un ou plusieurs processus (appelés *Poller Agents*) réalisant chacun, à un moment donné, une occurrence de consultation.
  Le contrôleur séquentiel (figure 1.13) est décrit autour de deux états : le premier correspond davantage à la phase de pré-invocation d'une consultation alors que le second se situe dans la post-invocation. Les prises de décision quant à la poursuite (e.g. mise à jour des compteurs, armement de la durée entre opérations) ou l'arrêt du processus de *polling* relèvent plutôt des transitions opérées à partir du second état, alors que les traitements initiaux, de reconfiguration, de suspension, ou de déclenchement par expiration de la période de *polling* sont associés aux transitions opérées à partir du premier état.
  La description du comportement du contrôleur concurrent (figure 1.14) est axée autour de la spécification d'un seul état. Les différents messages pouvant être reçus déclenchent les transitions permettant d'endosser les comportements similaires à ceux décrits dans le paragraphe précédent.

```
process PollingAdapter
  fpar ME, PERIOD, MAXTC, TB, NBOCMAX, MAXDURATION, MSA, SSI, TXI, MC, TARGET;
  dcl ModeDExecution T_ModeDExecution; dcl CtxteCourant T_PollingContext; dcl Cible T_Target;
  dcl NextInd, Ind Integer; dcl FirstTime Boolean; dcl StartTime Time; start; FirstTime = true;
  nextstate init;

  state init;
    input none provided FirstTime = true;
      task
        FirstTime := false, ModeDExecution := ME, CtxteCourant!PeriodeDePolling := PERIOD,
        CtxteCourant!DelaiAttenteMax := MAXTC, CtxteCourant!TypeDeBorne := TB,
        CtxteCourant!DureeAutorisee := MAXDURATION, CtxteCourant!ModeArretAutonome := MSA,
        CtxteCourant!SeuilOPSuccImproductives := SSI, CtxteCourant!TauxOPImproductives := TXI,
        CtxteCourant!ModeDeConsultation := MC, CtxteCourant!NbOperationsMax := NBOCMAX,
        CtxteCourant!NbOperationsInvoquees := 0, CtxteCourant!NbOperationsSuccImproductives := 0,
        CtxteCourant!NbOperationsImproductives :=0, Cible := TARGET;
        NextInd := 1;
        StartTime := now;
        decision ModeDExecution;
          (sequential):
            task output(Activate(CtxteCourant, Cible, NextInd), SequentialPollerControler),
            nexstate reconfiguration;
          (concurrent):
            task output(Activate(CtxteCourant, Cible, NextInd), ConcurrentPollerControler),
            nexstate reconfiguration;

  state reconfiguration;
    input ModifyExecutiveMode(NewMode) NewMode /= ModeDExecution;
      task
        decision ModeDExecution;
          (sequential) :
            task output(Suspend, SequentialPollerControler), nexstate waitContext;
          (concurrent) :
            task output(Suspend, ConcurrentPollerControler), nexstate waitContext;
    input ModifyValuableParameter(NewPeriodeDePolling, NewDelaiAttenteMax, NewDureeAutorisee,
      NewNbOperationsMax, NewSeuilOPSuccImproductives, NewTauxOPImproductives);
      task
        CtxteCourant!PeriodeDePolling := NewPeriodeDePolling,
        CtxteCourant!DelaiAttenteMax := NewDelaiAttenteMax,
        CtxteCourant!DureeAutorisee := NewDureeAutorisee,
        CtxteCourant!SeuilOPSuccImproductives := NewSeuilOPSuccImproductives,
        CtxteCourant!TauxOPImproductives := NewTauxOPImproductives,
        CtxteCourant!NbOperationsMax := NewNbOperationsMax,
        decision ModeDExecution ;
          (sequential) :
            task output(Modify(CtxteCourant), SequentialPollerControler), nexstate ReConfiguration ;
          (concurrent) :
            task output(Modify(CtxteCourant), ConcurrentPollerControler), nexstate ReConfiguration ;
    input StopPolling ;
  stop;

  state waitContext ;
    input AckSuspend(NbOperationsInvoquees, NbOperationsSuccImproductives, NbOperationsImproductives,
      Ind);
      task
        CtxteCourant!CptOperationsInvoquees := NbOperationsInvoquees,
        CtxteCourant!CptOperationsSuccImproductives := NbOperationsSuccImproductives,
        CtxteCourant!CptOperationsImproductives := NbOperationsImproductives,
        CtxteCourant!DureeTotMax := CtxteCourant!DureeTotMax - (now - StartTime), NextInd = Ind ;
        decision ModeDExecution ;
          (sequential) :
            task ModeDExecution := concurrent,
            output(Activate(CtxteCourant, Cible, NextInd), ConcurrentPollerControler),
            nexstate reconfiguration ;
          (concurrent) :
            task ModeDExecution := sequential,
            output(Activate(CtxteCourant, Cible, NextInd), SequentialPollerControler),
            nexstate reconfiguration ;
    input StopPolling ;
  stop;
endprocess PollingAdapter;
```

FIGURE 1.12: Extrait de la description SDL du *poller* adaptable

- Le *poller agent* (figure 1.15) représente le comportement basique associé à l'exécution d'une opération de consultation de la cible et à sa gestion. Il signale au processus le contrôlant la « productivité » de l'opération réalisée.

1.2.2.2 Algorithmes concernant le mécanisme d'*event reporting*

De la même façon que pour le *polling*, on peut spécifier des comportements qui permettent de paramétrer des *listeners* de notifications pouvant détecter l'une ou l'autre des situations que l'on souhaite observer.

A titre d'illustration, deux algorithmes sont ici proposés pour exprimer à l'aide de SDL les logiques de détection de silence (figure 1.16) et de détection de retard/avance dans la réception de notifications (figure 1.17).

1.2.3 Récapitulatif

Le cœur de la capacité de configurabilité a été définie et spécifiée de manière générique pour permettre de contrôler le comportement des mécanismes de surveillance. Cette capacité de configurabilité est extensible (de nouveaux paramètres de configuration peuvent être ajoutés au mécanisme) et est modulaire. La section suivante en présente une implémentation possible : d'une part, la configuration stockée au sein d'un serveur CIM ; d'autre part, l'implémentation des mécanismes de surveillance au moyen de programmes Java.

1.3 Implémentation

Le module de configurabilité contient l'ensemble des caractéristiques des mécanismes de surveillance (*polling* et *event reporting*) ainsi que les éléments qui leurs sont relatifs (statistiques, configurations, cibles, etc.). Les mécanismes sont représentés à l'aide de classes CIM, stockées sur un serveur et comportant toutes les informations de configuration utiles à leur exécution. La configuration des mécanismes et toutes les informations qui leur sont relatives sont stockées au sein du *CIM repository* de l'outil libre Open Pegasus [OpenPegasus].

Par ailleurs, les algorithmes ont permis de développer en Java un *poller* et un *listener* adaptables, ayant pour vocation d'appliquer cette configuration, récupérée sur le serveur CIM, aux mécanismes de surveillance. Le comportement des mécanismes de surveillance doit ainsi être entièrement dépendant de la configuration indiquée.

L'implémentation du stockage et de l'instrumentation du module de configurabilité fait l'objet de ce chapitre.

1.3.1 La configurabilité : une implémentation en CIM

Lorsqu'un mécanisme de surveillance comme le *polling* est dit configurable, cela signifie que ses paramètres de configuration peuvent être mis à jour pour influencer son comportement. Les mécanismes de surveillance n'apparaissant pas dans les schémas CIM standards du DMTF, une extension de CIM a été conçue pour capturer l'information de gestion du *polling* et de l'*event reporting*. En effet, en utilisant une architecture de ce type, le même modèle peut-être utilisé pour modéliser ce qui est surveillé (la portée au travers des éléments gérés, qui peuvent être des cibles

```
process SequentialPollerControler 1,1 ;
  dcl PollingConfiguration, PC, T_PollingContext; dcl Cible T_Target; dcl NextInd, Ind Integer;
  timer DureeTotMax, DureeInterOP;
  start ;
  nextstate waitNextInvoke ;

  state waitNextInvoke ;
    task
      input Activate(PollingConfiguration, Cible, Ind);
        PC := PollingConfiguration ;
        decision PC!TypeDeBorne = temporal ;
          (true) : set(now + PC!DureeAutorisee, DureeTotMax(self)) ;
        create PollerAgent() ;
        output(OrderNextOP(Cible(Ind), PC!DelaiAttenteMax, PC!ModeDeConsultation), offspring) ;
        PC!CptOperationsInvoquees := PC!CptOperationsInvoquees + 1,
        NextInd := if (Ind + 1) >= Cible!Length then Ind + 1 else 1 ;
        nextstate waitResp ;
      input DureeInterOP ;
        task
          output(OrderNextOP(Cible(NextInd), PC!DelaiAttenteMax, PC!ModeDeConsultation), offspring) ;
          PC!CptOperationsInvoquees := PC!CptOperationsInvoquees + 1,
          NextInd := if (Ind + 1) >= Cible!Length then Ind + 1 else 1 ;
        nextstate waitResp ;
      input DureeTotMax;
        task
          output(OrderStop,offspring),
        stop;
      input Suspend ;
        task
          reset(DureeTotMax), reset(DureeInterOP), output(OrderStop,offspring),
          output(AckSuspend(PC!CptOperationsInvoquees, PC!CptNbOperationsSuccImproductives,
          PC!CptNbOperationsImproductives, NextInd),
        nextstate waitNextInvoke;
      input Modify(PollingConfiguration) ;
        task

  state waitResp ;
    input OPProd ;
      task
        PC!CptOperationsSuccImproductives := 0 ;
        decision (PC!TypeDeBorne = iterative) and (PC!CptOperationsInvoquees >= PC!NbOperationsMax)
          (true) : task output(OrderStop,offspring) ; stop ;
          (false) : task set(now + PC!PeriodeDePolling, DureeInterOP(self)) ;
      nextstate waitNextInvoke ;
    input OPImp ;
      task
        PC!CptOperationsSuccImproductives := PC!CptOperationsSuccImproductives + 1 ;
        PC!CptOperationsImproductives := PC!CptOperationsImproductives + 1 ;
        decision (PC!ModeArretAutonome) ;
          (rateImp) : decision (PC!CptOperationsImproductives / PC!CptOperationsInvoquees) * 100 >=
            PC!TauxOPImproductives ;
            (true) : task output(OrderStop, sender), stop ;
            (false) : task
              decision (PC!TypeDeBorne = iterative) and (PC!CptOperationsInvoquees >= PC!NbOperationsMax)
                (true)  : task output(OrderStop,offspring), stop ;
                (false) : task set(now + PC!PeriodeDePolling,DureeInterOP(self)),nextstate waitNextInvoke;
          (thresholdSuccImp) : decision(PC!CptOperationsSuccImproductives >= PC!SeuilOPSuccImproductives);
            (true) : task output(OrderStop, sender), stop ;
            (false) : task
              decision (PC!TypeDeBorne = iterative) and (PC!NbOperationsInvoquees >= PC!NbOperationsMax)
                (true) : task output(OrderStop,offspring), stop ;
                (false) : task set(now + PC!PeriodeDePolling,DureeInterOP(self)),nextstate waitNextInvoke;
         (off) : decision(PC!TypeDeBorne = iterative) and (PC!NbOperationsInvoquees >= PC!NbOperationsMax)
            (true) : task output(OrderStop,offspring), stop ;
            (false) : task set(now + PC!PeriodeDePolling, DureeInterOP(self)), nextstate waitNextInvoke ;
    input DureeTotMax;
    task output(OrderStop,offspring), stop;
    save Suspend, Modify(PollingConfiguration) ;
endprocess SequentialPoller
```

FIGURE 1.13: Extrait de la description SDL du *poller* adaptable en mode séquentiel

et/ou sources) et comment ils le sont (les paramètres de configuration). Grâce à ce point de vue informationnel, il est aussi possible de définir un premier cadre architectural basé sur WBEM, duquel

```
process ConcurrentPollerControler 1,1 ;
  dcl PollingConfiguration, PC, T_PollingContext ; dcl Cible T_Target ; dcl NextInd, Ind Integer ;
  timer DureeTotMax, DureeInterOP ;
  start ;
  nextstate wait ;

  state wait ;
    input Activate(PollingConfiguration, Cible, Ind) ;
      task PC := PollingConfiguration, decision PC!TypeDeBorne = temporal,
        (true) : set(now + PC!DureeAutorisee, DureeTotMax(self)) ;
          create PollerAgent(),
          output(OrderNextOP(Cible(Ind), PC!DelaiAttenteMax, PC!ModeDeConsultation), offspring) ;
          PC!CptOperationsInvoquees := PC!CptOperationsInvoquees + 1,
          NextInd := if (Ind + 1) >= Target!Length then Ind + 1 else 1,
          nextstate wait ;
    input DureeInterOP ;
      task
        create PollerAgent(),
        output(OrderNextOP(Cible(Ind), PC!DelaiAttenteMax, PC!ModeDeConsultation), offspring) ;
        set(now + PC!PeriodeDePolling, DureeInterOP(self)),
        PC!CptOperationsInvoquees := PC!CptOperationsInvoquees + 1,
        NextInd := if (Ind + 1) >= Target!Length then Ind + 1 else 1,
        nextstate wait ;
    input OPProd ;
      task output(OrderStop, sender), PC!CptOperationsSuccImproductives := 0,
      decision (PC!TypeDeBorne = iterative) and (PC!CptOperationsInvoquees >= PC!NbOperationsMax)
        (true) : task output(OrderStop, sender), stop ;
        (false) : nextstate wait ;
    input OPImp ;
      task output(OrderStop, sender) ;
      PC!CptOperationsSuccImproductives := PC!CptOperationsSuccImproductives + 1 ;
      PC!CptOperationsImproductives := PC!CptOperationsImproductives + 1 ;
      decision (PC!ModeArretAutonome) ;
        (rateImp) : decision (PC!CptOperationsImproductives / PC!NbOperationsInvoquees) * 100 >=
          PC!TauxOPImproductives ;
          (true) : task output(OrderStop, offspring), stop ;
          (false) : task
            decision (PC!TypeDeBorne = iterative) and (PC!CptOperationsInvoquees >= PC!NbOperationsMax)
              (true) : task output(OrderStop,offspring),stop ;
              (false) : task
              nextstate wait ;
        (thresholdSuccImp) : decision (PC!NbOperationsSuccImproductives >= PC!SeuilOPSuccImproductives) ;
          (true) : task output(OrderStop, offspring), stop ;
          (false) : task
            decision  (PC!TypeDeBorne = iterative) and (PC!CptOperationsInvoquees >= PC!NbOperationsMax)
              (true): task output(OrderStop,offspring), stop ;
              (false) : task
              nextstate wait ;
        (off): decision (PC!TypeDeBorne = iterative) and (PC!NbOperationsInvoquees >= PC!NbOperationsMax)
          (true)  : task output(OrderStop,offspring), stop ;
          (false) : task
          nextstate wait ;
    input DureeTotMax;
      task output(OrderStop, sender), output(OrderStop,offspring), stop ;
    input Suspend ;
      task
        reset(DureeTotMax), reset(DureeInterOP), output(OrderStop,offspring),
        output(AckSuspend(PC!CptOperationsInvoquees, PC!CptNbOperationsSuccImproductives,
        PC!CptNbOperationsImproductives, NextInd),
        nextstate wait ;
    input Modify(PollingConfiguration) ;
      task PC := PollingConfiguration,
      nextstate wait ;
endprocess ConcurrentPoller ;
```

FIGURE 1.14: Extrait de la description SDL du *poller* adaptable en mode concurrent

pourra être dérivé une implémentation d'activités de surveillance reconfigurable.

1.3.1.1 Présentation fonctionnelle

Les mécanismes sont rendus configurables sur deux plans :

```
process PollerAgent ;
  dcl DureeOPMax Duration ; dcl ModeDeConsultation T_ModeDeConsultation ; dcl EltCible Charstring ;
  timer TimeoutOP ;
  start ;
  task nextstate invoke ;

  state invoke ;
    input OrderNextOP(EltCible, DureeOPMax, ModeConsultation) ;
      task set(now + DureeOPMax, TimeoutOP) ;
      decision ModeDeConsultation ;
        (refreshed) : ouput InvokeOP(Refresh, EltCible) ;
        (local) : output InvokeOP(Local, EltCible) ;
        nextstate waitResp ;
    input OrderStop ;
      stop;

  state waitResp ;
    input ConsultationResponse ;
      task reset(DureeOPMax), output(OPProd, parent), nexstate invoke ;
    input TimeoutOP ;
      task output(OPImp, parent), nexstate invoke ;
    input OrderStop ;
      stop;
endprocess ConcurrentPoller ;
```

FIGURE 1.15: Extrait de la description SDL d'un processus de consultation

```
dcl MaxDelayWait Duration ;
timer SilentTimeout;
/* ... */
start ;
task nextstate wait ;

state wait ;
  input Indication(InfoIndication) ;
    task output Indication(InfoIndication) ;
    set(now + MaxDelayWait, SilentTimeout) ;
    nextstate wait ;
  input SilentTimeout ;
    task output SilentDetection ;
/* ... */
```

FIGURE 1.16: Extrait de la description SDL d'un processus de détection de silence

- D'un point de vue « plan de contrôle », sont définis les caractéristiques des mécanismes de surveillance (*polling* et *event reporting*) ainsi que les éléments qui leur sont relatifs (statistiques, configurations, cibles, etc.). La configuration est stockée au sein d'instances CIM de mécanismes de surveillance et peut être récupérée dynamiquement par le plan opérationnel.
- D'un point de vue « plan opérationnel », le prototype est chargé de récupérer la configuration d'un mécanisme de surveillance et d'exécuter celui-ci conformément aux modalités spécifiées dans cette configuration. Si la configuration du mécanisme est modifiée manuellement au sein des instances CIM, le plan opérationnel en est automatiquement averti, récupère la nouvelle configuration et adapte immédiatement son comportement selon ses modalités.

1.3.1.2 Informations de gestion de la configurabilité des mécanismes de surveillance

Le schéma CIM de la figure 1.18 définit les classes CIM nécessaires pour définir la configurabilité des mécanismes de surveillance. Les classes grisées représentent l'extension proposée au modèle CIM. Les autres classes CIM sont natives, et seules les propriétés utilisées dans le développement du prototype sont indiquées.

```
dcl Period, Epsilon Duration ;
timer MinDelayTimeout, MaxDelayTimeout ;
/* ... */
start ;
task nextstate init ;

state init ;
  input Indication(InfoIndication) ;
    task output Indication(InfoIndication) ;
    set(now + Period - Epsilon, MinDelayTimeout) ;
    nextstate waitMin ;

state waitMin ;
  input Indication(InfoIndication) ;
    task output IsochronousLossDetection ;
  input MinDelayTimeout ;
    set(now + (2 * Epsilon), MaxDelayTimeout) ;
    nexstate waitMax ;

state waitMax ;
  input Indication(InfoIndication) ;
    task output Indication(InfoIndication) ;
    set(now + Period - Epsilon, MinDelayTimeout) ;
    nextstate waitMin ;
  input MaxDelayTimeout ;
    output IsochronousLossDetection ;
/* ... */
```

FIGURE 1.17: Extrait de la description SDL d'un processus de détection de perte d'isochronisme

La classe `IRIT_MonitoringService` représente un service de surveillance abstrait ; elle apporte une classe mère commune à différents types de services de surveillance.

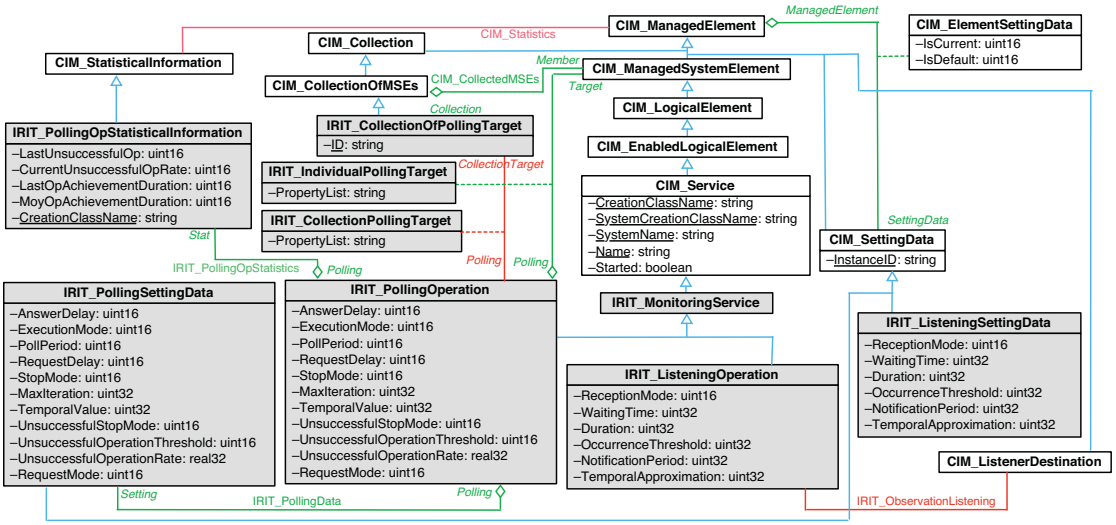

FIGURE 1.18: Intégration des mécanismes de surveillance en CIM

1.3.1.2.1 Vision gestion du mécanisme de *polling*

La classe `IRIT_PollingOperation` représente une fonctionnalité de *polling* pouvant être gérée ; chaque propriété de la classe est un paramètre pouvant être utilisé pour configurer le comportement du mécanisme de *polling* exécuté. La classe comprend l'ensemble des propriétés définies dans le chapitre 1.2 :

- Le mode d'exécution `ExecutionMode` d'une opération de *polling* : lorsqu'il est périodique (c'est-

à-dire que le non-entrelacement des requêtes n'est pas garanti), alors la propriété `PollPeriod` (l'intervalle de temps entre le début de deux requêtes de *polling* consécutives) est obligatoire. Dans le cas d'un mode sans entrelacement, la propriété `RequestDelay` (l'intervalle de temps entre la fin de la requête de *polling* i et le début de la requête de *polling* $i+1$) est obligatoire.
- Le temps d'attente maximum de chaque requête de *polling* `AnswerDelay` signifie que si le *manager* n'a pas reçu une réponse dans cet intervalle de temps, la requête est considérée improductive.
- Le mode de requête `RequestMode` d'une opération de *polling* vaut `source` quand l'information de gestion est directement produite et retournée par l'agent sur la cible, ou `local` quand la collecte est réalisée par le biais d'une entité intermédiaire.
- Le mode d'arrêt `StopMode` correspond à la manière dont le *polling* s'arrête. Le *polling* peut être soit illimité, soit limité. Dans le cas où il est limité, deux types de conditions sont à prendre en considération : un nombre maximum d'occurrences de *polling* (la propriété `MaxIteration` est obligatoire) ou une durée totale du *polling* (la propriété `TemporalValue` est obligatoire).
- Le mode d'arrêt autonome `UnsuccessfulStopMode` permet au mécanisme de *polling* de s'arrêter de manière autonome dans le cas où le nombre de requêtes de *polling* improductives dépasse un seuil (la propriété `UnproductiveOperationThreshold` est obligatoire) ou un taux (la propriété `UnproductiveOperationRate` est obligatoire) préalablement spécifié. Autrement, le *polling* continue jusqu'à une date d'expiration (déduite des paramètres définis par le mode classique d'arrêt) ou jusqu'à une demande externe d'arrêt du *polling*.

Le schéma propose également de spécifier une ou plusieurs cibles pour le service de *polling*. Ces cibles sont supposées déjà existantes sur le système, et doivent être associées au mécanisme de *polling* correspondant pour permettre une consultation plus rapide :

- L'association `IRIT_IndividualPollingTarget` permet d'agréger une ou plusieurs cibles individuelles de type `CIM_ManagedSystemElement` pour une opération de *polling*. L'association contient une propriété `PropertyList` permettant d'indiquer les propriétés particulières sur lesquelles l'opération de *polling* devrait s'appliquer.
- La classe `IRIT_CollectionOfPollingTarget` permet d'assembler des cibles de *polling*, principalement pour des questions d'identification. Une clef `ID` lui a été ajoutée pour permettre de créer une collection de cibles. L'association `IRIT_CollectionPollingTarget` permet d'établir une relation entre une opération de *polling* `IRIT_PollingOperation` et une collection de cibles `IRIT_CollectionOfPollingTarget`. L'association contient une propriété `PropertyList` permettant d'indiquer les propriétés particulières sur lesquelles l'opération de *polling* devrait s'appliquer.

De plus, pour respecter les « bonnes pratiques » de la modélisation en CIM, des extensions ont été déterminées pour gérer les configurations et les statistiques :

- La classe `IRIT_PollingSettingData` dérive de la classe `CIM_SettingData` et est composée des mêmes propriétés que la classe `IRIT_PollingOperation`. Elle permet de spécifier les différents types de configuration applicables (par défaut, courante, autre configuration possible) pour une opération de *polling*. L'association `IRIT_PollingData` permet d'agréger ces configurations applicables `IRIT_PollingSettingData` dans l'opération de *polling* `IRIT_PollingOperation`.
- La classe `IRIT_PollingOpStatisticalInformation` regroupe un ensemble de statistiques utiles concernant un mécanisme de *polling*. L'association `IRIT_PollingOpStatistics` permet

d'agréger un ensemble de statistiques dans un `IRIT_PollingOperation`.

Notons que cette classe peut être utilisée pour la gestion de l'observabilité d'un *polling* et être exploitée pour des besoins d'auto-optimisation.

1.3.1.2.2 Vision gestion du mécanisme d'*event reporting*

Même si la plupart sont expérimentales, les classes définies dans le schéma `CIM_Event` présentent tous les paramètres permettant de déterminer comment les notifications devraient être produites, filtrées et délivrées au consommateur lorsque celui-ci s'y est abonné. Nous ne sommes pas intervenu sur cet existant du standard non encore stabilisé.

Ainsi, le but de la classe `IRIT_ListeningOperation` est de définir un type d'observation (*listening*) pour simplement surveiller comment temporellement sont reçus ou non les notifications et événements. En effet, la fréquence de réception peut être pertinente afin de détecter tout problème survenant sur le système géré. Le mode de réception `ReceptionMode` a été déterminé pour sélectionner le critère d'observation de la fréquence de réception des notifications :

- Le mode de réception est positionné à `silence` pour vérifier si aucune notification n'est reçue pendant une certaine durée `WaitingTime`. Cette observation peut mener à deux interprétations possibles : le système fonctionne correctement et aucune alerte n'a été levée ; ou, une (ou plusieurs) notification(s) aurait(aient) dû être reçue(s) mais n'est(ne sont) jamais arrivée(s), d'où la conclusion qu'un problème a dû survenir sur le système.
- Le mode de réception est positionné à `burst` pour repérer une quantité de notifications supérieure à un seuil donné `OccurrenceThreshold` pendant une durée donnée `Duration`. Cette observation peut indiquer qu'un composant du système est dégradé ou en panne.
- Enfin, le mode de réception est positionné à `heartbeat` pour vérifier que les notifications sont bien reçues selon une périodicité `NotificationPeriod` définie avec un décalage temporel `TemporalApproximation` autorisé.

L'association `IRIT_ObservationListening` permet de lier nos extensions aux classes standardisées dans le schéma `CIM_Event`.

1.3.2 Test de fonctionnement

L'exécution opérationnelle des mécanismes de surveillance doit être réalisée conformément à la configuration stockée sur le serveur. Deux programmes ont été développés en Java pour permettre respectivement d'exécuter opérationnellement les mécanismes de *polling* et d'*event reporting*. Ces programmes Java ont été développés à l'aide de l'API SBLIM afin de leur attribuer le rôle de clients CIM pouvant communiquer avec le serveur CIM et donc avec la configuration qui y est stockée.

Le *poller* adaptable a pour fonctions de :

- récupérer les informations de configuration d'une opération de *polling* sur le serveur CIM lors de son lancement,
- interroger la cible sur le serveur CIM et ce selon le comportement dicté par les paramètres de configuration préalablement récupérés,
- adapter le comportement du *polling* en cours d'exécution si sa configuration a été modifiée : techniquement, cette adaptation du comportement est déclenchée par la réception d'un message de notification d'adaptation.

Le *listener* adaptable a pour fonctions de :

- attendre les notifications,
- indiquer si la réception des notifications effective est conforme au mode de réception indiqué,
- adapter le comportement du *listener* en cours d'exécution si sa configuration a été modifiée : techniquement, cette adaptation du comportement est déclenchée par la réception d'un message de notification d'adaptation.

1.3.3 Exemples de fonctionnement

A l'issue de cette implémentation, le prototype obtenu est composé d'un *poller* et d'un *listener* utilisables en l'état.

1.3.3.1 *Poller* adaptable

Pour exécuter un *polling*, il faut lancer le programme de *poller* adaptable en lui précisant quelle configuration de *polling* appliquer. Pour ce faire, il faut indiquer au *poller* adaptable l'identifiant de la configuration de *polling* contenue sur le serveur CIM. Considérons l'instance de configuration de *polling* suivante :

IRIT_PollingOperation	
CreationClassName	IRIT_PollingOperation
Name	CfgSeq1
SystemCreationClassName	AFDX_EndSystem
SystemName	127.0.0.1
ExecutionMode	2
PollPeriod	
RequestDelay	5000
AnswerDelay	2000
RequestMode	1
StopMode	2
MaxIteration	20
TemporalValue	
UnsuccessfulStopMode	1
UnsuccessfulOperationThreshold	
UnsuccessfulOperationRate	
Started	true

FIGURE 1.19: Instance CIM de configuration d'un *polling*

Cette instance est ici vue en dehors du schéma CIM. Celui-ci devrait être représenté dans son intégralité pour faire apparaître, notamment, les cibles que le *poller* doit interroger : `AFDX_EndSystem` et `AFDX_EndSystem333`. Toutefois, la reconstitution sur papier de l'ensemble des instances CIM impliquées prendrait ici une place trop importante et n'est pas nécessaire à la compréhension de ce paragraphe et aux conclusions que l'on souhaite en tirer.

La figure 1.20 présente une trace d'exécution du *poller* adaptable appliquant cette configuration de *polling*.

Il peut y être observé que l'ensemble des paramètres de configuration ont été pris en compte par le *poller*. La deuxième ligne de la figure 1.20 représente les valeurs des paramètres de configuration selon la forme suivante :

```
AnswerDelay : ExecutionMode : PollPeriod : RequestDelay : StopMode : MaxIteration :
      TemporalValue : UnsuccessfulStopMode : UnsuccessfulOperationThreshold :
                  UnsuccessfulOperationRate : RequestMode
```

```
|DONE| Connected to CIM server root/AFDX@192.168.56.101:5988
|DONE| Initial configuration uploaded: 2000:2:0:5000:2:20:0:1:0:0.0:1
|INFO| Polling execution mode = no overlapping
|INFO| Adaptation request listening
REQUEST n° 1
|INFO| Total Duration needed for Answer: 211
+ Target path: root/AFDX:AFDX_EndSystem.CreationClassName="AFDX_EndSystem",Name="127.0.0.1"
EnabledState: 1
RequestedState: 12
REQUEST n° 2
|INFO| Total Duration needed for Answer: 38
+ Target path: root/AFDX:AFDX_EndSystem.CreationClassName="AFDX_EndSystem333",Name="127.0.0.1"
EnabledState: 3
RequestedState: 12
[...sortie écran tronquée pour plus de visibilité...]
REQUEST n° 19
|INFO| Total Duration needed for Answer: 33
+ Target path: root/AFDX:AFDX_EndSystem.CreationClassName="AFDX_EndSystem",Name="127.0.0.1"
EnabledState: 1
RequestedState: 12
REQUEST n° 20
|INFO| Total Duration needed for Answer: 27
+ Target path: root/AFDX:AFDX_EndSystem.CreationClassName="AFDX_EndSystem333",Name="127.0.0.1"
EnabledState: 3
RequestedState: 12
|DONE| Max iteration number reached: polling termination
```

FIGURE 1.20: Fonctionnement du *poller* adaptable

Ainsi, il est possible d'observer l'application des paramètres de configuration suivants :

- le mode d'exécution (`ExecutionMode`) était indiqué comme « non entrelacé » (2) sur l'instance CIM ; ce mode d'exécution a bien été respecté par le *poller* adaptable qui a bien récupéré la bonne configuration sur le serveur CIM (troisième ligne sur la figure 1.20),
- le *polling* est borné par un nombre maximum d'itérations (`MaxIteration`) positionné à `20`, ce qui est conforme à la valeur du paramètre `MaxIteration` indiqué dans l'instance CIM,
- le temps d'attente maximum de la réponse (`AnswerDelay`) étant positionné à 2s, toutes les requêtes sont productives (elles ont nécessité entre 27 et 211ms pour être exécutées)...

1.3.3.2 *Listener* adaptable

Le *listener* adaptable permet d'observer comment temporellement sont reçues les notifications. Soient les trois configurations d'*event reporting* de la figure 1.21.

IRIT_ListeningOperation	
CreationClassName	IRIT_EventReporting
Name	ERsilence
SystemCreationClassName	AFDX_EndSystem
SystemName	127.0.0.1
ReceptionMode	1
WaitingTime	120
Duration	
OccurrenceThreshold	
NotificationPeriod	
TemporalApproximation	

IRIT_ListeningOperation	
CreationClassName	IRIT_EventReporting
Name	ERburst
SystemCreationClassName	AFDX_EndSystem
SystemName	127.0.0.1
ReceptionMode	2
WaitingTime	
Duration	1400
OccurrenceThreshold	2
NotificationPeriod	
TemporalApproximation	

IRIT_ListeningOperation	
CreationClassName	IRIT_EventReporting
Name	ERheartbeat
SystemCreationClassName	AFDX_EndSystem
SystemName	127.0.0.1
ReceptionMode	3
WaitingTime	
Duration	
OccurrenceThreshold	
NotificationPeriod	2000
TemporalApproximation	45

FIGURE 1.21: Instances CIM de configuration d'*event reporting*

Dans la figure 1.22, le *listener* applique la première configuration (`ERsilence`) : l'*event reporting*

est positionné dans un mode de détection de silence (1) avec un temps d'attente maximum de la notification (`WaitingTime`) de 120 ms. On peut observer que le *listener* adaptable détecte plusieurs périodes de 120ms pendant lesquelles aucune notification ne lui parvient.

```
|DONE| Connected to CIM server root/AFDX@192.168.56.101:5988
|INFO| Silence detection mode
 ---> Current Configuration : 1:120:0:0:0:0
|INFO| Adaptation request listening
INIT SILENCE MODE   RECU *** --> INDICATION 4
ALL IS OK     RECU *** --> INDICATION 5
ALL IS OK     RECU *** --> INDICATION 6
ALL IS OK     RECU *** --> INDICATION 7
|INFO| A silence has been detected
INIT SILENCE MODE   RECU *** --> INDICATION 8
ALL IS OK     RECU *** --> INDICATION 9
ALL IS OK     RECU *** --> INDICATION 10
ALL IS OK     RECU *** --> INDICATION 11
ALL IS OK     RECU *** --> INDICATION 12
|INFO| A silence has been detected
INIT SILENCE MODE   RECU *** --> INDICATION 13
ALL IS OK     RECU *** --> INDICATION 14
ALL IS OK     RECU *** --> INDICATION 15
```

FIGURE 1.22: Fonctionnement du *listener* adaptable en mode détection de silence

Dans la figure 1.23, le *listener* applique la seconde configuration (`ERburst`) : l'*event reporting* est positionné dans un mode de détection de salves (2) sur une durée fixée à 1400ms et un seuil de deux occurrences. On peut observer que le *listener* adaptable détecte plusieurs salves de notifications conformément aux périodes définies.

```
|DONE| Connected to CIM server root/AFDX@192.168.56.101:5988
|INFO| Burst detection mode
 ---> Current Configuration : 2:0:1400:2:0:0
|INFO| Adaptation request listening
INIT HISTORY RECU *** --> INDICATION 0
INIT HISTORY RECU *** --> INDICATION 1
|INFO| A burst has been detected  RECU *** --> INDICATION 2
INIT HISTORY RECU *** --> INDICATION 3
ALL IS OK     RECU *** --> INDICATION 4
ALL IS OK     RECU *** --> INDICATION 5
|INFO| A burst has been detected RECU *** --> INDICATION 6
INIT HISTORY RECU *** --> INDICATION 7
ALL IS OK     RECU *** --> INDICATION 8
ALL IS OK     RECU *** --> INDICATION 9
|INFO| A burst has been detected RECU *** --> INDICATION 10
INIT HISTORY RECU *** --> INDICATION 11
```

FIGURE 1.23: Fonctionnement du *listener* adaptable en mode détection de salves

Enfin dans la figure 1.24, le *listener* applique la troisième configuration (`ERheartbeat`) : l'*event reporting* est positionné dans un mode de détection périodique (3) avec une période de 2s et avec un écart temporel autorisé de 45ms. On peut observer que le *listener* adaptable détecte des avances et des retards sur la réception prévue des notifications.

```
|DONE| Connected to CIM server root/AFDX@192.168.56.101:5988
|INFO| Periodicity loss detection mode
 ---> Current Configuration : 3:0:0:0:2000:45
|INFO| Adaptation request listening
INIT PERIODIC HISTORY      RECU *** --> INDICATION 0
ALL IS OK     RECU *** --> INDICATION 1
ALL IS OK     RECU *** --> INDICATION 2
ISOCHRONISM LOSS DETECTED (ADVANCE)     RECU *** --> INDICATION 3
ALL IS OK     RECU *** --> INDICATION 4
ALL IS OK     RECU *** --> INDICATION 5
ISOCHRONISM LOSS DETECTED (ADVANCE)     RECU *** --> INDICATION 6
ALL IS OK     RECU *** --> INDICATION 7
ISOCHRONISM LOSS DETECTED (DELAY)RECU *** --> INDICATION 8
ISOCHRONISM LOSS DETECTED (ADVANCE)     RECU *** --> INDICATION 9
ISOCHRONISM LOSS DETECTED (ADVANCE)     RECU *** --> INDICATION 10
ALL IS OK     RECU *** --> INDICATION 11
ISOCHRONISM LOSS DETECTED (DELAY)RECU *** --> INDICATION 12
ISOCHRONISM LOSS DETECTED (ADVANCE)     RECU *** --> INDICATION 13
```

FIGURE 1.24: Fonctionnement du *listener* adaptable en mode détection périodique

Les exemples peuvent se multiplier. De nombreux tests ont été réalisés pour vérifier que tous les paramètres de configuration sont bien pris en compte dans tous les cas (quelles que soient les cibles, en modifiant chacun des modes d'exécution, en modifiant les valeurs des paramètres). Ainsi, il faut se référer au chapitre 2 de la partie III pour observer des cas d'ajustements dynamiques du comportement des *poller* et *listener* adaptables.

1.4 Conclusion

Cette partie a porté sur la définition, la spécification, et l'implémentation du module de « configurabilité » visible en vert sur la figure 1.25.

Positionnons-nous sur le plan de contrôle du service de surveillance adaptable. Dans ce contexte, la capacité a tout d'abord fait l'objet d'une spécification semi-formelle des divers éléments la composant pour caractériser la configuration des mécanismes de surveillance dans son ensemble. Cette configuration a été ensuite modélisée en CIM, puis instrumentée dans le serveur CIM de l'outil Open Pegasus. CIM a été initialement choisi par rapport au contexte général de la gestion intégrée. Toutefois, il est possible d'utiliser une autre technologie pour stocker les informations de configuration. En outre, cette configuration est extensible et peut être étendue, mais représente une base correcte sur laquelle il est possible de s'appuyer pour démontrer la faisabilité de l'approche à ce niveau.

Plus à droite, au niveau du plan opérationnel du service de surveillance adaptable, on peut observer que cette configuration doit pouvoir être appliquée de manière opérationnelle sur les mécanismes de surveillance adaptable. Pour ce faire, deux programmes Java ont été développés comme des clients CIM : le *poller* adaptable, et le *listener* adaptable. Leur but est de récupérer la configuration sur le serveur CIM et de l'appliquer pour que leur comportement soit conforme à la configuration stockée. Ces deux programmes pourraient tout à fait être remplacés par d'autres outils : il serait possible d'envisager une instrumentation utilisant le protocole SNMP ou encore Netflow. La configurabilité est donc générique vers le bas et son utilisation ne demande qu'à être étendue

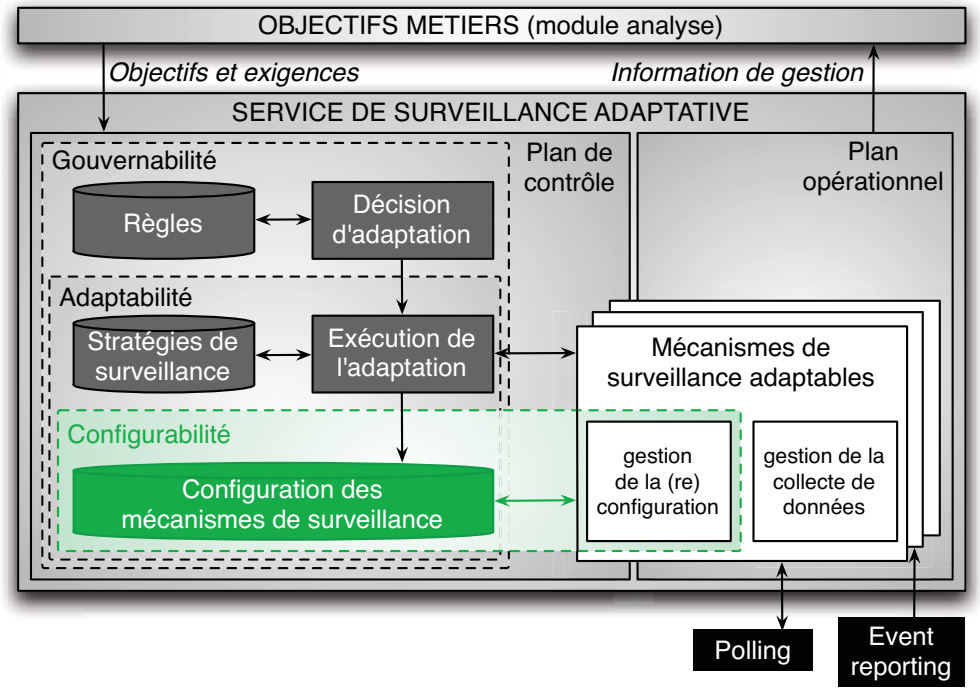

FIGURE 1.25: Le module de « configurabilité »

à d'autres technologies et d'autres protocoles. Une fois cette extension faite grâce aux algorithmes génériques qui ont été fournis, le passage de l'une à l'autre des technologies n'aura aucun surcoût de changement.

Ainsi, la spécification et l'implémentation proposées répondent à l'ensemble des besoins définis en première partie et ont permis de proposer un premier prototype permettant de rendre les mécanismes configurables initialement et en cours d'exécution, ce qui correspond bien à la définition de la « configurabilité » qui avait été faite :

Définition

La **configurabilité** est la capacité d'initialiser et de modifier dynamiquement et sans interruption la portée et les valeurs des paramètres gouvernant le comportement des mécanismes de surveillance.

La « configurabilité » doit désormais servir de base à l'élaboration de la capacité d'« adaptabilité », dont le but est d'offrir une interface pour ajuster dynamiquement la configuration stockée dans le serveur CIM, et donc finalement le comportement global des mécanismes de surveillance en cours d'exécution.

L'adaptabilité ou comment adapter la surveillance

2

Après avoir mis en place la configurabilité des mécanismes de surveillance, il est désormais possible d'étendre cette capacité en ajoutant la couche adaptabilité au *framework*. Ce chapitre formalise et conceptualise les divers opérateurs d'adaptation permettant de modifier la configuration et la portée des mécanismes de surveillance. Puis est présenté le prototype permettant d'adapter dynamiquement la configuration et la portée de la surveillance dans un contexte de gestion intégrée. Le prototype se basant sur la couche inférieure, nous avons été contraints à faire des choix techniques d'implémentation de cette couche.

2.1 Définition

La surveillance favorise la détection de toute situation de fonctionnement anormal du système. Cette détection d'anomalies peut déclencher une adaptation de la surveillance. Une définition de la surveillance adaptable en découle :

Définition

Rendre la **surveillance adaptable**, c'est doter l'activité de surveillance de tous les moyens qui lui sont nécessaires pour être reconfigurée dynamiquement.

L'adaptabilité de la surveillance s'appuie sur la couche configurabilité du *framework* proposé, dans le sens où elle permet d'automatiser la reconfiguration de la surveillance, de sorte à l'adapter pour des objectifs.

Il s'agit de la capacité à exécuter l'adaptation de la surveillance. Ainsi, cette couche intermédiaire permet d'agir sur l'état d'un ensemble de mécanismes de surveillance et sur le cycle de vie de chacun de ses éléments. Des opérateurs d'adaptation (ajout, suppression, modification, suspension, reprise) ont été spécifiés et implémentés de sorte à mener des actions de contrôle sur le cycle de vie des mécanismes.

La couche d'adaptabilité a été définie, conceptualisée, implémentée et testée. Son interface fait l'objet de ce chapitre.

2.2 Les opérateurs d'adaptation

Les classes présentées dans la partie précédente (configurabilité) ont été instrumentées au moyen de fichiers MOFs sur le produit Open Pegasus, utilisé ici comme simple serveur CIM. Les classes peuvent être instanciées, modifiées et supprimées, mais seulement manuellement. La couche supérieure, la couche adaptabilité, basée sur la configurabilité, doit être spécifiée pour automatiser la gestion des classes CIM créées.

Définition

L'**adaptabilité** est la capacité de modifier dynamiquement et sans interruption le comportement d'une activité de surveillance.

2.2.1 Définitions préliminaires

L'adaptabilité consiste donc essentiellement à fournir des opérateurs permettant d'ajouter, modifier et supprimer des mécanismes de surveillance de manière dynamique et sans entraver le comportement global du système, mais doit aussi permettre de modifier leur état opérationnel. Il faut introduire l'état opérationnel Op d'un mécanisme de surveillance, pour indiquer si le mécanisme est activé (*running*) ou suspendu (*pending*) :

$$Op = \{running|pending\}$$

L'introduction de l'état opérationnel d'un mécanisme de surveillance permet de définir une stratégie de surveillance comme un ensemble de mécanismes de surveillance, incluant leur type (comment surveiller ?), les éléments gérés sur lesquels ils s'appliquent (quoi surveiller ?), et le mode temporel sur lequel ils s'opèrent (quand surveiller ?). Comme cela peut être observé sur la figure 2.1, la stratégie S devient une composition entre les mécanismes M, leur configuration respective C et leur état opérationnel Op.

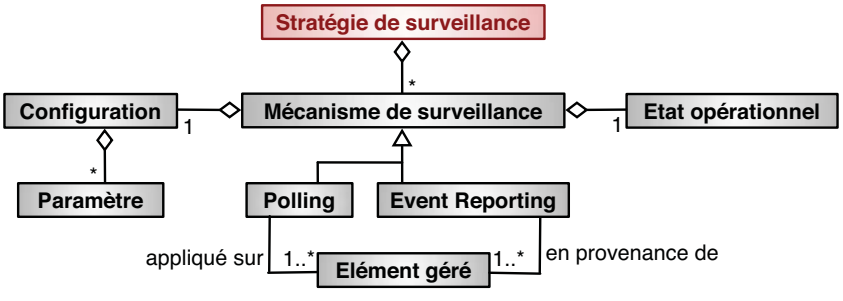

FIGURE 2.1: Adaptabilité de la surveillance

Soit S une stratégie de surveillance appliquée sur un ensemble d'éléments gérés T. Ainsi, $\forall\, n \in \mathbb{N}^{\star}$ et $\forall\, t_n \in T = \langle t_1, t_2, ..., t_n \rangle$,

$$S = \left(M_T, C, Op\right) = \left\{\left\langle M_{t_1}, C_1, Op_1\right\rangle, \left\langle M_{t_2}, C_2, Op_2\right\rangle, ..., \left\langle M_{t_n}, C_n, Op_n\right\rangle\right\}$$

L'évolution d'une stratégie S en une stratégie S', qui modifie l'état global de la surveillance, est un phénomène appelé « adaptation » [Chung04] :

$$\delta_S = \left(S \rightarrow S'\right)$$

... avec $S' = \left(M'_{T'}, C', Op'\right)$

Pour appliquer l'adaptation, la couche d'adaptabilité est nécessaire. Elle est constituée de cinq opérateurs basiques supportant la reconfiguration automatique et dynamique d'une activité de surveillance. En effet, si on veut appliquer cette définition, il nous faut des opérateurs qui agissent soit sur la constitution de l'ensemble, soit sur les valeurs d'un élément donc sur la constitution d'un triplet $\langle M_t, C, Op \rangle$.

2.2.2 Spécification des opérateurs

2.2.2.1 Opérateur d'adjonction

L'opérateur $\mathscr{A}$ définit l'adjonction d'un mécanisme de surveillance à la stratégie courante : il implique la création du mécanisme lui-même, de sa configuration et d'un état opérationnel indiquant si le mécanisme fonctionne ou non (l'état est ici *running*). Chaque mécanisme est indicé par i dans la suite. Ainsi, $\forall\, i \in \mathbb{N}^\star$ et $\forall\, t_i \in T'$,

$$\left(M_T, C, Op\right) \overset{\mathscr{A}(m_{t_i}, c_i)}{\longrightarrow} \left(M'_{T'}, C', Op'\right) \text{ avec } \left(M'_{T'}, C', Op'\right) = \left(M_T, C, Op\right) \cup \left\{m_{t_i}, c_i, running\right\}$$

Une fois créé, le mécanisme peut être lancé.

Les implications sont alors les suivantes :

- Un mécanisme de surveillance m_{t_i} est appliqué sur une cible t_i : si celle-ci n'existait pas dans l'ensemble des cibles T de la stratégie courante S, elle est ajoutée à l'ensemble qui devient le nouvel ensemble T' ; sinon, les ensembles des cibles de départ et d'arrivée sont les mêmes :

$$\left(\forall\, t_i \in T \Rightarrow T = T'\right) \vee \left(T \subset T'\right)$$

- Un mécanisme de surveillance m_{t_i} est ajouté à l'ensemble des mécanismes de surveillance M_T donc la cardinalité de l'ensemble est augmentée de 1 :

$$\left| M'_{T'} \right| = |M_T| + 1$$

- Un mécanisme de surveillance m_{t_i} est associé à une configuration particulière c_i : cette dernière est donc ajoutée à l'ensemble des configurations C, qui voit sa cardinalité augmentée de 1 :

$$\left| C' \right| = |C| + 1$$

2.2.2.2 Opérateur de suppression

L'opérateur $\mathscr{D}$ définit la suppression d'un mécanisme de surveillance de la stratégie initiale : il implique la suppression du mécanisme lui-même et de sa configuration. Ainsi, $\forall\, i \in \mathbb{N}^\star$ et $\forall\, t_i \in T'$,

$$\left(M_T, C, Op\right) \overset{\mathscr{D}(m_{t_i})}{\longrightarrow} \left(M'_{T'}, C', Op'\right) \text{ avec } \left(M'_{T'}, C', Op'\right) = \left(M_T, C, Op\right) - \left\{m_{t_i}, c_i, Op_i\right\}$$

Cette suppression implique un arrêt définitif du mécanisme.

Les implications sont alors les suivantes :

- Un mécanisme de surveillance m_{t_i} est appliqué sur une cible t_i : si cette cible existe par ailleurs dans l'ensemble des cibles T (un autre mécanisme, *polling* ou *event reporting* pouvant porter sur cette même cible), alors elle n'est pas supprimée de l'ensemble de cibles de départ (qui est identique à l'ensemble d'arrivée) ; sinon, elle est supprimée de l'ensemble des cibles qui devient T' :

$$\left(T = T' \Rightarrow \exists\, t_i | m_{t_i} \in M'_{T'}\right) \vee \left(T' \subset T\right)$$

- Un mécanisme de surveillance m_{t_i} est supprimé de l'ensemble des mécanismes de surveillance M_T donc la cardinalité de l'ensemble est diminuée de 1 :

$$\left| M'_{T'} \right| = |M_T| - 1$$

- Un mécanisme de surveillance m_{t_i} est associé à une configuration particulière c_i : cette dernière est donc supprimée de l'ensemble des configurations C, qui voit sa cardinalité diminuée de 1 :

$$\left| C' \right| = |C| - 1$$

2.2.2.3 Opérateur de modification

L'opérateur $\mathscr{U}$ définit la modification de la configuration ou de la portée d'un mécanisme de surveillance par rapport à la stratégie initiale. Ainsi, $\forall\, i \in \mathbb{N}^\star$ et $\forall\, t_i \in T'$,

$$\left(M_T, C, Op\right) \overset{\mathscr{U}(m_{t_i}, c_i)}{\longrightarrow} \left(M_{T'}, C', Op\right) \text{ avec } C \neq C' \vee T \neq T'$$

Les implications sont alors les suivantes :

- Modifier la configuration d'un mécanisme de surveillance m_{t_i} signifie modifier soit la cible t_i sur laquelle porte le mécanisme, soit ses paramètres de configuration C. Si l'ensemble des cibles de départ T et d'arrivée T' sont les mêmes, alors c'est que la configuration a changé ; sinon, c'est l'ensemble des cibles sur lesquelles porte le mécanisme de surveillance qui a changé :

$$\left(T = T' \Rightarrow C_f \neq C'\right) \vee \left(T \neq T' \Rightarrow C = C'\right)$$

- Bien que l'ensemble des cibles puisse être différent, le nombre de mécanismes de surveillance, lui, ne change pas :

$$\left| M'_{T'} \right| = |M_T|$$

- Bien que certains paramètres de configuration puissent être différents, le nombre de paramètres de configuration ne change pas :

$$\left| C' \right| = |C|$$

2.2.2.4 Opérateur de suspension

La suppression d'un mécanisme est définitive : s'il est nécessaire par la suite, il faut le recréer. De ce fait, il peut être intéressant de ne faire que suspendre un mécanisme pendant son exécution, et garder ainsi la possibilité de le relancer par la suite. Ainsi, $\forall\, i \in \mathbb{N}^\star$ et $\forall\, t_i \in T'$,

$$\left(M_T, C, Op\right) \overset{\mathscr{S}(m_{t_i})}{\longrightarrow} \left(M_T, C, Op'\right) \text{ avec } Op_i = \textit{pending}$$

Le mécanisme considéré ici serait plutôt un *polling* car dans le cas de l'*event reporting*, suppression et suspension correspondent à une seule et même action de désabonnement.

2.2.2.5 Opérateur de reprise

La remise en service d'un mécanisme de surveillance correspond à la réactivation d'un mécanisme précédemment suspendu. Ainsi, $\forall\, i \in \mathbb{N}^\star$ et $\forall\, t_i \in T'$,

$$\left(M_T, C, Op\right) \overset{\mathscr{R}\left(m_{t_i}\right)}{\longrightarrow} \left(M_T, C, Op'\right) \text{ avec } Op_i = running$$

2.2.3 Les interfaces des opérateurs

Les opérateurs ont été formalisés de sorte à pouvoir gérer les mécanismes de surveillance et leur cycle de vie. Leurs interfaces présentant les entrées/sorties nécessaires à l'implémentation des mécanismes ont également été définies, de sorte à répondre à notre besoin de *well-definition*.

Soient donc les ensembles de types suivants : un mécanisme de surveillance est soit un *polling*, soit un *event reporting* ; un état opérationnel est positionné à *running* ou *pending*, une configuration a un nom, une ou plusieurs cibles sont identifiées par un chemin d'accès, et un *event reporting* a la possibilité de s'abonner à une ou plusieurs sources de notifications.

Sachant que id correspond à l'identifiant du mécanisme sur lequel on opère, les opérateurs s'interfacent donc de la manière suivante avec ces types (figure 2.3) :

Pour ajouter un *polling*, seuls sont nécessaires la configuration et la cible du *polling* : la création produit alors l'identifiant du *polling* et son état opérationnel (à *running* obligatoirement). Pour supprimer, suspendre, reprendre un *polling*, seul l'identifiant du *polling* est nécessaire. Pour mettre à jour la configuration du *polling*, son identifiant et sa nouvelle configuration doivent être passées en paramètres. Enfin, pour mettre à jour la cible du *polling*, seuls sont nécessaires son identifiant et sa nouvelle cible.

Dans le cas de l'*event reporting*, l'ajout de ce type de mécanisme nécessite une configuration et un contexte d'abonnement : à quoi doit s'abonner le mécanisme (c'est l'équivalent du chemin du producteur des notifications). Pour supprimer un tel mécanisme, l'identifiant est nécessaire. Et enfin, pour mettre à jour la configuration d'un *event reporting*, il faut lui passer en paramètre son identifiant et sa nouvelle configuration.

2.2.4 Bilan

Cette formalisation nous offre donc une interface générique et *well-defined* pour chacun des opérateurs nécessaires à l'adaptation de la surveillance. Il est possible de les réutiliser quel que soit l'environnement technologique d'implémentation, simplement en les spécialisant et en les ajustant à l'environnement cible souhaité.

2.3 Implémentation

Les entrées, sorties et implications ayant été mises en évidence, la couche « adaptabilité » a été implémentée au moyen d'algorithmes correspondant aux opérateurs précédemment définis et permettant une intégration dans l'environnement CIM/WBEM. Ils concernent deux plans :

- le plan de contrôle correspond à la gestion des modifications dynamiques sur le modèle CIM (autrement dit, sur les classes et associations CIM correspondantes) défini au niveau de la couche « configurabilité »,

```
module adaptableMonitoring {

typedef enum operationalState {running, pending};
typedef pollingConfigurationPath string ;
typedef erConfigurationPath string;
typedef idMechanism string ;
typedef targetName string ;
typedef sequence <targetName> targetList;
typedef susbscriptionContextPath string;


interface PollingService {

        exception badConfigurationParameterException { string raison; long id; };
        exception badTargetException { string raison; long id; };

        // OPERATEUR d'ADJONCTION et de LANCEMENT d'un MECANISME de POLLING ADAPTABLE
        // @ ensures \result != null
        // @ ensures operationalState(\result) == running
        // @ signals badConfigurationParameterException : verifyValues(initial_config) == false
        // @ signals badTargetException : alreadyExists(initial_list) == false and create(initial_list) == false
        idMechanism addPolling(in configurationPath initial_config, in targetList initial_list)
                              raises (badConfigurationParameterException, badTargetException);

        // OPERATEUR de SUPRESSION et d'ARRET d'un MECANISME de POLLING
        // @ requires alreadyExists(id_poller) == true
        void deletePolling(in idMechanism id_poller);

        // OPERATEUR de SUSPENSION d'un MECANISME de POLLING
        // @ requires alreadyExists(id_poller) == true
        // @ requires operationalState(id_poller) == running
        // @ ensures  operationalState(id_poller) == pending
        void suspendPolling(in idMechanism id_poller);

        // OPERATEUR de REPRISE d'EXECUTION d'un MECANISME de POLLING
        // @ requires alreadyExists(id_poller) == true
        // @ requires operationalState(id_poller) == pending
        // @ ensures  operationalState(id_poller) == running
        void resumePolling(in idMechanism id_poller);

        // OPERATEUR de MODIFICATION des paramètres comportementaux d'un MECANISME de POLLING
        // @ requires alreadyExists(id_poller) == true
        // @ requires operationalState(id_poller) == running
        // @ ensures operationalState(id_poller) == running
        // @ signals badConfigurationParameterException : verifyValues(new_config) == false
        void updateConfigurationPolling(in idMechanism id_poller, in pollingConfigurationPath new_config)
                raises (badConfigurationParameterException);
```

FIGURE 2.2: Interfaces des opérateurs (partie 1)

- le plan opérationnel correspond à l'utilisation du modèle pour lancer l'exécution du mécanisme de surveillance enregistré dans le modèle CIM (autrement dit, le lancement du *poller* adaptable en respectant les informations de configuration du *polling* ou le lancement du *listener* adaptable en respectant les informations de configuration de l'*event reporting*).

Les algorithmes groupés dans cette partie sont ainsi à mettre en relation avec le modèle CIM de la figure 1.18. Ils ont été définis pour respecter en compte les différentes contraintes d'intégrité et de cohérence de données.

```
        // OPERATEUR d'AJOUT de cibles à un MECANISME de POLLING
        // @ requires alreadyExists(id_poller) == true
        // @ requires operationalState(id_poller) == running
        // @ ensures operationalState(id_poller) == running
        // @ signals badTargetException : alreadyExists(add_list) == false and create(add_list) == false
        void extendScopePolling(in idMechanism id_poller, in targetList add_list)
                raises (badTargetException);

        // OPERATEUR de SUPPRESSION de cibles d'un MECANISME de POLLING
        // @ requires alreadyExists(id_poller) == true
        // @ requires operationalState(id_poller) == running
        // @ ensures operationalState(id_poller) == running
        // @ signals badTargetException : alreadyExists(del_list) == false
        void reduceScopePolling(in idMechanism id_poller, in targetList del_list)
                raises (badTargetException);
        };

interface EventReportingService {

        exception badConfigurationParameterException { string raison; long id; };
        exception badSourceException { string raison; long id; };

        // OPERATEUR d'ADJONCTION et de LANCEMENT d'un LISTENER ADAPTABLE
        // @ ensures verifyValues(initial_config) == true
        // @ ensures \result != null
        // @ ensures operationalState(\result) == running
        // @ signals badConfigurationParameterException : verifyValues(initial_config) == false
        // @ signals badSourceException : Subscribe(source) == true
        idMechanism addEventReportingListener(in erConfigurationPath initial_config, in susbscriptionContextPath
source)
                                raises (badConfigurationParameterException, badSourceException);

        // OPERATEUR de SUPRESSION et d'ARRET d'un LISTENER ADAPTABLE
        // @ requires alreadyExists(id_ERlistener) == true
        void deleteEventReportingListener(in idMechanism id_ERlistener);

        // OPERATEUR de MODIFICATION des paramètres comportementaux d'un LISTENER ADAPTABLE
        // @ requires alreadyExists(id_ERlistener) == true
        // @ requires operationalState(id_ERlistener) == running
        // @ ensures operationalState(id_ERlistener) == running
        // @ signals badConfigurationParameterException : verifyValues(new_config) == false
        void updateConfigurationEventReportingListener(in idMechanism ERlistener, in erConfigurationPath
new_config)
                raises (badConfigurationParameterException);
        };
}
```

FIGURE 2.3: Interfaces des opérateurs (partie 2)

2.3.1 Implémentation CIM

L'adaptabilité des mécanismes de surveillance apporte les fonctionnalités suivantes :

- pour le *polling* : ajout, suppression, modification de la configuration et de la portée, suspension et reprise,
- pour l'*event reporting* : ajout, suppression, modification de la configuration.

L'ensemble de ces opérateurs sont visibles sur le diagramme de classe Java de la figure 2.4.

2.3.1.1 Adjonction

2.3.1.1.1 *Polling*

La figure 2.5 propose l'algorithme correspondant à l'ajout d'un mécanisme de *polling*.

Diverses contraintes et options sont positionnées :

- Il est possible de décider ou non de spécifier des valeurs pour les paramètres de configuration.

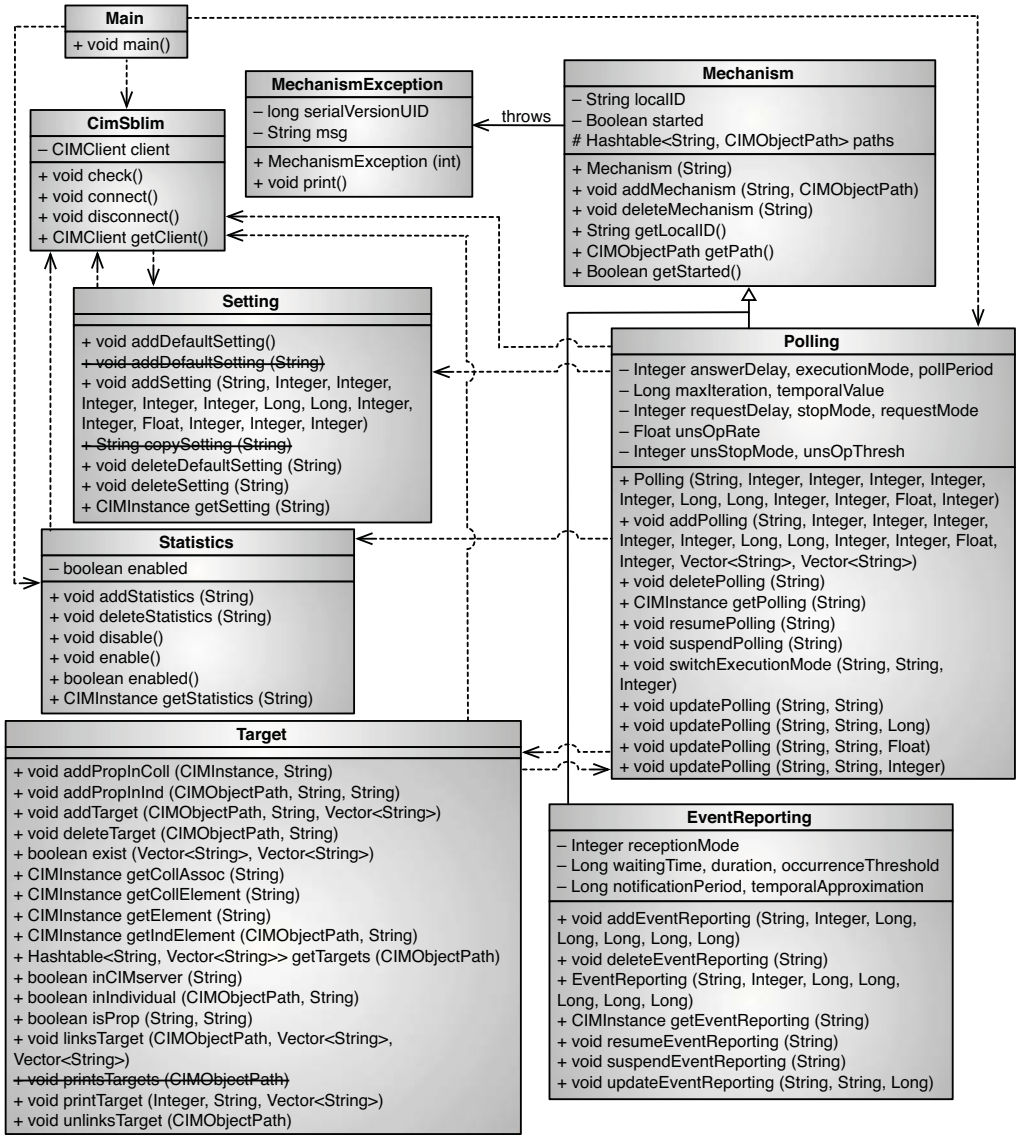

FIGURE 2.4: Diagramme de classes d'implémentation Java de la couche « adaptabilité »

Les valeurs manquantes peuvent alors être récupérées dans une configuration par défaut prédéfinie comme une instance de la classe `IRIT_PollingSettingData`.

- Si des valeurs sont spécifiées alors qu'elles sont inutiles (par exemple, un intervalle de temps `RequestDelay` alors que le mode d'exécution `ExecutionMode` est périodique `Periodic` et nécessite une période de *polling* `PollPeriod`), alors un avertissement doit être émis.
- Les cibles de l'opération de polling doivent être spécifiées : elles sont obligatoires. Une forte contrainte est ici positionnée sur les cibles du *polling* : si celles-ci n'existent pas au sein du serveur CIM et/ou ne sont pas interrogeables pour diverses raisons (en général, car la cible est inaccessible), la création du *polling* est simplement annulée.
- Toute création d'une opération de *polling* génère la création d'une instance de `IRIT_PollingSettingData` contenant la configuration courante de l'opération.

```
si les paramètres sont cohérents alors                                  Plan de contrôle
  si le mécanisme n'existe pas déjà alors
    si les cibles sont présentes et interrogeables et de même type alors
      instancier la classe IRIT_PollingOperation;
      instancier la classe IRIT_PollingSettingData;
      instancier l'association IRIT_PollingData;
      si il n'y a qu'une seule cible alors
        instancier l'association IRIT_IndividualPollingTarget;
      sinon
        instancier la classe IRIT_CollectionOfPollingTarget;
        instancier l'association CIM_CollectedMSEs;
        instancier l'association IRIT_CollectionPollingTarget;
      fin si;                                                   Plan opérationnel
      lancer le poller adaptable;
    fin si;
  fin si;
fin si;
```

FIGURE 2.5: Algorithme d'ajout de *polling*

2.3.1.1.2 *Event reporting*

De manière similaire, la figure 2.6 propose l'algorithme correspondant à l'ajout d'un mécanisme d'*event reporting*.

```
si les paramètres sont cohérents alors                                  Plan de contrôle
  si le mécanisme n'existe pas déjà alors
    instancier la classe IRIT_Listening;
    instancier la classe IRIT_ListeningSettingData;
    instancier l'association IRIT_ObservationListening;                 Plan opérationnel
    lancer le listener adaptable;
  fin si;
fin si;
```

FIGURE 2.6: Algorithme d'ajout d'*event reporting*

Si un paramètre de configuration est manquant ou erroné, l'*event reporting* est créé et son mode de réception `ReceptionMode` positionné à `none`. Le *listener* réceptionne les notifications mais ne gère pas l'analyse de leur fréquence d'arrivée.

2.3.1.2 Suppression

2.3.1.2.1 *Polling*

La figure 2.7 propose l'algorithme correspondant à la suppression d'un mécanisme de *polling*.

Si le mécanisme de *polling* n'existe pas, l'exécution de l'opérateur est annulée.

2.3.1.2.2 *Event reporting*

De manière similaire, la figure 2.8 propose l'algorithme correspondant à la suppression d'un mécanisme d'*event reporting*.

Si le mécanisme d'*event reporting* n'existe pas, l'exécution de l'opérateur est annulée.

```
si le mécanisme existe bien alors
  arrêter le poller adaptable;                                    Plan opérationnel
  supprimer l'association IRIT_PollingData;
  supprimer la classe IRIT_PollinSettingData:
  si il n'y a qu'une seule cible alors
    supprimer l'association IRIT_IndividualPollingTarget;
  sinon
    supprimer l'association CIM_CollectedMSEs;
    supprimer l'association IRIT_CollectionPollingTarget:
    supprimer la classe IRIT_CollectionOfPollingTarget;
  fin si;
  supprimer la classe IRIT_PollingOperation;
fin si;                                                           Plan de contrôle
```

FIGURE 2.7: Algorithme de suppression de *polling*

```
si le mécanisme existe bien alors
  arrêter le listener adaptable;                                  Plan opérationnel
  supprimer l'association IRIT_ObservationListening;
  supprimer la classe IRIT_ListeningSettingData;
  supprimer la classe IRIT_Listening;
fin si;                                                           Plan de contrôle
```

FIGURE 2.8: Algorithme de suppression d'*event reporting*

2.3.1.3 Modification

Plusieurs algorithmes sont nécessaires pour l'opérateur de modification. En effet, celui-ci permet les modifications dynamiques suivantes :

- un seul paramètre de configuration est modifié : c'est une modification ponctuelle de la configuration du mécanisme,
- tous les paramètres de configuration sont modifiés : c'est une modification globale de la configuration du mécanisme,
- dans le cas du *polling*, la liste des cibles interrogées peut être modifiée (par ajout ou suppression d'une cible) : c'est une modification de la portée du mécanisme.

2.3.1.3.1 Modification de la configuration

La figure 2.9 propose l'algorithme correspondant à la modification ponctuelle ou globale d'un mécanisme de *polling*.

```
si le mécanisme existe bien alors                                 Plan de contrôle
  si le(s) nouveau(x) paramètre(s) est(sont) cohérent(s) alors
    modifier la(les) valeur(s) du(des) paramètre(s) dans la classe IRIT_PollingOperation;
    enregistrer la nouvelle configuration dans la classe IRIT_PollingSettingData;
  fin si;
fin si;
```

FIGURE 2.9: Algorithme de modification de configuration de *polling*

La figure 2.10 propose l'algorithme correspondant à la modification ponctuelle ou globale d'un mécanisme d'*event reporting*.

```
si le mécanisme existe bien alors                                          Plan de contrôle
  si le(s) nouveau(x) paramètre(s) est(sont) cohérent(s) alors
    modifier la(les) valeur(s) du(des) paramètre(s) dans la classe IRIT_Listening;
    enregistrer la nouvelle configuration dans la classe IRIT_ListeningSettingData;
  fin si;
fin si;
```

FIGURE 2.10: Algorithme de modification de configuration d'*event reporting*

Dans le cas d'une modification ponctuelle, il suffit de préciser quel paramètre doit être modifié et quelle est sa nouvelle valeur.

Dans le cas d'une modification globale, il faut préciser le nom de la nouvelle configuration devant être appliquée au mécanisme. Il faut noter que les paramètres sélecteurs (non modifiables) conservent leur valeur (à l'exception du mode d'exécution `ExecutionMode` pour le *polling* et du mode de réception `ReceptionMode` pour l'*event reporting* qui sont des paramètres modifiables), tandis que les sous-paramètres sont, eux, tous modifiés.

2.3.1.3.2 Modification de la portée

La portée d'un mécanisme de *polling* peut également être modifiée en ajoutant une cible (figure 2.11) ou en supprimant une cible (figure 2.12).

```
récupérer les chemins de toutes les cibles de l'opération de polling;      Plan de contrôle
si l'opération de polling n'a qu'une seule cible alors
  instancier la classe IRIT_CollectionOfPollingTarget avec les deux cibles;
  instancier l'association CIM_CollectedMSEs;
  instancier l'association IRIT_CollectionPollingTarget;
  supprimer l'association IRIT_IndividualPollingTarget;
fin si;
si l'opération de polling a plus d'une cible alors
  récupérer la classe IRIT_CollectionOfPollingTarget contenant les liens vers les cibles;
  si la nouvelle cible est de même type que les cibles déjà enregistrées alors
    associer cette cible à la collection via CIM_CollectedMSEs;
    si des propriétés sont précisées alors
      ajouter les propriétés à PropertyList dans l'association IRIT_CollectionPollingTarget;
    fin si;
  fin si;
fin si;
```

FIGURE 2.11: Algorithme d'ajout de cible au *polling*

2.3.1.4 Suspension

L'opérateur de suspension ne concerne que le mécanisme de *polling*. Il n'est pas possible de suspendre un mécanisme d'*event reporting* dans la mesure où les actions correspondant à la suppression et à la suspension d'un tel mécanisme sont identiques : c'est un désabonnement.

La figure 2.13 propose l'algorithme correspondant à la suspension d'un mécanisme de *polling*.

Si le mécanisme de *polling* n'existe pas ou s'il est déjà suspendu, l'exécution de l'opérateur est annulée.

```
récupérer les chemins de toutes les cibles de l'opération de polling;          Plan de contrôle
si l'opération de polling n'a qu'une seule cible alors
  annuler l'opération;
fin si;
si l'opération de polling a deux cibles alors
  instancier l'association IRIT_IndividualPollingTarget avec la cible restante;
  supprimer l'association CIM_CollectedMSEs;
  supprimer l'association IRIT_CollectionPollingTarget;
  supprimer la classe IRIT_CollectionOfPollingTarget;
fin si;
si l'opération de polling a plus de deux cibles alors
  récupérer la classe IRIT_CollectionOfPollingTarget contenant les liens vers les cibles;
  supprimer la cible de la collection via CIM_CollectedMSEs;
fin si;
```

FIGURE 2.12: Algorithme de suppression de cible du *polling*

```
si le mécanisme existe bien alors                                              Plan de contrôle
  si le mécanisme est bien actif alors
    modifier l'état de la classe IRIT_PollingOperation à pending;
  fin si;
  enregistrer le contexte opérationnel courant;                                Plan opérationnel
  arrêter le poller adaptable;
fin si;
```

FIGURE 2.13: Algorithme suspension de *polling*

2.3.1.5 Reprise

Comme dans le cas de la suspension, l'opérateur de reprise ne concerne que le mécanisme de *polling*. Il n'est pas possible de reprendre un mécanisme d'*event reporting* dans la mesure où les actions correspondant à l'ajout et à la reprise d'un tel mécanisme sont identiques : c'est un abonnement.

La figure 2.14 propose l'algorithme correspondant à la reprise d'un mécanisme de *polling*.

```
si le mécanisme existe bien alors                                              Plan de contrôle
  si le mécanisme est bien suspendu alors
    modifier l'état de la classe IRIT_PollingOperation à running;
  fin si;
  charger le contexte opérationnel courant;                                    Plan opérationnel
  lancer le poller adaptable;
fin si;
```

FIGURE 2.14: Algorithme de reprise de *polling*

Si le mécanisme de *polling* n'existe pas ou s'il est déjà actif, l'exécution de l'opérateur est annulée.

2.3.2 Exemples de fonctionnement

A l'issue de cette implémentation, le prototype obtenu présente, en plus du *poller* et du *listener* adaptables, une interface composée d'opérateurs permettant de lancer des opérations d'adaptation de la configuration et/ou de la portée des mécanismes de surveillance.

2.3.2.1 Modifications portant sur le *polling*

Cette section propose quelques exemples d'adaptation du comportement du *poller* adaptable. Tous les exemples de fonctionnement ne sont pas proposés ici car ne sont pas visualisables sur un document papier – modification de la période de *polling* de 2s à 10s, notamment – mais ont été testés et sont fonctionnels.

Notez qu'une ou plusieurs opérations peuvent être improductives au moment de la reconfiguration. Ceci est dû au fait que le serveur CIM, Open Pegasus, ne peut pas gérer deux accès concurrents.

2.3.2.1.1 Modification dynamique du nombre d'itérations

Cette adaptation dynamique vise à modifier une valeur d'un paramètre d'une instance de *polling*. La modification souhaitée est visible sur la figure 2.15.

IRIT_PollingOperation	
CreationClassName	IRIT_PollingOperation
Name	CfgSeq1
SystemCreationClassName	AFDX_EndSystem
SystemName	127.0.0.1
ExecutionMode	2
PollPeriod	
RequestDelay	5000
AnswerDelay	2000
RequestMode	1
StopMode	2
MaxIteration	20
TemporalValue	
UnsuccessfulStopMode	1
UnsuccessfulOperationThreshold	
UnsuccessfulOperationRate	
Started	true

$\mathcal{U}$

IRIT_PollingOperation	
CreationClassName	IRIT_PollingOperation
Name	CfgSeq1
SystemCreationClassName	AFDX_EndSystem
SystemName	127.0.0.1
ExecutionMode	2
PollPeriod	
RequestDelay	5000
AnswerDelay	2000
RequestMode	1
StopMode	2
MaxIteration	30
TemporalValue	
UnsuccessfulStopMode	1
UnsuccessfulOperationThreshold	
UnsuccessfulOperationRate	
Started	true

FIGURE 2.15: Instance de *polling* avant/après

Lors de l'exécution du *poller* adaptable appliquant la configuration `CfgSeq1`, une adaptation est réalisée au moyen de l'opérateur $\mathscr{U}$. La modification du nombre d'itérations (de 20 à 30) est repérée par le *poller* adaptable, qui modifie le comportement du *polling* (voir la figure 2.16).

```
|DONE| Connected to CIM server root/AFDX@192.168.56.101:5988
|DONE| Initial configuration uploaded: 2000:2:0:5000:2:20:0:1:0:0.0:1
|INFO| Polling execution mode = no overlapping
|INFO| Adaptation request listening
REQUEST n° 1
|INFO| Total Duration needed for Answer: 208
+ Target path: root/AFDX:AFDX_EndSystem.CreationClassName="AFDX_EndSystem",Name="127.0.0.1"
EnabledState: 1
RequestedState: 12
|INFO| Polling behaviour modification requested
 ---> Current Configuration: 2000:2:0:5000:2:20:0:1:0:0.0:1
 ---> Wished Configuration: 2000:2:0:5000:2:30:0:1:0:0.0:1
 ---> Current Configuration: 2000:2:0:5000:2:30:0:1:0:0.0:1
REQUEST n° 2
|INFO| Total Duration needed for Answer: 40
+ Target path: root/AFDX:AFDX_EndSystem.CreationClassName="AFDX_EndSystem333",Name="127.0.0.1"
EnabledState: 3
RequestedState: 12
[...sortie écran tronquée pour plus de visibilité...]
REQUEST n° 30
|INFO| Total Duration needed for Answer: 30
+ Target path: root/AFDX:AFDX_EndSystem.CreationClassName="AFDX_EndSystem333",Name="127.0.0.1"
EnabledState: 3
RequestedState: 12
|DONE| Max iteration number reached: polling termination
```

Polling.updatePolling("CfgSeq1", "MaxIteration", 30l);

FIGURE 2.16: Modification dynamique du nombre d'itérations du *polling*

2.3.2.1.2 Suppression d'une cible

Cette adaptation dynamique vise à réduire la portée du *polling*. Lors de l'exécution du *poller* adaptable, la cible `AFDX_EndSystem333` est supprimée. Cette modification est repérée, et le comportement du *poller* est aussitôt adapté (voir la figure 2.17).

```
|DONE| Connected to CIM server root/AFDX@192.168.56.101:5988
|DONE| Initial configuration uploaded: 2000:2:0:5000:2:30:0:1:0:0.0:1
|INFO| Polling execution mode = no overlapping
|INFO| Adaptation request listening
REQUEST n° 1
|INFO| Total Duration needed for Answer: 105
+ Target path: root/AFDX:AFDX_EndSystem.CreationClassName="AFDX_EndSystem",Name="127.0.0.1"
EnabledState: 1
RequestedState: Target.deleteTarget(Polling.getPolling("CfgSeq1").getObjectPath(), "root/
REQUEST n° 2   AFDX:AFDX_EndSystem.CreationClassName=\"AFDX_EndSystem333\",Name=\"127.0.0.1\"");
|INFO| Total Duration needed for Answer: 58
+ Target path: root/AFDX:AFDX_EndSystem.CreationClassName="AFDX_EndSystem333",Name="127.0.0.1"
EnabledState: 3
RequestedState: 12
|INFO| Target has been removed
REQUEST n° 3
|INFO| Total Duration needed for Answer: 36
+ Target path: root/AFDX:AFDX_EndSystem.CreationClassName="AFDX_EndSystem",Name="127.0.0.1"
EnabledState: 1
RequestedState: 12
REQUEST n° 4
|INFO| Total Duration needed for Answer: 32
+ Target path: root/AFDX:AFDX_EndSystem.CreationClassName="AFDX_EndSystem",Name="127.0.0.1"
EnabledState: 1
RequestedState: 12
[...sortie écran tronquée pour plus de visibilité...]
|DONE| Max iteration number reached: polling termination
```

FIGURE 2.17: Suppression dynamique d'une cible du *polling*

2.3.2.1.3 Ajout d'une cible

Cette adaptation vise à augmenter la portée du *polling*. Lors de l'exécution du *poller* adaptable, la cible `AFDX_EndSystem333` est rajoutée au *polling*. Le comportement du *poller* en est aussitôt modifié (voir la figure 2.18).

```
|DONE| Connected to CIM server root/AFDX@192.168.56.101:5988
|DONE| Initial configuration uploaded: 2000:2:0:5000:2:30:0:1:0:0.0:1
|INFO| Polling execution mode = no overlapping
|INFO| Adaptation request listening
REQUEST n° 1
|INFO| Total Duration needed for Answer: 47
+ Target path: root/AFDX:AFDX_EndSystem.CreationClassName="AFDX_EndSystem",Name="127.0.0.1"
Enabled Vector<String> props=new Vector<String>();
Request props.add("EnabledState"); props.add("RequestedState");
REQUEST Target.addTarget(Polling.getPolling("CfgSeq1").getObjectPath(), "root/
|INFO|  AFDX:AFDX_EndSystem.CreationClassName=\"AFDX_EndSystem333\",Name=\"127.0.0.1\"", props);
+ Target path: root/AFDX:AFDX_EndSystem.CreationClassName="AFDX_EndSystem",Name="127.0.0.1"
EnabledState: 1
RequestedState: 12
|INFO| Target has been added
REQUEST n° 3
|INFO| Total Duration needed for Answer: 39
+ Target path: root/AFDX:AFDX_EndSystem.CreationClassName="AFDX_EndSystem",Name="127.0.0.1"
EnabledState: 1
RequestedState: 12
REQUEST n° 4
|INFO| Total Duration needed for Answer: 59
+ Target path: root/AFDX:AFDX_EndSystem.CreationClassName="AFDX_EndSystem333",Name="127.0.0.1"
EnabledState: 3
RequestedState: 12
[...sortie écran tronquée pour plus de visibilité...]
|DONE| Max iteration number reached: polling termination
```

FIGURE 2.18: Ajout dynamique d'une cible du *polling*

2.3.2.1.4 Modification du temps d'attente maximum de la réponse

Cette adaptation consiste en trois modifications successives toutes réalisées au moyen de l'opérateur $\mathscr{U}$. Les modifications du temps d'attente maximum de la réponse (de 2000ms à 50ms, puis à 20ms, et enfin à 40ms) sont repérées par le *poller* adaptable qui modifie le comportement du *polling* (voir la figure 2.19).

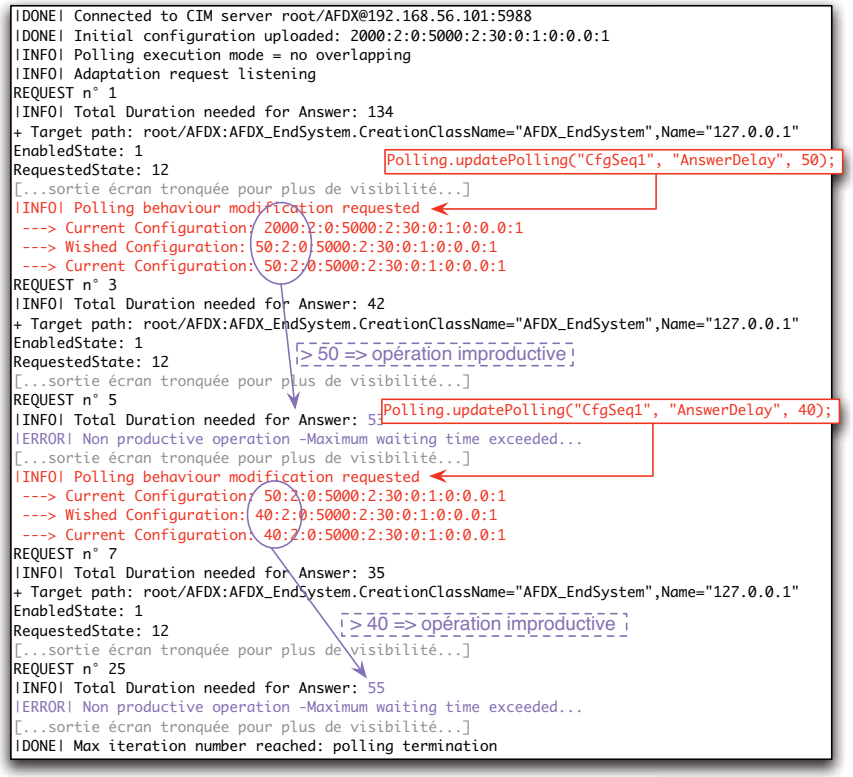

FIGURE 2.19: Modification dynamique du temps d'attente maximum de la réponse

Cette adaptation permet en outre d'observer des opérations improductives dues au fait que les réponses sont arrivées trop tardivement.

2.3.2.1.5 Modification du nombre d'opérations successives improductives

Les opérations successives improductives peuvent être utiles dans le cas d'un mode d'arrêt autonome par seuil d'opérations. Ainsi, dans la figure 2.20, une modification de ce seuil de 4 opérations à 2 est opérée, puis repérée par le *poller* adaptable qui modifie le comportement du *polling*.

2.3.2.1.6 Suspension et reprise du *polling*

Ces adaptations permettent de montrer la possibilité de suspendre et reprendre un *poller* adaptable, et de mettre en évidence la sauvegarde de son contexte opérationnel courant.

Lors de l'exécution du *poller*, une suspension est réalisée au moyen de l'opérateur $\mathscr{S}$. Le contexte opérationnel du *polling* est sauvegardé et l'exécution est suspendue, comme cela est visible sur le cadre du haut de la figure 2.21.

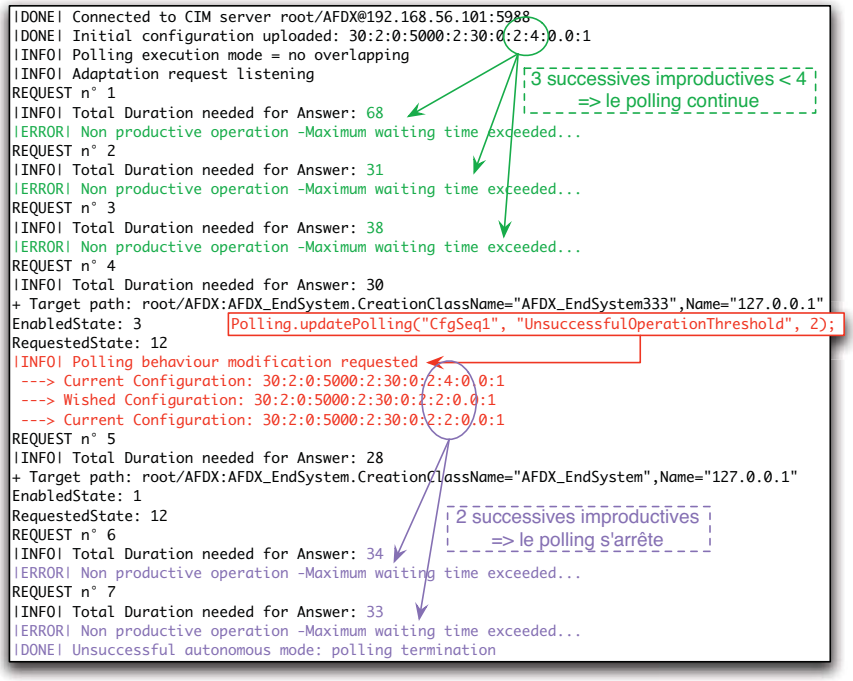

FIGURE 2.20: Modification dynamique du seuil d'opérations successives improductives

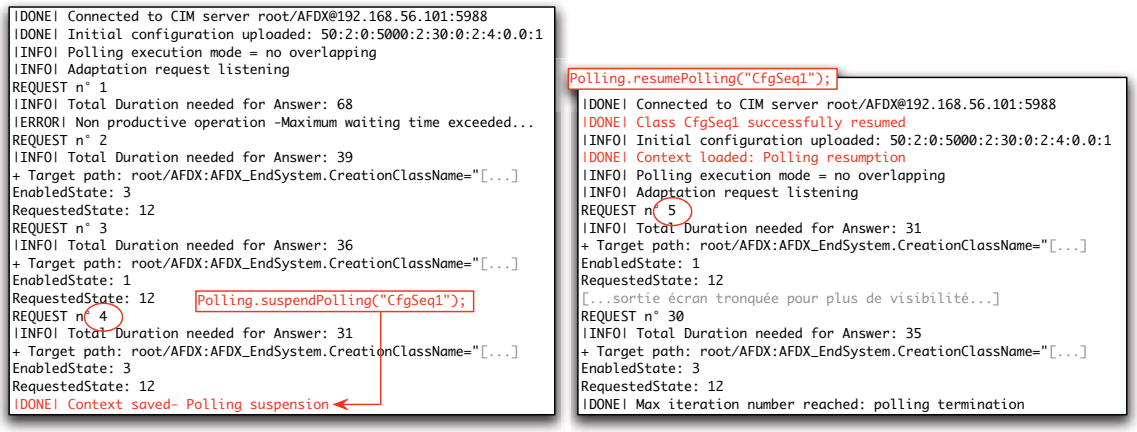

FIGURE 2.21: Suspension et reprise du *polling*

La reprise du *polling*, réalisée au moyen de l'opérateur $\mathscr{R}$, est visualisée sur le cadre de droite de la figure 2.21 : le contexte du *polling* est restauré et l'exécution reprend.

2.3.2.2 Modifications portant sur l'*event reporting*

Cette section propose des exemples d'adaptation du comportement du *listener* adaptable : une adaptation est opérée ici sur chacun des modes d'observation.

2.3.2.2.1 Détection de silence

Lors de l'exécution du *listener*, une modification est réalisée sur l'*event reporting* au moyen de l'opérateur $\mathscr{U}$. La modification du temps d'attente maximum de la notification (de 30000ms à 120ms) est repérée par le *listener* adaptable. Aucune notification n'arrive avec plus de 30s de retard, mais certaines peuvent mettre plus de 120ms à parvenir. La modification doit donc faire apparaître des silences (figure 2.22).

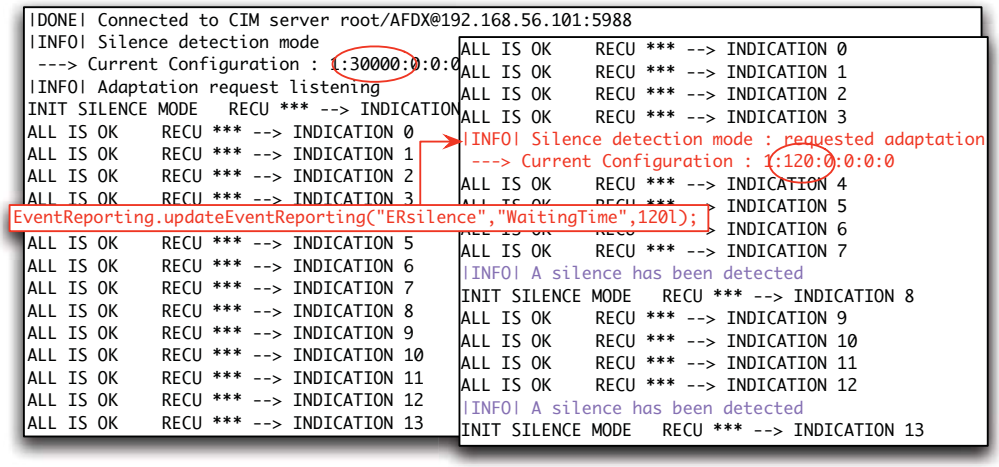

FIGURE 2.22: Détection de silence et modification du temps d'attente maximum

2.3.2.2.2 Détection de salves

Lors de l'exécution du *listener*, une modification est réalisée sur l'*event reporting* au moyen de l'opérateur $\mathscr{U}$. La modification du seuil de notifications (de 2 à 25) est repérée par le *listener* adaptable. Cette modification réduit la quantité de salves détectées car il ne faut plus 2 notifications sur la durée précisée pour qu'une salve soit détectée, mais 25 (figure 2.23).

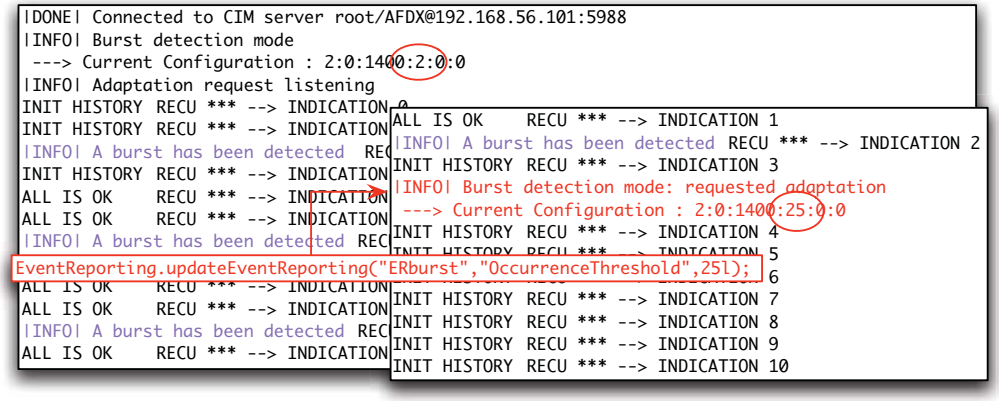

FIGURE 2.23: Détection de salves et modification du seuil de notifications

2.3.2.2.3 Détection de perte d'isochronisme

Lors de l'exécution du *listener*, une modification est réalisée sur l'*event reporting* au moyen de l'opérateur $\mathscr{U}$. La modification de l'approximation temporelle pour la réception de la notification (de 45 à 200) est repérée par le *listener* adaptable. Cette modification réduit le nombre de pertes d'isochronisme car agrandit le décalage temporel autorisé (figure 2.24).

```
|DONE| Connected to CIM server root/AFDX@192.168.56.101:5988
|INFO| Periodicity loss detection mode
 ---> Current Configuration : 3:0:0:0:2000:45
|INFO| Adaptation request listening
INIT PERIODIC HISTORY      RECU *** --> INDICATION 0
ALL IS OK     RECU *** --> INDICATION 1
ALL IS OK     RECU *** --> INDICATION 2
ISOCHRONISM LOSS DETECTED (ADVANCE)     RECU *** --> INDICATION 3
ALL IS OK     RECU *** --> INDICATION 4
ALL IS OK     RECU *** --> INDICATION 5
ISOCHRONISM LOSS DETECTED (ADVANCE)     RECU *** --> INDICATION 6
ALL IS OK     RECU *** --> INDICATION 7
ISOCHRONISM LOSS DETECTED (DELAY)RECU *** --> INDICATION 8
ISOCHRONISM LOS EventReporting.updateEventReporting("ERheartbeat","TemporalApproximation",200l);
ISOCHRONISM LOSS DETECTED (ADVANCE)     RECU *** --> INDICATION 10
INIT PERIODIC HISTORY      RECU *** --> INDICATION 0
|INFO| Periodicity loss detection mode: requested adaptation
 ---> Current Configuration : 3:0:0:0:2000:200
ALL IS OK     RECU *** --> INDICATION 1
ALL IS OK     RECU *** --> INDICATION 2
ALL IS OK     RECU *** --> INDICATION 3
ALL IS OK     RECU *** --> INDICATION 4
ALL IS OK     RECU *** --> INDICATION 5
ISOCHRONISM LOSS DETECTED (ADVANCE)     RECU *** --> INDICATION 6
ALL IS OK     RECU *** --> INDICATION 7
ISOCHRONISM LOSS DETECTED (DELAY)RECU *** --> INDICATION 8
ALL IS OK     RECU *** --> INDICATION 9
ISOCHRONISM LOSS DETECTED (ADVANCE)     RECU *** --> INDICATION 10
```

FIGURE 2.24: Détection de perte d'isochronisme et modification du décalage temporel

2.3.3 Mesures du surcoût temporel de l'adaptation

Le prototype permettant de gérer l'adaptation de la surveillance ayant été développé au sein d'un environnement CIM/WBEM, des tests ont été réalisés pour d'une part mesurer le surcoût dû à la gestion de l'adaptation, et d'autre part pour mettre en évidence les implications concernant la manière d'utiliser les opérateurs d'adaptation développés.

Les résultats présentés dans les tableaux suivants ont été obtenus en calculant la moyenne de cinquante mesures réalisées dans des conditions favorables : le client CIM tourne sur la machine hôte, tandis que le serveur CIM tourne sur la machine virtuelle située sur la machine hôte. Seuls les objets nécessaires au calcul sont enregistrés dans le *CIM repository* (minimisation du temps de récupération des objets CIM).

Les tableaux ci-après présentent le temps nécessaire à l'exécution des opérateurs d'adaptabilité du démonstrateur dans le cas du *polling*. Ce temps d'exécution a été décomposé en deux :

- le temps « modèle » correspond au temps nécessaire à la modification des instances CIM (plan de contrôle),
- le temps « opérationnel » correspond au temps nécessaire pour l'exécution opérationnelle de l'opérateur considéré (plan opérationnel).

Les tableaux mettent en évidence le surcoût provoqué par le temps « modèle ». Chaque opérateur mesuré fait l'objet d'une première analyse indépendamment des autres opérateurs. Une discussion est enfin apportée sur l'ensemble des mesures et résultats en conclusion de cette partie.

2.3.3.1 Opérateur d'adjonction

2.3.3.1.1 Mesures brutes

Les mesures correspondent à la durée comprise entre l'invocation de l'opérateur d'adjonction et le moment où le *polling* est opérationnel (c'est-à-dire le lancement de la première requête). Cet intervalle de temps inclut l'enregistrement de la configuration du *polling* sur le serveur CIM. La majeure partie du temps d'exécution est consommée par l'instanciation du modèle CIM.

Une opération de *polling* peut être ajoutée avec des cibles individuelles ou des cibles collectives. Il est donc intéressant de montrer l'augmentation du temps d'exécution en fonction de ces deux cas particuliers.

Le tableau 2.1 présente les temps d'exécution de l'ajout de $n = 1..10$ *polling* dont chacun est appliqué sur une cible dite « individuelle ».

	Temps d'exécution		Nombre d'éléments CIM nécessaires
	modèle	opérationnel	
1	439 ms (87 %)	61 ms (13 %)	5
2	938 ms (89 %)	118 ms (11 %)	10
3	1480 ms (89 %)	184 ms (11 %)	15
4	2079 ms (89 %)	262 ms (11 %)	20
5	2709 ms (88 %)	346 ms (12 %)	25
6	3317 ms (88 %)	427 ms (12 %)	30
7	4008 ms (88 %)	518 ms (12 %)	35
8	4686 ms (88 %)	616 ms (12 %)	40
9	5438 ms (88 %)	725 ms (12 %)	45
10	6203 ms (88 %)	838 ms (12 %)	50

TABLE 2.1: Ajout de n *polling* avec cibles individuelles

Le tableau 2.2 présente les temps d'exécution de l'ajout d'un seul *polling* appliqué sur $n = 2..10$ cibles dites « collectives ».

Ces résultats peuvent être visualisés sur le graphe de la figure 2.25.

	Temps d'exécution		Nombre d'éléments CIM nécessaires
	modèle	opérationnel	
2	641 ms (89 %)	82 ms (11 %)	9
3	717 ms (93 %)	48 ms (7 %)	11
4	897 ms (95 %)	51 ms (5 %)	13
5	1168 ms (94 %)	74 ms (6 %)	15
6	1327 ms (97 %)	48 ms (3 %)	17
7	1625 ms (97 %)	56 ms (3 %)	19
8	1850 ms (97 %)	49 ms (3 %)	21
9	2166 ms (97 %)	64 ms (3 %)	23
10	2423 ms (98 %)	58 ms (12 %)	25

TABLE 2.2: Ajout d'un *polling* avec *n* cibles collectives

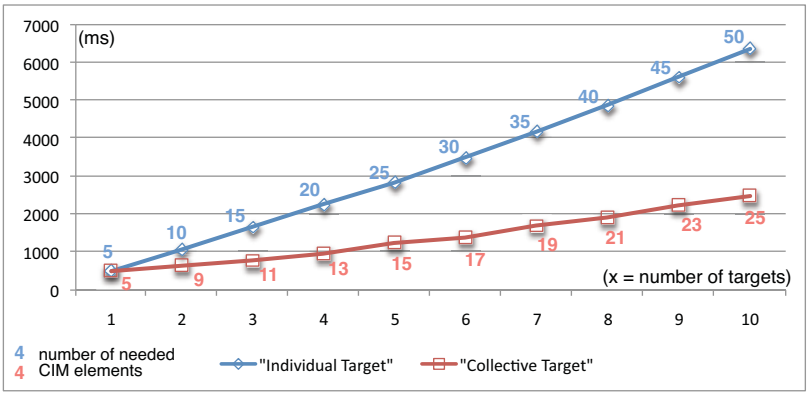

FIGURE 2.25: Durée d'exécution pour l'opérateur d'adjonction

2.3.3.1.2 Analyse et impacts

Comme le montrent les résultats, l'augmentation du nombre de cibles fait augmenter le temps d'exécution de l'ajout du *polling*. Mais une différence apparaît selon la manière d'utiliser l'opérateur :

- une opération de *polling* appliquée sur dix cibles est créée et rendue opérationnelle en 2481 ms (un *polling* lancé, vingt-cinq éléments CIM créés et/ou lus dans le *CIM repository*),
- tandis que dix opérations de *polling* appliquées chacune sur une cible sont créées et rendues opérationnelles en 7041 ms (dix *polling* lancés, cinquante éléments CIM créés et/ou lus dans le *CIM repository*).

Le ratio est de 2,8. Par conséquent, l'ajout d'opérations de *polling* avec cibles collectives est globalement plus intéressant que l'ajout d'opérations de *polling* avec cibles individuelles, car il peut être réalisé plus rapidement.

2.3.3.2 Opérateur de suppression

2.3.3.2.1 Mesures brutes

Les mesures correspondent à la durée comprise entre l'invocation de l'opérateur de suppression et le moment où l'information de contrôle du *polling* est totalement supprimée du serveur CIM. Le temps d'exécution correspond surtout à la modification du modèle CIM, car le plan opérationnel n'a

qu'à arrêter le *poller* adaptable en cours d'exécution.

Comme une opération de *polling* peut être ajoutée avec des cibles individuelles ou des cibles collectives, il convient de mesurer le temps d'exécution de la suppression de *polling* pour chacun de ces deux cas particuliers.

Le tableau 2.3 présente les temps d'exécution de la suppression de $n = 1..10$ *polling* dont chacun est appliqué sur une cible dite « individuelle ».

	Temps d'exécution		Nombre d'éléments CIM nécessaires
	modèle	opérationnel	
1	216 ms (100 %)	négligeable	4
2	468 ms (100 %)	négligeable	8
3	767 ms (100 %)	négligeable	12
4	1089 ms (100 %)	négligeable	16
5	1437 ms (100 %)	négligeable	20
6	1780 ms (100 %)	négligeable	24
7	2141 ms (100 %)	négligeable	28
8	2513 ms (100 %)	négligeable	32
9	2921 ms (100 %)	négligeable	36
10	3344 ms (100 %)	négligeable	40

TABLE 2.3: Suppression de n *polling* avec cibles individuelles

Le tableau 2.4 présente les temps d'exécution de la suppression d'un seul *polling* appliqué sur $n = 2..10$ cibles dites « collectives ».

	Temps d'exécution		Nombre d'éléments CIM nécessaires
	modèle	opérationnel	
2	1042 ms (100 %)	négligeable	7
3	893 ms (100 %)	négligeable	8
4	882 ms (100 %)	négligeable	9
5	952 ms (100 %)	négligeable	10
6	971 ms (100 %)	négligeable	11
7	957 ms (100 %)	négligeable	12
8	898 ms (100 %)	négligeable	13
9	1024 ms (100 %)	négligeable	14
10	1036 ms (100 %)	négligeable	15

TABLE 2.4: Suppression d'un *polling* avec n cibles collectives

Il faut noter ici que, pour une cible individuelle comme pour un ensemble de cibles, le temps opérationnel est négligeable : le *poller* adaptable s'arrête en moins d'une milliseconde.

Ces résultats peuvent être visualisés sur le graphe de la figure 2.26.

2.3.3.2.2 Analyse et impacts

Comme le montrent les résultats, l'augmentation du nombre de cibles fait augmenter le temps d'exécution de la suppression de *polling* avec cibles individuelles. Par contre, quand un *polling* est appliqué sur plusieurs cibles, le temps d'exécution de la suppression est presque constant :

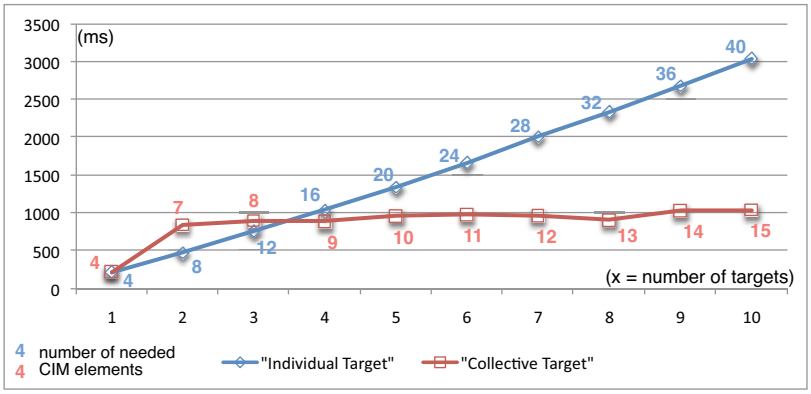

FIGURE 2.26: Durée d'exécution pour l'opérateur de suppression

- une opération de *polling* appliquée sur dix cibles est supprimée en 1036 ms (un *polling* arrêté, quinze éléments CIM supprimés et/ou lus dans le *CIM repository*).
- tandis que dix opérations de *polling* appliquées chacune sur une cible sont supprimées en 3344 ms (dix *polling* arrêtés, quarante éléments CIM supprimés et/ou lus dans le *CIM repository*).

Le ratio est de 3,2. Par conséquent, l'utilisation de cibles collectives s'avère plus intéressante lorsque plus de trois cibles sont impliquées ; autrement, l'utilisation de cibles individuelles est plus rapide.

Toutefois, ceci n'est pas sans contrepartie : en effet, le *polling* des cibles collectives est réalisé cycliquement, de sorte qu'une seule cible est interrogée à chaque requête. Ainsi, là où une cible individuelle de *polling* est interrogée *maxIteration* fois, dans le cas de *polling* avec cibles collectives, chaque cible est interrogée $\frac{maxIteration}{nbCiblesCollectives}$ fois. Pour pouvoir interroger plus souvent les cibles collectives d'un *polling*, il faut donc jouer sur le nombre d'itérations de ce *polling*.

2.3.3.3 Opérateurs de suspension et reprise

2.3.3.3.1 Mesures brutes

Dans le cas de la suspension d'un *polling*, les mesures correspondent à la durée comprise entre l'invocation de l'opérateur de suspension et le moment où le *polling* est opérationnellement suspendu. Cet intervalle de temps inclut l'enregistrement du contexte opérationnel courant du *polling* à suspendre (nombre de requêtes déjà réalisées, nombre de requêtes improductives...).

Dans le cas de la reprise d'un *polling*, les mesures correspondent à la durée comprise entre l'invocation de l'opérateur de reprise et le moment où le *polling* est opérationnel. Cet intervalle de temps inclut la récupération du contexte opérationnel du *polling* à reprendre, ainsi que la mise à jour du modèle CIM.

Le nombre de cibles du *polling* et sa configuration globale n'affectent pas les temps d'exécution de ces deux opérateurs.

Les temps d'exécution de ces deux opérateurs sont présentés sur le tableau 2.5.

La reprise du *polling* est plus coûteuse en temps car il est nécessaire de récupérer le contexte opérationnel du *polling* (en local), mais aussi la configuration complète et la liste des cibles du *polling* (sur le serveur CIM).

	Temps d'exécution		Nombre d'éléments CIM nécessaires
	modèle	opérationnel	
Suspension	100 ms (92 %)	9 ms (8 %)	1
Reprise	101 ms (56 %)	79 ms (44 %)	1

TABLE 2.5: Suspension et reprise d'une opération de *polling*

2.3.3.3.2 Analyse et impacts

Soit le cas d'un *polling* possédant deux cibles :

- le temps d'exécution total de l'ajout est de 723 ms ;
- le temps d'exécution total de la suppression est de 1042 ms ;
- le temps d'exécution total de la suspension est de 109 ms ;
- le temps d'exécution total de la reprise est de 181 ms.

Un ajout et une suppression combinés nécessitent donc un temps d'exécution de 1765 ms ; tandis qu'une suspension et une reprise combinées nécessitent un temps d'exécution de 290 ms.

Certaines cibles étant plus importantes que d'autres et nécessitant d'être interrogées plus d'une fois, cette possibilité de suspendre et reprendre le *polling* apparaît comme particulièrement intéressante, vu que la création et la suppression sont plus consommatrices en terme de durée d'exécution.

2.3.3.4 Opérateurs de modification

2.3.3.4.1 Modification de la configuration du *polling*

Les mesures correspondent à la durée comprise entre l'invocation de l'opérateur de modification et le moment où le *polling* prend en compte opérationnellement la nouvelle valeur de paramètre modifiée. Trois types de mises à jour sont mesurées pour montrer qu'une adaptation dynamique peut être réalisée rapidement :

- la modification de la valeur d'un paramètre sélecteur, autrement dit la valeur du paramètre `ExecutionMode` (`Periodic` ou `NoOverlapping`),
- la modification de n'importe quel autre paramètre de configuration modifiable (les sous-paramètres) : cette modification implique l'exécution de divers tests pour prendre en compte les contraintes de consistance et d'intégrité (par exemple, le sous-paramètre `PollPeriod` peut être modifié uniquement si son paramètre sélecteur `ExecutionMode` est positionné à `Periodic`),
- la modification globale de la configuration du *polling*, qui modifie en une seule opération l'ensemble des paramètres modifiables du *polling*.

Les temps d'exécution de ces trois types de modification sont présentés sur le tableau 2.6.

2.3.3.4.2 Modification de la portée du *polling*

Dans le cas d'un ajout d'une cible au *polling*, les mesures correspondent à la durée comprise entre l'invocation de l'opérateur de modification et le moment où le *polling* prend en compte opérationnellement la cible supplémentaire. Une différence doit être faite selon le nombre de cibles initial :

	Temps d'exécution		Nombre d'éléments CIM nécessaires
	modèle	opérationnel	
Paramètre sélecteur	114 ms (84 %)	22 ms (16 %)	3
Sous-paramètre	131 ms (88 %)	18 ms (12 %)	3
Modification globale	115 ms (89 %)	13 ms (11 %)	3

TABLE 2.6: Modification de la configuration d'une opération de *polling*

- si le *polling* est initialement appliqué sur une seule cible, le modèle CIM est plus fortement impacté : destruction de l'organisation « cible individuelle » au profit d'une organisation « cible collective » composée de la cible initiale et de la nouvelle cible,
- si le *polling* est initialement appliqué sur deux cibles ou plus, le modèle CIM n'est que peu impacté : il faut juste lier la nouvelle cible à l'ensemble des cibles collectives déjà existant.

Les temps d'exécution de l'ajout de cibles au *polling* sont présentés sur le tableau 2.7.

Configuration initiale	Configuration finale	Temps d'exécution
1 cible individuelle	2 cibles collectives	845 ms
2 cibles collectives	3 cibles collectives	374 ms
3 cibles collectives	4 cibles collectives	378 ms
4 cibles collectives	5 cibles collectives	396 ms

TABLE 2.7: Ajout de cibles à une opération de *polling*

La même distinction doit être faite dans le cas d'une suppression d'une cible du *polling* :

- si le *polling* est initialement appliqué sur une cible, la suppression est annulée,
- si le *polling* est initialement appliqué sur deux cibles, le modèle CIM est plus fortement impacté : destruction de l'organisation « cibles collectives » au profit d'une organisation « cible individuelle » composée de la seule cible restante,
- si le *polling* est initialement appliqué sur trois cibles ou plus, le modèle CIM n'est que peu impacté : il faut juste délier la cible à supprimer de l'ensemble des cibles collectives déjà existant.

Les temps d'exécution de la suppression de cibles du *polling* sont présentés sur le tableau 2.8.

Configuration initiale	Configuration finale	Temps d'exécution
2 cibles collectives	1 cible individuelle	1250 ms
3 cibles collectives	2 cibles collectives	304 ms
4 cibles collectives	3 cibles collectives	317 ms
5 cibles collectives	4 cibles collectives	328 ms

TABLE 2.8: Suppression de cibles d'une opération de *polling*

Ces résultats peuvent être visualisés sur le graphe de la figure 2.27.

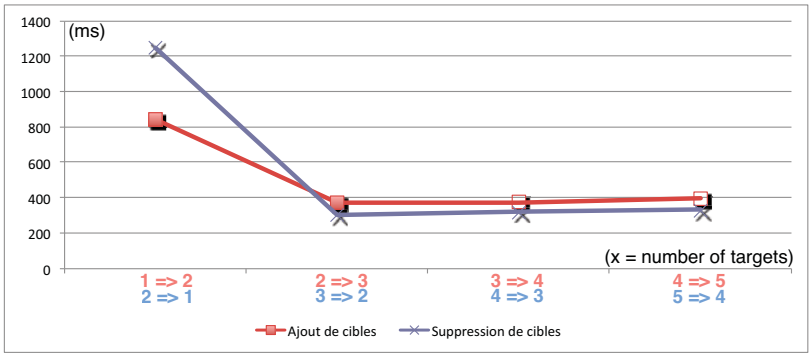

FIGURE 2.27: Durée d'exécution pour la modification de la portée

2.3.3.5 Discussion sur les résultats expérimentaux

L'interface d'adaptabilité a été implémentée et testée pour montrer aussi bien la faisabilité de l'approche que le surcoût temporel lié à la gestion de l'adaptation. Positionnée à un haut niveau d'abstraction et grâce aux facilités apportées par le modèle CIM pré-existant, l'implémentation n'a demandé que peu d'efforts de modélisation. Deux impacts peuvent être mis en évidence et offrent un intérêt à l'utilisation du prototype :

- la réduction du coût en temps et en ressources grâce à l'utilisation conjointe des opérateurs de suspension et de reprise en lieu et place des opérateurs d'ajout et de suppression,
- outre la possibilité de modifier la configuration des mécanismes de surveillance, est ici aussi proposée celle d'élargir ou de réduire la portée d'un mécanisme de *polling* par l'ajout ou la suppression de cibles.

Prenons désormais un peu de recul par rapport à ces premiers résultats ? Dans quels domaines applicatifs sont-ils acceptables ?

Si l'utilisation de ce prototype doit se faire sur une durée limitée, il est impératif de comparer ces mesures au temps imposé pour déterminer le pourcentage de temps qui serait alloué à l'adaptation de la surveillance et voir si celui-ci est acceptable.

Par contre, si l'adaptation de la surveillance n'est pas soumis à des contraintes de temps, alors l'approche est acceptable et intéressante car elle apporte des moyens efficaces pour faciliter l'adaptation de la surveillance : celle-ci est en effet réalisable en ligne, en temps réel et automatiquement.

2.4 Conclusion

La surveillance adaptative est définie comme la capacité qu'a une fonction de surveillance de décider et de mettre en application, sans interruption de service, l'ajustement de son comportement pour maintenir son efficacité en respectant les variations des contraintes fonctionnelles et opérationnelles, ainsi que pour améliorer son efficacité dans un objectif d'auto-optimisation.

Le comportement d'un mécanisme de surveillance est conditionné par ses paramètres de configuration spécifiés dans le module « configurabilité » du prototype, présenté en chapitre 1.

Le chapitre 2 a porté, lui, sur l'« adaptabilité » des mécanismes de surveillance qui permet de mettre en application l'ajustement du comportement des mécanismes de surveillance dynamiquement

et sans interruption de service, au travers d'une interface présentant un ensemble d'opérateurs d'adaptation. Ainsi, cette partie a porté sur la définition, la spécification et l'implémentation de ces opérateurs d'adaptation, constitutifs de la capacité d'« adaptabilité » visible en rouge sur la figure 2.28.

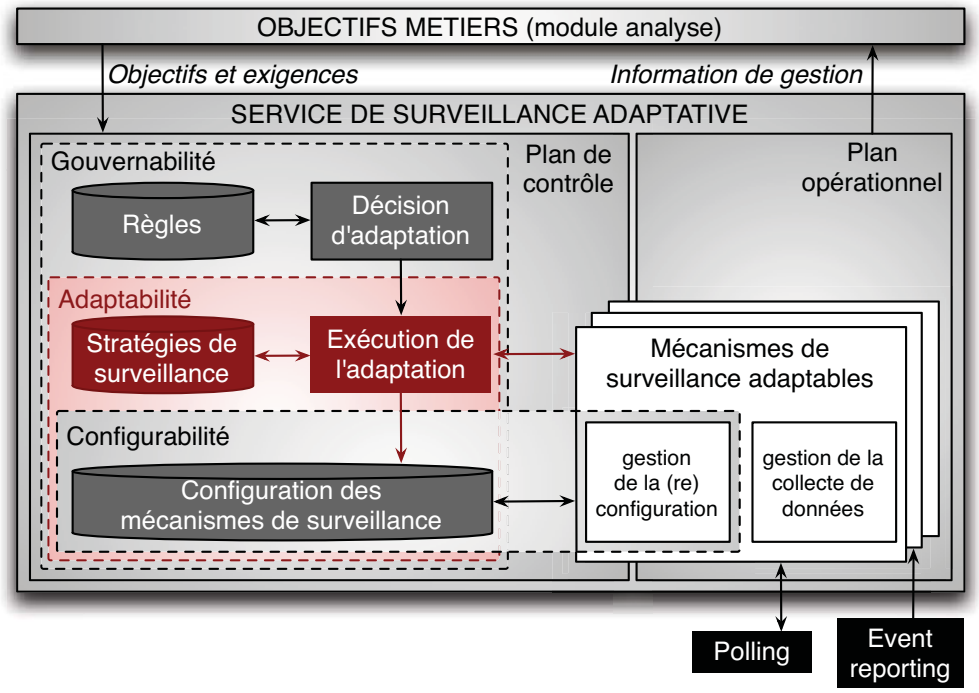

FIGURE 2.28: Le module d'« adaptabilité »

Cette capacité se positionne majoritairement dans le plan de contrôle de la surveillance adaptative et a pour but de lancer des opérations d'adaptation de la configuration, spécifiée dans le module de « configurabilité », en fonction des décisions d'adaptation que le module de « gouvernabilité » aura élues. L'« adaptabilité » a donc typiquement un rôle d'interface entre les deux autres modules.

Cette interface a été spécifiée et implémentée de sorte à répondre à l'ensemble des besoins de *well-definition*, de généricité, de modularité et de réutilisabilité et propose un prototype rendant la surveillance à la fois configurable et adaptable dynamiquement, ce dernier point correspondant donc bien à la définition de l'« adaptabilité » qui avait été faite :

Définition

L'**adaptabilité** est la capacité de modifier dynamiquement et sans interruption le comportement d'une activité de surveillance.

Le prototype doit maintenant être étendu de sorte à automatiser la prise de décision d'adaptation en fonction des contraintes de variations et des besoins d'optimisation. Ceci mène à l'élaboration de la dernière capacité : la « gouvernabilité ».

La gouvernabilité ou comment piloter l'adaptation de la surveillance

3

Après une présentation d'un des moyens permettant de piloter l'adaptation de la surveillance, la gestion à base de politiques, la couche « gouvernabilité » a été intégrée au démonstrateur afin de piloter l'adaptation de la surveillance de manière dynamique et sans interruption. Cette partie vise à démontrer la faisabilité de cette intégration avec l'évaluation et l'exécution de politiques Ponder2 dans un exemple simple et un cas d'étude plus concret.

3.1 Définition

Le pilotage de l'adaptation de la surveillance vise à décider si et comment l'activité de surveillance doit être ajustée, en mettant en application un ensemble d'actions de base. Pour ce faire, il est nécessaire de détecter un besoin d'adapter la surveillance.

Pour exprimer une telle logique, nous avons opté pour le paradigme de gestion à base de politiques (*Policy-Based Management* ou PBM). La gestion à base de politiques constitue une candidature sérieuse est la gestion à base de politiques, présentée dans cette partie, avec un état de l'art des langages permettant le mieux de répondre au problème du pilotage de l'adaptation de la surveillance. Ensuite, est présenté un exemple d'intégration de l'un de ces langages dans le *framework* pour en illustrer un fonctionnement complet, et démontrer la faisabilité et le bien-fondé de la capacité de « gouvernabilité », qui se définit comme suit :

Définition

La **gouvernabilité** est la capacité de l'infrastructure à pouvoir détecter un besoin d'adaptation de la surveillance et à déclencher opérationnellement cette adaptation.

3.2 Des politiques pour piloter l'adaptation de la surveillance

3.2.1 Définition et domaine d'application

Selon le Journal Officiel du 12 mai 2000, une politique d'entreprise est l'ensemble des orientations choisies pour conduire les affaires d'une entreprise. Autrement dit, une politique représente l'ensemble des règles qui tendent à réaliser les objectifs de l'entreprise.

Que représente donc une politique, à plus petite échelle, sur le système informatique de l'entreprise ? Un ensemble d'objectifs peut être positionné sur ce système, comme :

- « lancer une sauvegarde du système informatique à 17h chaque vendredi » (objectif de type gestion),
- « autoriser l'administrateur à aller modifier tout type de fichiers » (objectif de type contrôle d'accès),
- « restreindre l'accès d'un utilisateur ordinaire à son seul dossier personnel et au serveur partagé » (objectif de type contrôle d'accès).

Ces objectifs, quel que soit leur domaine d'application, sont considérés comme des règles gouvernant le comportement global d'un système informatique vis-à-vis de ces ressources. Dès lors que ces règles sont exprimées de manière intelligible pour le système informatique (c'est-à-dire à l'aide d'un langage), on les nomme « politiques ».

En somme, dans la gestion à base de politiques, une politique a un but d'automatisation d'activités de gestion et est définie de la manière suivante [Sloman94] :

Définition

Les **politiques** sont des règles gouvernant les différents comportements d'un système informatique. Ces règles doivent être exprimées de manière intelligible pour le système.

Une autre définition, plus technique, peut être donnée [Stone01] :

Définition

Les **politiques** spécifient quelles actions doivent être réalisées lorsqu'un ensemble de conditions sont rassemblées.

Les politiques informatiques ont tout d'abord été utilisées dans le domaine de la sécurité. Elles servaient alors à définir les droits d'accès de chaque utilisateur aux services et ressources du système. Ces politiques sont appelées « politiques d'autorisation ».

Par exemple, les règles suivantes peuvent être définies (exprimées ici en langage naturel pour plus de compréhension) :

- « Le chef d'entreprise peut éditer et modifier tout fichier présent sur les serveurs du système informatique. Aucune restriction ne lui est imposée. »
- « Le comptable de l'entreprise peut éditer et modifier tout fichier présent sur le serveur comptabilité, et peut éditer et modifier tout fichier contenu sur le serveur partagé de l'entreprise. Il ne peut pas avoir accès aux autres serveurs et encore moins éditer ou modifier les fichiers s'y trouvant. »
- « Un employé peut éditer et modifier tout fichier du serveur partagé. Il ne peut pas avoir accès aux autres serveurs et encore moins éditer ou modifier les fichiers s'y trouvant. »

Dans le domaine de la sécurité, les politiques sont de types CA (*Condition*/*Action* avec *Action* valant « oui » ou « non ») : leur but est simplement de vérifier si l'accès est autorisé, ou interdit. Ainsi, une politique de sécurité est très différente d'une politique de gestion, car il est demandé à la politique de gestion non plus seulement une autorisation d'accès, mais également un ensemble d'actions permettant de réaliser certaines tâches : ce sont les politiques de type CA* (*Condition*/*Actions*).

D'autres types de politiques intègrent la notion d'événement : ce sont les politiques de type ECA (*Event*/*Condition*/*Action*). Ces dernières sont particulièrement adaptées au domaine de la gestion car une politique peut être déclenchée par un grand nombre d'événements distincts et particuliers (la réception d'une indication, d'une alerte, une date et/ou une heure particulières). En sécurité, une politique est simplement déclenchée par une demande d'accès.

Ainsi, les politiques sont utilisées dans le domaine de la gestion de systèmes, entre autres, afin de spécifier toute action particulière déclenchée par un événement particulier ayant lieu dans une condition particulière. Ces politiques, dans ce cas appelées « politiques d'obligation », sont une généralisation des politiques utilisées dans le domaine de la sécurité.

Par exemple, il est possible de définir les actions suivantes à l'aide de telles politiques :

- « Il est 17h (événement) : si nous sommes vendredi (condition), lancer la sauvegarde automatique du système (action). »
- « Il est 12h (événement) : si nous ne sommes pas en week-end ou en période de congés (condition), lancer la fermeture de toutes les sessions utilisateurs (action). »

3.2.2 Classification des langages de politiques

De nombreux langages de politiques ont été inventés par les chercheurs. Pour les classifier et décider quel sera le langage utilisé au sein du prototype, des critères de distinction généraux doivent être dégagés.

3.2.2.1 Niveaux d'expression des politiques de gestion

[Kephart07] [Lobo99], entre autres, explicitent l'idée, de plus en plus répandue, que les politiques peuvent être idéalement exprimées sur trois niveaux d'abstraction : les actions, les objectifs, ou l'utilité. Et d'après [Derbel09], en combinant ces trois niveaux d'abstraction, une quatrième catégorie de politiques peut aussi être considérée : les politiques de comportement.

3.2.2.1.1 Les politiques d'action

Comme cela est représenté sur la figure 3.1, un événement se produisant sur le système déclenche indirectement les actions spécifiées dans la politique P. Celle-ci vérifie alors quelle est la condition satisfaite (respectivement w ou y), et lance le traitement correspondant (respectivement les actions x, g et z ou la seule action z).

Soit la politique suivante : « Il est 17h : si nous sommes vendredi, lancer la sauvegarde automatique du système, vérifier que la sauvegarde a réussi et éteindre. »

- L'événement déclencheur de la politique P est une certaine heure dans la journée : 17h.
- La politique ne doit toutefois être réalisée que le vendredi : ainsi la condition w correspond à « vendredi », et la condition y à « non vendredi ».

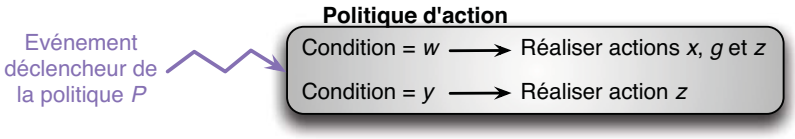

FIGURE 3.1: Politique d'actions

- Les actions sont à réaliser en fonction de la condition :
  - s'il est vendredi (condition w) alors la sauvegarde automatique du système est lancée (action x), il y a vérification du bon déroulement de la sauvegarde (action g) et il y a enfin extinction du système (action z),
  - sinon (condition y) aucune sauvegarde n'est lancée : le système est simplement éteint (action z).

Ce niveau d'expression de politique est le plus simple. La politique n'a pas de visibilité globale sur l'état du système ou sur son contexte d'exécution : elle est appelée pour vérifier une condition et appliquer machinalement les actions correspondantes, quels que soient le contexte d'exécution et l'état du système.

3.2.2.1.2 Les politiques d'objectif

Comme représenté sur la figure 3.2, ce type de politiques est plus abstrait. Ce ne sont pas les actions à réaliser qui sont décrites, mais les buts désirés et non désirés que la politique doit atteindre (ou non). Cette dernière définit alors d'elle-même la stratégie permettant de parvenir à un objectif à atteindre particulier (cette stratégie est représentée par la fonction s).

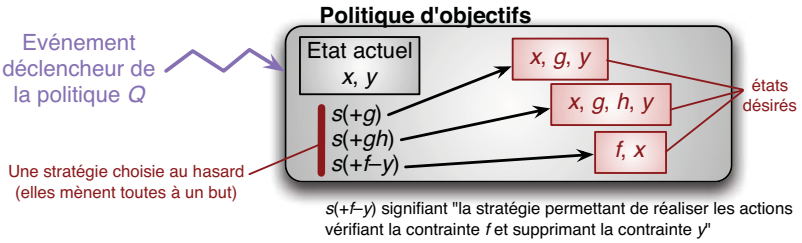

FIGURE 3.2: Politique d'objectifs

Dans l'exemple de la figure 3.2, le contexte courant remplit les contraintes x et y. A elles seules, ces deux contraintes ne forment pas un but désiré. Par contre, si le système remplissait en plus la contrainte g ou bien les contraintes g et h ou pas la contrainte y mais la contrainte f, l'état du système aurait atteint un but désiré. Ainsi, la politique définit trois stratégies en fonction du contexte, dont chacune permet d'accéder à un des buts désirés [1].

Soit la politique suivante : « Le temps de réponse doit être inférieur à 10ms, 20ms ou 30ms. »

- l'événement déclencheur de la politique est la réception d'une réponse,
- l'état du réseau et le temps de réponse (imaginons qu'il soit de 35ms) sont également connus lors de l'évaluation de la politique,

1. Il est également possible que plusieurs stratégies soient définies pour atteindre un but particulier.

- de là, la politique définit seule en fonction du contexte courant quelle est la stratégie à appliquer pour atteindre un des trois buts : baisser le temps de réponse (qui était de 35ms) soit à 30ms, soit à 20ms, soit à 10ms.

Aucun critère n'existe ici pour définir quelle stratégie doit être appliquée plutôt qu'une autre. L'important est d'atteindre un objectif adapté au contexte courant, quel qu'il soit.

3.2.2.1.3 Les politiques d'utilité

Ce type de politiques est une extension des politiques d'objectifs, dans le sens où ne sont définis que les buts désirés, et non les actions à accomplir pour y parvenir ; la politique doit être capable de définir elle-même la stratégie permettant d'atteindre un objectif désiré. Mais en plus, comme cela est visible sur la figure 3.3, des valeurs de désirabilité sont positionnées sur chacun des buts de sorte à influencer le choix de la stratégie à appliquer vers un état préféré préalablement spécifié [Walsh04]. Ces valeurs de désirabilité sont souvent assimilées à des priorités et sont ordonnées : dans l'exemple de la figure 3.3, la valeur de désirabilité la plus basse (1) correspond à la plus forte priorité.

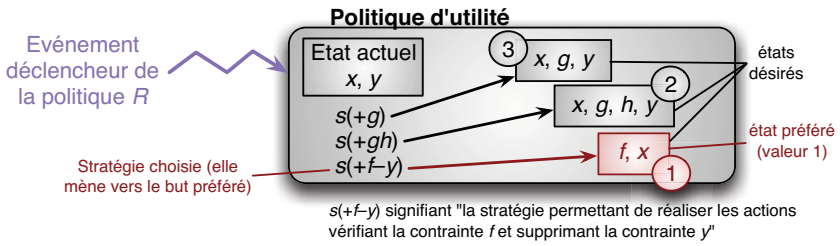

FIGURE 3.3: Politique d'utilité

Ainsi, les politiques d'utilité ont pour vocation d'optimiser le comportement du système.

Dans l'exemple de la figure 3.3, la stratégie adoptée est la troisième, car c'est grâce à elle que le système atteindra le but préféré (de valeur de désirabilité 1). Si cette stratégie n'est pas applicable, la politique fait en sorte d'adopter la stratégie menant à la valeur de désirabilité suivant directement 1 (2, puis 3, et ainsi de suite).

Soit la politique suivante : « Le temps de réponse doit être inférieur à 10ms, 20ms ou 30ms. Le temps de réponse doit être minimisé. »

- l'événement déclencheur de la politique est la réception d'une réponse,
- la politique connaît l'état du réseau et le temps de réponse (par exemple, il est toujours de 35ms),
- la politique a des buts à atteindre (baisser le temps de réponse de 35ms soit à 30ms, soit à 20ms, soit à 10ms), et doit appliquer la fonction d'utilité indiquant qu'il faut minimiser au maximum le temps de réponse (autrement dit, le but choisi est min (30ms, 20ms, 10ms) = 10ms). Une autre fonction d'utilité aurait pu être, par exemple, de choisir la valeur but de temps de réponse la plus proche du temps de réponse courant, soit ici de baisser le temps de réponse de 35ms à 30ms.

Grâce aux politiques d'utilité, il est possible de définir le comportement du système le plus adapté à son état courant. En effet, la valeur de désirabilité permet de définir quelle stratégie semble être la plus efficace pour atteindre le but préféré par rapport au contexte courant (cette stratégie a alors la plus forte valeur d'utilité).

Mais évidemment, la valeur de désirabilité doit être recalculée pour le contexte courant, ce qui implique l'utilisation de fonctions d'utilité pouvant être complexes à mettre en place.

3.2.2.1.4 Les politiques de comportement

Les politiques de comportement reprennent les trois niveaux d'abstraction sus-nommés. En effet, celles-ci définissent les actions à réaliser (le comportement attendu) pour atteindre le but désiré (représentant l'objectif de gestion) sélectionné par la fonction d'utilité mise en place (représentant l'objectif de haut niveau).

Les politiques de comportement de [Derbel09] sont composées de différents types de règles : des règles de configuration permettant de réorganiser l'infrastructure en fonction de l'environnement et des règles de surveillance permettant de contrôler les entités autonomes présentes sur l'infrastructure.

3.2.2.1.5 Discussion

Il apparaît évident au premier abord que les politiques d'objectif et de comportement semblent à privilégier par rapport aux politiques d'actions. Elles permettent en effet de spécifier des stratégies et conduisent à des choix plus optimaux pour adapter le comportement du système à l'état de son environnement.

Toutefois, des quatre niveaux d'abstraction proposés, seul le premier est à l'heure actuelle applicable et mis en application. En effet, la spécification d'un ensemble d'actions est tout à fait possible. Mais comment expliciter à un système une stratégie permettant d'arriver à un état particulier ? Certains langages tendent vers une telle solution – et sont donc à fortement privilégier dans le cadre de la surveillance adaptative –, en positionnant des priorités ou des valeurs de désirabilité sur des groupes d'actions particuliers, mais définissent toujours malgré tout les actions à réaliser pour parvenir à positionner le système dans un état particulier.

3.2.2.2 Domaine d'application

Les politiques sont principalement utilisées dans deux grands domaines. Historiquement, elles avaient été créées pour répondre à des besoins sécuritaires et permettaient de gérer toutes les problématiques de contrôles d'accès aux ressources, de filtrage de flux ou paquets, mais aussi de routage. Toutefois, ce domaine d'utilisation n'est pas adapté à la problématique d'adaptation de la surveillance.

La problématique s'inscrit en effet dans un contexte de gestion, et plus précisément de surveillance adaptative. Ainsi, les politiques de gestion recherchées sont de type ECA (*Event*/*Condition*/*Action*). Elles présentent en effet l'avantage de répondre à plusieurs besoins : à savoir la gestion des événements (au sens large : réception d'événements ou constatation d'horaire), la possibilité de réaliser diverses actions plus ou moins précises et une réactivité en temps réel.

3.2.2.3 Caractéristiques et fonctionnalités du langage

Plusieurs caractéristiques ou fonctionnalités du langage sont à prendre en compte dans le choix d'un langage de politiques. Ainsi, il faudra porter son attention sur la capacité du langage à :

- historiser les informations collectées à des fins de statistiques ou encore pour un apprentissage du comportement (de sorte à réitérer des actions en fonction d'un contexte particulier déjà rencontré),
- pouvoir envoyer des notifications d'événements.

Afin de rendre la surveillance la plus efficace possible, il est nécessaire de l'adapter au mieux à l'état de l'environnement considéré, ce qui nous pousse à prendre en compte les notions de prioritées et de stratégies vues précédemment.

3.2.2.4 Architecture et outillage

Outre le langage de politiques lui-même, les architectures et outillage qui le supportent sont des critères importants permettant de choisir un langage. L'absence ou le mauvais fonctionnement (ou toutes difficultés d'utilisation, de configuration, de paramétrage) peuvent être un réel frein à l'utilisation d'un langage, car une implémentation s'avèrerait nécessaire sur l'outil pour le rendre utilisable.

L'architecture avec laquelle doit être utilisé le langage est toute aussi importante. Certains langages n'en ont pas et les politiques sont stockées en dur et de manière anarchique dans l'environnement. Pourtant, plus l'architecture employée est structurée et modulaire, plus la gestion des politiques est facile à réaliser.

La plupart des langages utilisent un *framework* particulier s'adaptant parfaitement à eux et généralement inspiré de l'architecture de gestion à base de politiques la plus répandue et reconnue, PDP/PEP/PR, dans son mode *outsourcing*, proposée par le *Network Working Group* de l'IETF [Yavatkar00]. Cette architecture est schématisée en figure 3.4. Plusieurs éléments sont à y observer :

- le PR (*Policy Repository*) permet de stocker les politiques,
- le PDP (*Policy Decision Point*) permet d'évaluer les politiques applicables et de décider des actions à réaliser en fonction de celles-ci,
- le PEP (*Policy Enforcement Point*) est quant à lui chargé d'appliquer les décisions prises par le PDP.

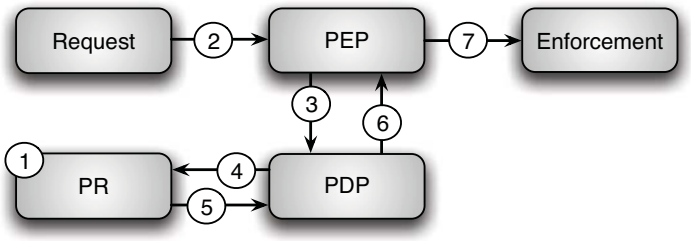

FIGURE 3.4: L'architecture PDP/PEP/PR

L'architecture en mode *outsourcing* fonctionne de la manière suivante : tout d'abord, les politiques sont écrites et stockées dans le PR ①, pour un accès ultérieur par le PDP. Lors d'une demande ②, le PEP transmet ③ la demande au PDP qui vérifie ④ dans le PR quelles politiques sont applicables. Une fois qu'il en a pris connaissance ⑤, il transmet sa décision ⑥ au PEP qui est alors chargé d'appliquer les actions correspondantes ⑦.

Enfin, toute architecture ou infrastructure utilisée par un langage de politiques peut être spécifique à une plateforme, ou au contraire générique. Ici, un environnement particulier a été choisi : CIM/WBEM. Il faut alors penser à vérifier si le langage peut être intégré à l'infrastructure en environnement CIM/WBEM.

3.2.3 Comparatif des langages de politiques principaux

Cette section dresse un bref état de l'art de divers langages de politiques pour déterminer, en fonction des critères de distinction précédemment établis, quel langage est le meilleur candidat à la problématique du pilotage de l'adaptation de la surveillance.

3.2.3.1 Etat de l'art

3.2.3.1.1 *Clark Policy Term*

Né en 1989, le *Clark Policy Term* est un des premiers langages utilisé pour la spécification de politiques de routage dans un réseau [Clark89]. Une politique écrite à l'aide de ce langage permet de déterminer le chemin que doit emprunter un paquet de données, dans le respect des exigences de coût, de qualité de services et des autorisations d'accès définies préalablement [Boutaba07].

Les politiques peuvent être aussi simples que complexes. Il s'agissait d'ailleurs du but premier de ce langage : être flexible de sorte à offrir la possibilité de représenter un très grand nombre de politiques de routage [Stone01].

Néanmoins, ce langage a plusieurs défauts :

- Pour commencer, le langage n'est pas naturel et implique de parfaitement connaître la sémantique d'un terme et donc de mémoriser la position de chacun des paramètres, surtout dans le cas où les politiques doivent imposer de grosses restrictions.
- Ensuite, le terme n'indique que la source et la destination du message : autrement dit, il n'indique pas le chemin explicite permettant d'arriver à destination. Ainsi, forcer un chemin revient à multiplier les termes, ce qui, combiné avec un langage peu naturel, peut engendrer un certain nombre d'erreurs (redondance dans l'expression des termes, inversion d'hôtes source et destination, etc.).
- Enfin, ce langage ne permet que de créer des filtres pour le routage des données et ne permet pas de réaliser des actions de modification sur les autres équipements du réseau. Toutefois, il est à la base d'autres langages de politiques plus riches, tels que PPL, qui peuvent grâce à lui être appréhendés plus facilement.

3.2.3.1.2 *Traffic Flow Policy Languages*

Les langages de politiques utilisant les flux de trafic ont pour principe de sélectionner le trafic réseau le plus approprié sur lequel appliquer une politique [Boutaba07]. Ainsi, la politique n'est qu'une simple représentation abstraite des fonctionnalités d'un nœud.

Dans l'ensemble, ce type de langage de politiques n'a qu'une efficacité limitée, et ce pour deux raisons majeures : tout d'abord, la visibilité se cantonne au seul nœud sur lequel porte la politique, et ensuite, le langage ne procure aucun moyen de savoir comment le système géré par les politiques va réagir.

Trois langages sont ici présentés.

3.2.3.1.2.1 *Pax Pattern Description Language*

Apparu en 1998, PAX-PDL (*PAX Pattern Description Language*) [Nossik98] est un langage de politiques de bas niveau permettant de définir et positionner des filtres sur des nœuds de sorte à

autoriser ou refuser des entrées de données. Pour que le passage de données soit autorisé, les entrées de données doivent correspondre à des *patterns* préalablement positionnés sur le nœud.

Bien que ce langage permette de décrire assez précisément les flux autorisés, il ne permet néanmoins pas de définir comment les flux doivent être reconnus : ce langage est purement descriptif.

La structure de PAX-PDL est basée sur celle du langage C, de sorte à fournir un certain nombre d'outils au langage : des séquences de phrases construites, des mots-clefs prédéfinis, et même la possibilité d'utiliser le préprocesseur.

PAX-PDL permet essentiellement de décrire des politiques de contrôle d'accès, agrémentées ou non de filtres. De bas niveau et visant essentiellement le domaine de la sécurité, PAX-PDL ne permet *a priori* pas de répondre à notre problématique.

3.2.3.1.2.2 *Path-based Policy Language*

Né en 2001, PPL (*Path-based Policy Language*) est un autre langage de politiques basé sur les flux de trafic. L'utilisation de PPL tend à respecter un ensemble de politiques positionnées sur plusieurs nœuds, de sorte à proposer à un flux de données un chemin cohérent et respectant l'ensemble des politiques définies [Stone01].

Le but de PPL est de représenter des politiques réseaux à un niveau abstrait – comme dans le cas du langage *Clark Policy Term* [Clark89] auquel il ressemble quelque peu – afin de pouvoir supporter l'hétérogénéité des équipements et réseaux. Le langage se veut également non ambigu afin que ces politiques puissent être traduites en logique formelle de sorte à détecter les conflits (la résolution des conflits tend à empêcher l'exécution de politiques ayant des buts contraires ou non compatibles).

Bien que PPL ajoute l'expression de chemins explicites par rapport à la syntaxe du *Clark Policy Term*, les emplacements de chacun des termes doivent ici encore être mémorisés pour spécifier correctement la politique. Toutefois, comme son prédecesseur, PPL est essentiellement axé contrôle d'accès (avec filtrage des flux).

3.2.3.1.2.3 *Simple Ruleset Language*

SRL (*Simple Ruleset Language*) est un autre langage de politiques basé sur les flux de trafic, apparu en 1999. Il est lui aussi basé sur la reconnaissance de *patterns*, mais offre en plus la possibilité de stocker des informations concernant les flux en transit, de sorte à pouvoir, si besoin, agir sur ces données *a posteriori* [Boutaba07].

SRL est utilisé dans un contexte bien particulier utilisant une architecture jugée difficile à appréhender [Brownlee99], mais la description des règles avec le langage SRL ne nécessite pas de connaissances concernant cette architecture. Le langage vise en effet à simplifier au maximum la création des règles par et pour les utilisateurs : ces derniers n'ont qu'à spécifier quel flux mesurer et quelles informations stocker. Notons par ailleurs que les aspects de sécurité des données ne sont ici pas du tout pris en compte.

Comme la plupart des langages de flux de trafic, SRL est surtout basé sur le contrôle d'accès et le filtrage des paquets. SRL ne permet donc pas de répondre à notre problématique.

3.2.3.1.3 *Policy Management for Autonomic Computing* : ACPL ou SPL ?

En 2005, IBM décide de développer PMAC (*Policy Management for Autonomic Computing*), une plateforme générique permettant de gérer des systèmes à base de politiques et dont le modèle d'information est inspiré de PCIM [Agrawal05b]. Cette plateforme supporte les principes de l'*Autonomic Computing* [Kephart03], qui définit un cadre de travail pour le *self-management* des systèmes (les fonctions *Monitor*, *Analyze*, *Plan* et *Execute* de la boucle MAPE). Ainsi la plateforme PMAC doit pouvoir être utilisée dans des cas divers tels que la gestion de la QoS, la reconfiguration, l'audit de systèmes...

Au centre de la conception de la plateforme PMAC, les politiques : elles sont ici perçues comme des entités de logique externe pouvant déterminer le comportement des systèmes gérés. Grâce à elles, la gestion à base de politiques peut parvenir à deux buts :

- les opérations effectuées sur ou par les ressources peuvent être guidées de sorte à suivre le comportement spécifié par la politique,
- les opérations effectuées sur ou par les ressources peuvent être reconfigurées dynamiquement pour réagir à certains événements survenus dans leur environnement, afin de s'adapter à l'environnement actuel.

La plateforme PMAC est implémentée en Java et supporte deux langages de politiques :

- ACPL [Agrawal05b] (*Autonomic Computing Policy Language*) est un langage de politiques basé sur XML (d'où sa verbosité) utilisant les expressions ACEL [Agrawal05a] (*Autonomic Computing Expression Language*). ACPL est un langage fortement typé pouvant être analysé et validé par n'importe quel éditeur XML. ACPL permet en effet d'exprimer la plupart des conditions « communes » de politiques tout en respectant les conventions XML.
- SPL[2] [Chilukuri06] (*Simplified Policy Language*) est un langage de politiques concis et plus facilement utilisable que ACPL : sa syntaxe a été créée pour le développement de politiques basées sur l'*Autonomic Computing* et facilement compréhensibles par l'homme. Plus simple d'utilisation, il ne suit pas les conventions XML. Attention cependant aux erreurs de syntaxe.

L'interpréteur de la plateforme PMAC ne comprend que le langage ACPL. Si une politique a été écrite en SPL, un traducteur intégré à la plateforme convertit la politique en ACPL avant de l'interpréter. Ici, l'intérêt de SPL est de simplifier l'écriture de la politique pour le gestionnaire des politiques qui désirerait ne pas utiliser l'éditeur ACPL intégré à la plateforme PMAC. Enfin, l'utilisation de SPL permet à l'utilisateur de ne pas avoir à se familiariser avec le langage ACPL et son *XML Schema*.

PMAC propose deux langages intéressants permettant de guider et reconfigurer les opérations à réaliser par les ressources du système sous-jacent et gère les concepts de l'*Autonomic Computing*. Toutefois, les langages ne permettent pas de gérer les événements et les politiques doivent donc être évaluées régulièrement.

3.2.3.1.4 *Policy Description Language*

En 1999, Bell Labs considère qu'une politique est une fonction décrivant la stratégie permettant d'accomplir un ensemble particulier de buts [Lobo99]. De ce fait, PDL (*Policy Description Language*) devient un des premiers langages à spécifier des politiques s'appuyant fortement sur la logique et

2. SPL ne doit pas être confondu avec CIM-SPL qui est une version ultérieure de ce langage associée avec CIM.

rédigées sous la forme de règles ECA[3], rendant le langage à la fois succint, simple à spécifier et efficacement implémentable [Agrawal07] [Boutaba07], les politiques PDL pouvant être compilées au sein de classes Java [Virmani00].

Les auteurs du langage [Lobo99] envisagent différents niveaux d'abstraction pour spécifier des politiques, le plus haut niveau étant la spécification en langage naturel (il s'agit alors d'objectifs de gestion), et le plus bas niveau étant la spécification de règles de production bas niveau. Selon eux, PDL se situe à un niveau intermédiaire : ils considèrent en effet, entre l'environnement et le *Policy Server*, un service intermédiaire dont le but est de communiquer au *Policy Server* les changements survenus dans l'environnement pouvant nécessiter l'application d'une politique.

Ajoutée à cela, la définition des politiques est ici réalisée en cours d'exécution. Le *manager* procède, pour ce faire, en deux étapes :

- sont récupérés auprès du *Policy Server* les événements que le système est capable de surveiller, les actions pouvant être invoquées par la future politique, et l'ensemble des fonctions que possède le système pour évaluer l'état de l'environnement,
- une fois ces informations récupérées, le *manager* peut spécifier la politique puis l'implémenter dans le *Policy Server*.

Il est également à noter que le *manager* garde, pour des besoins de gestion, un historique des événements passés. Ainsi, si un événement se produit à nouveau, le *manager* ne déclenche pas à nouveau une récupération des informations précédemment mentionnées : il connaît déjà ces informations et sait quelle politique mettre en place. Il se contente donc de spécifier et implémenter directement la politique correspondante.

PDL introduit deux types de politiques :

- les politiques exécutant des actions sur l'environnement (éteindre tel équipement, augmenter la bande passante associée aux utilisateurs de la classe `Gold`...),
- les politiques déclenchant un événement (indiquer qu'il faut passer en fonctionnement restreint, alerter d'un non fonctionnement de tel équipement). Ces politiques sont particulièrement utiles lorsque des événements doivent être corrélés : si une alerte est reçue plusieurs fois, elle est agrégée et un seul événement est émis par la politique, précisant à la fois le sujet de l'événement et le nombre d'alertes qui ont été reçues.

PDL offre des fonctionnalités intéressantes et est particulièrement bien adapté au domaine de la gestion, notamment grâce à sa gestion des événements. De plus PDL possède une syntaxe simple.

3.2.3.1.5 *eXtensible Access Control Markup Language*

XACML (*eXtensible Access Control Markup Language*) est un langage de politiques orientées sécurité créé par OASIS[4] en 2003 [Godik03]. La création de ce langage reposait sur deux buts essentiels : standardiser la gestion des contrôles d'accès par l'utilisation de XML et renforcer l'interopérabilité entre systèmes hétérogènes.

De ce fait, le standard XACML apporte deux langages :

- un langage déclaratif permettant de spécifier des politiques de contrôles d'accès : ces politiques sont exprimées par un ensemble de règles (il peut n'y en avoir qu'une) à vérifier, et

3. une politique ECA peut se traduire en langage naturel par l'expression : sur l'arrivée de l'événement E, et si la condition C est remplie, alors réaliser l'action A.

4. L'*Organization for the Advancement of Structured Information Standards* a pour but de développer des standards universels pour faciliter les communications dans le cadre de l'*e-business*.

optionnellement d'actions à réaliser,
- un langage requête/réponse permettant d'autoriser ou refuser l'accès aux ressources par le système demandeur, en fonction des politiques préalablement spécifiées.

Si XACML est retenu comme un candidat pour répondre à la problématique de surveillance adaptative, alors qu'il est essentiellement axé contrôle d'accès, c'est parce qu'il est *attribute-based* et s'appuie sur une architecture modulaire :

- Eu égard à cette volonté de normaliser la gestion du contrôle d'accès, la définition du langage s'appuie sur le principe qu'une politique s'applique sur un ensemble d'attributs : un sujet, une cible, un rôle, une tâche. Non restrictif, ce modèle permet d'appliquer une politique sur tout élément jugé pertinent.
- L'architecture est proche de l'architecture PDP/PEP/PR. Sur la figure 3.5, divers composants peuvent être repérés :
  - le *Context Handler* est chargé de la communication (et de la traduction si nécessaire) entre le PEP et les autres éléments pour une récupération des données nécessaires à l'évaluation de la politique,
  - le PAP (*Policy Administration Point*) permet de créer et stocker les politiques de sécurité,
  - le PEP (*Policy Enforcement Point*) contrôle les accès aux ressources (en fonction des autorisations et refus donnés par le PDP),
  - le PIP (*Policy Information Point*) permet de stocker ou récupérer les valeurs d'attributs et données nécessaires pour l'évaluation des politiques par le PDP,
  - le PDP (*Policy Decision Point*) évalue les politiques applicables et décide si l'accès est autorisé ou non.

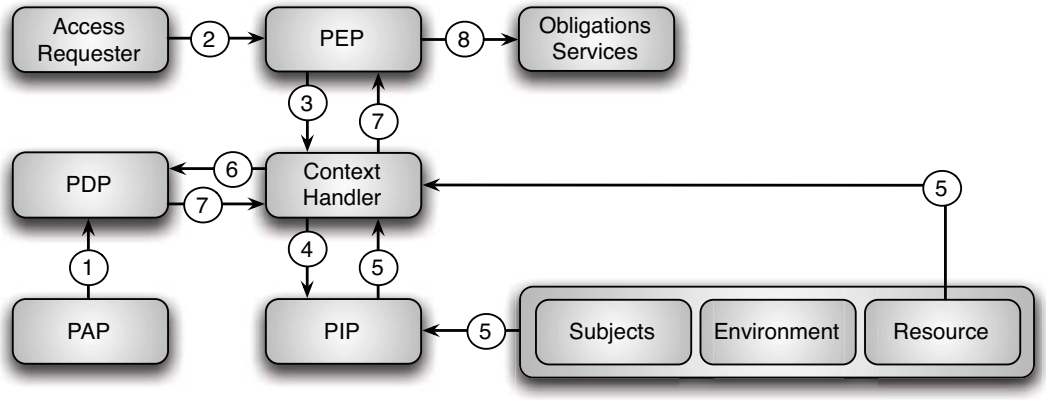

FIGURE 3.5: L'architecture autour de XACML

L'initialisation se fait en ①, au niveau des PAP et PDP : les politiques sont écrites et stockées par le PAP et rendues accessibles au PDP, pour que celui-ci puisse plus tard les évaluer. Lors d'une demande d'accès ②, le PEP transfère la demande d'accès au *Context Handler* ③, qui demande à son tour si nécessaire au PIP les valeurs d'attributs et les données nécessaires pour évaluer la politique ④. Une fois les données reçues par le *Context Handler* ⑤, celui-ci les regroupe et les envoie au PDP ⑥ pour qu'il évalue la politique et rende sa décision, qui est transférée au PEP ⑦. Le PEP met alors en application la politique : autorisation ou refus d'accès et s'il y a lieu réalisation des actions spécifiées par la politique (appelées *obligations*) ⑧.

XACML est un langage permettant d'écrire des politiques à un niveau de granularité très fin (mais de grande verbosité, ce qui rend son utilisation difficile). Bien qu'essentiellement dédiées au contrôle d'accès, les politiques XACML peuvent être détournées pour fonctionner dans un esprit davantage orienté gestion, grâce à son modèle *attribute-based*. De plus la gestion des événements est rendue possible en XACML, comme le montrent [Laborde08].

3.2.3.1.6 *CIM-Simplified Policy Language*

Le *CIM-Simplified Policy Language* repose sur deux modèles, CIM et PCIM, standardisés par le DMTF [Agrawal07] :

- CIM (*Common Information Model*) est un modèle orienté objet défini par le DMTF et permettant de décrire un système géré dans son intégralité. CIM est utilisé dans un but d'unification et est de ce fait indépendant des plateformes, des logiciels, des systèmes et de l'architecture. Le modèle utilise des classes d'objets et des mécanismes de relations utilisant les concepts objets (héritage, composition, dépendances).
- Le *CIM Policy Model* [DMTF03b] utilise la modélisation CIM afin de représenter simplement les systèmes de gestion à base de politiques. Le concept de politiques est illustré sur la figure 3.6.

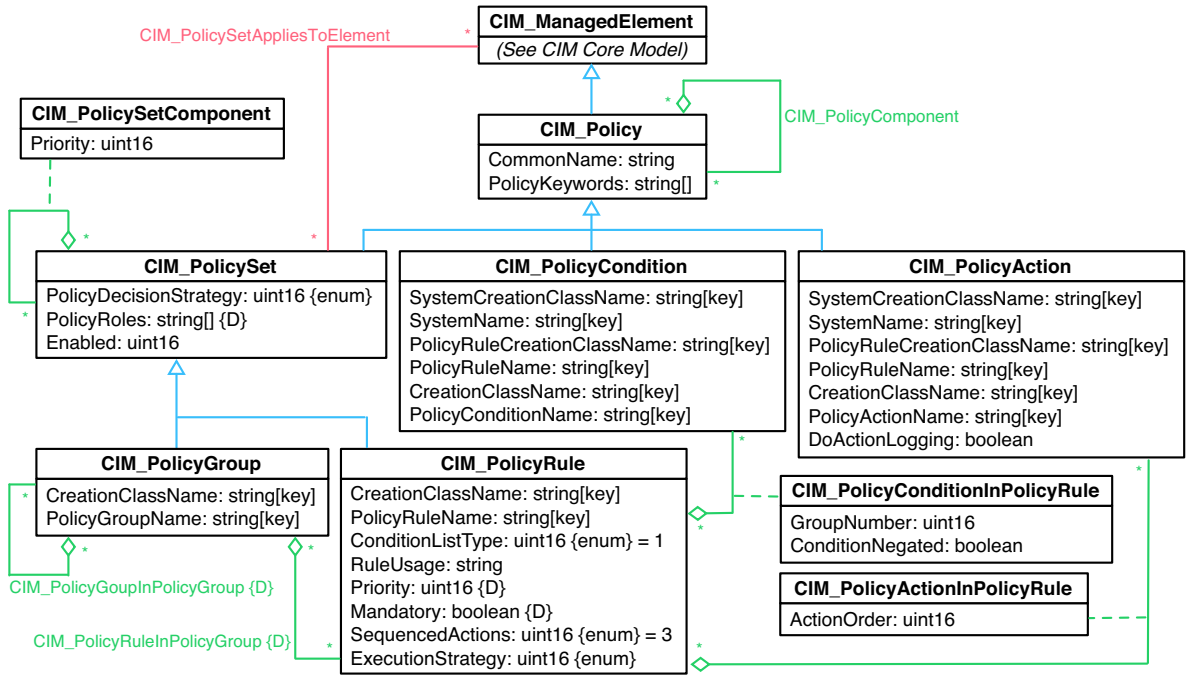

FIGURE 3.6: Représentation CIM du concept de politiques (version 2.7.0)

Tout comme CIM, le *CIM Policy Model* est indépendant des plateformes, des logiciels, des systèmes et de l'architecture. De plus, il est abstrait et nécessite d'être implémenté. C'est à cette fin qu'a été créé le langage CIM-SPL.

CIM-SPL est fortement inspiré du *CIM Policy Model*. Ainsi, en CIM-SPL, une politique `CIM_Policy` est composée de règles `CIM_PolicyRule`. Une règle `CIM_PolicyRule` est composée au moins d'une condition, `CIM_PolicyCondition`, et d'une action, `CIM_PolicyAction`. Plusieurs règles `CIM_PolicyRule` peuvent être regroupées en un groupe de politiques `CIM_PolicyGroup`. De plus un

`CIM_PolicyGroup` peut contenir d'autres `CIM_PolicyGroup` [DMTF09].

CIM-SPL est ainsi devenu un standard du DMTF et apporte les moyens nécessaires à la spécification de règles de politiques du type « si condition alors actions » (autrement dit CA+, CIM-SPL ne gérant pas les événements) permettant de gérer des ressources informatiques.

Le langage CIM-SPL est riche et peut permettre de répondre à la problématique de la gouvernabilité de l'adaptation de la surveillance.

Tout d'abord, CIM-SPL est parfaitement intégré au sein d'un environnement CIM/WBEM, du fait de sa conception respectant toutes les contraintes des modèles CIM et PCIM. Fortement contraint, il ne peut être que parfaitement compatible avec un tel environnement. Par exemple, des valeurs de propriétés peuvent être modifiées à l'aide de politiques CIM-SPL. De plus, les politiques CIM-SPL sont tout à fait compatibles avec une adaptation des ressources informatiques en cours d'exécution.

Le modèle CIM était à l'origine créé pour des besoins de gestion de ressources informatiques. CIM-SPL est utilisé dans le même esprit et n'est pas contraint par des mécanismes pro-sécurité, même s'il peut aussi être utilisé dans un tel contexte. En effet, la plupart des langages de politiques ont d'abord été créés pour la sécurité et sont donc fortement orientés dans leur syntaxe vers des contrôles d'accès et filtrage de paquets. Même s'ils peuvent être détournés de leur objectif premier, la syntaxe reste contrainte par des mécanismes sécuritaires pouvant rendre difficile l'implémentation d'actions de gestion.

Enfin, le langage CIM-SPL intègre les notions de priorités et de stratégies permettant d'instaurer des préférences dans les actions à réaliser et ce en fonction de l'état courant du système géré. Ces politiques permettent de favoriser l'autonomie du système en lui proposant une stratégie de choix et plusieurs cas possibles distingués par des valeurs de désirabilité (ce qui revient à établir des politiques d'objectifs).

Toutefois, CIM-SPL n'est pas de type ECA, mais seulement de type CA+. Autrement dit, il ne gère pas les événements de manière directe (seulement au sein de conditions d'applicabilité) et implique ainsi une réévaluation de la totalité des politiques tous les x temps. Ceci peut être un réel frein à l'utilisation temps réel notamment, ainsi qu'à l'intégration des notifications comme déclencheurs de politique.

Autre bémol, l'outil OpenSource Apache Imperius – la seule plateforme libre et gratuite permettant d'écrire et stocker des politiques CIM-SPL – n'est plus maintenue par ses créateurs dans l'incubateur d'Apache, qui a donc procédé à une clôture du projet.

3.2.3.1.7 Ponder

Apparu en 2000, le langage Ponder propose une vision différente par rapport aux autres langages de politiques créés jusqu'alors. Les auteurs de Ponder, lors de leur démarche de conception, ont estimé que les concepts de « pouvoir » et de « motivation » doivent être coinjointement pris en compte par le langage [Moffett91]. Ces deux concepts représentent les préconditions à respecter impérativement pour qu'une action soit accomplie : l'agent devant réaliser une action dictée par une politique doit avoir reçu la demande – ou avoir par lui-même l'intention – de réaliser l'action (motivation) et doit avoir l'autorisation de la réaliser (pouvoir). De fait, le langage ne s'adresse plus seulement à un contexte de sécurité, comme la plupart des langages existant, mais aussi à un contexte plus large de *management*, en intégrant des politiques de gestion.

En partant de ce constat, diverses exigences ont été positionnées [Damianou01] :

- Le langage doit avoir un champ d'action suffisamment large pour supporter à la fois des contraintes de gestion et de sécurité : il est constitué d'un ensemble de politiques hiérarchisées et regroupées selon leur rôle respectif dans l'organisation (*role-based access control* pour les aspects de sécurité, et *role-based management* pour les aspects de gestion).
- Le langage doit permettre de spécifier des politiques dans de très grands systèmes et doit être extensible. Une solution étant la gestion d'objets et de collections d'objets, les auteurs décident qu'un langage orienté objet est le plus adapté à leur domaine d'application.
- Le langage doit permettre l'analyse des politiques pour la résolution des conflits et des inconsistances dès que les politiques ont été spécifiées dans le système. De plus, il apparaît que des liens doivent exister de sorte à savoir ① à partir d'un objet, quelles sont les politiques qui lui sont applicables, et ② à partir d'une politique, quels sont les objets sur lesquels elle doit s'appliquer. Ainsi, le langage doit être déclaratif.
- Enfin, le langage doit être compréhensible, et simple à utiliser.

Au terme de dix années de réflexion et de conception, les auteurs proposent Ponder, défini comme un langage déclaratif, orienté objet, flexible, expressif et extensible, permettant de créer des politiques de sécurité et de gestion pour des systèmes distribués [Damianou01] [Boutaba07] [Agrawal07]. Le langage utilise la notion de « domaines » (représentés dans les politiques de la même façon qu'un répertoire) pour regrouper et hiérarchiser tous les éléments Ponder (sujets, cibles, politiques...).

Les politiques exprimables en Ponder sont soit axées « sécurité » (politiques de contrôle d'accès), soit axées « gestion » (politiques d'obligation) :

- Les « politiques de contrôle d'accès » peuvent être décomposées en différents sous-groupes :
  - Les « politiques d'autorisation » sont des règles d'accès utilisées pour définir les actions qu'un sujet est autorisé à réaliser sur un (ou plusieurs) objet(s) cible(s).
    Ce type de politique permet de contrôler les accès aux ressources du réseau.
  - Les « politiques d'autorisation négative » (ou d'interdiction) sont des règles d'accès utilisées pour définir les actions qu'un sujet **n'**est **pas** autorisé à réaliser sur un (ou plusieurs) objet(s) cible(s).
    Ponder fait partie de ces langages permettant de spécifier des politiques d'interdiction. En effet, elles offrent une plus grande cohérence avec les objectifs de gestion et besoins exprimés par les utilisateurs. Toutefois, il est important de gérer correctement ces politiques et tenter de repérer (et minimiser !) les conflits et incohérences pouvant apparaître entre politiques d'autorisation et d'interdiction. Une analyse est impérative.
  - Les « politiques de filtrage d'information » sont des politiques d'autorisation positive ou négative auxquelles un filtre est ajouté. Ce filtre n'est qu'une condition visant à restreindre un peu plus le champ d'action de la politique.
    Ainsi, l'utilisation des filtres permet d'ajouter un niveau de granularité pour restreindre les accès.
  - Les « politiques de délégation » sont associées aux politiques d'autorisation. Elles permettent de spécifier les droits d'accès pouvant être délégués par un sujet. Celui-ci ne peut d'ailleurs déléguer que des droits qu'il possède.
  - Il est possible d'autoriser ou de refuser la réalisation d'actions grâce aux politiques d'autorisation positive et négative. Mais parfois, il est utile d'interdire de manière ponctuelle

et temporaire certaines actions. C'est le principe des « politiques de restriction temporaire ».

- Les « politiques d'obligation » décrivent les actions devant être réalisées lorsqu'un événement particulier se produit. Ces politiques reprennent le schéma ECA (*Event*/*Condition*/*Actions*). Il en existe de plusieurs types :
  - les « politiques de sécurité » permettent de spécifier les actions à réaliser (et par qui elles doivent l'être) quand une règle de sécurité est violée,
  - les « politiques de gestion de la QoS » permettent de gérer la qualité de service offerte aux utilisateurs, de programmer des sauvegardes de systèmes, ou de réaliser des configurations logicielles,
  - les « politiques d'historisation » permettent de spécifier quelles sont les informations à historiser, où et par qui.

Diverses possibilités issues du monde objet sont ajoutées au langage de sorte à simplifier la création des politiques :

- Généricité et héritage : une politique dite « générique » est spécifiée comme une fonction à appeler. Lors de l'appel, seuls le sujet et la cible sont à spécifier, les actions à réaliser étant communes pour tous [5]. Ceci simplifie grandement le travail de spécification des politiques. De même il est possible de spécifier des politiques héritant d'autres politiques (grâce au mot-clef `extends`) : ce concept permet l'encapsulation et la réutilisation de politiques déjà existantes.
- Groupement de politiques par domaines : il est possible de créer des collections de politiques basées chacune sur une même cible, un même domaine, ou une même application. Cette structuration/hiérarchisation des groupes de politiques permet de refléter la structuration organisationnelle et de préserver la manière dont opèrent les administrateurs de système (gestion des rôles).
- Le langage gère les collections d'objets et est flexible, de sorte à supporter des architectures diverses (par exemple, une architecture CIM/WBEM [Lymberopoulos04]).
- La consistance des politiques portant sur une cible peut être validée par vérification des possibilités de ladite cible : exigences positionnées avant déploiement (par le biais de métapolitiques exprimées en langage OCL) et contraintes dues à l'état actuel du système.

Comme CIM-SPL ou XACML, Ponder répond aux attentes de la problématique de gouvernabilité de l'adaptation de la surveillance.

Ce langage est riche, et permet de spécifier de nombreux types de politiques de sorte à élargir le champ d'actions du langage. Son appartenance au monde objet lui apporte également de nombreux outils, tels que la généricité, l'héritage (hiérarchisation des politiques), le groupement en collections. Mais ce qu'il faut noter par dessus tout, c'est que Ponder peut être utilisé dans une approche de *self-management*. Dans le *Ponder framework*, l'implémentation est séparée de la gestion des politiques de sorte à laisser la possibilité de modifier dynamiquement la stratégie de gestion (et donc modifier le comportement du système).

En effet, tous les éléments – les politiques, leurs rôles respectifs et les relations existant entre elles – sont implémentés en tant qu'objets. De ce fait, ces éléments sont à considérer comme de simples objets auto-gérables. Ainsi il devient possible d'écrire des politiques d'autorisation permettant de

5. Imaginons par exemple le cas d'une promotion d'un employé. A chaque fois, la politique est la même : seul le sujet change. Spécifier une politique générique pour la promotion de trente employés permet de ne créer qu'une politique et de l'appliquer trente fois plutôt que de la recréer trente fois.

spécifier qui a le droit d'ajouter, supprimer ou éditer des rôles, des relations, voire même d'autres politiques. Ajouté à cela, il est également possible de créer des politiques d'obligation spécifiant des actions de modifications portant sur d'autres politiques dès qu'un événement particulier se produit, comme par exemple, l'activation d'une politique permettant de s'adapter à un nouveau contexte de travail (passage d'état de fonctionnement normal, en état de fonctionnement réduit).

Notons également, d'après [Moffett91], que des politiques Ponder peuvent être « enregistrées » comme non modifiables (elles sont alors codées « en dur ») de sorte à les rendre permanentes ; les politiques modifiables sont, elles, enregistrées dans le gestionnaire des politiques de sorte à pouvoir être activées, désactivées, modifiées dynamiquement.

Enfin, une implémentation de politiques Ponder dans une architecture CIM/WBEM a été expérimentée par [Lymberopoulos04] en 2004. Les politiques Ponder peuvent en effet extraire des données d'un modèle CIM en cours d'exécution afin de connaître l'état actuel du système. Ces données sont alors évaluées par la politique pour adapter le comportement des équipements.

3.2.3.2 Quel langage choisir ?

Le tableau 3.1 dresse un comparatif de ces langages en fonction des critères précédemment établis.

	ECA (Niveau d'expression)	Politiques de gestion	Caract. et fonctionnalités	Architecture et outillage	Outillage CIM existant
Clark Policy Term	☹	☹	☹	☹	☹
PAX-PDL	☹	☹	☹	☹	☹
PPL	☹	☹	☹	☹	☹
SRL	☹	☹	☺	☹	☹
ACPL	☹	☺	☺	😐	😐
SPL	☹	☺	☺	😐	😐
PDL	☺	☺	☺	☹	☹
XACML	☺	😐	☺	☺	😐
Ponder	☺	☺	☺	☺	😐
CIM-SPL	😐	☺	☺	☺	☹

TABLE 3.1: Comparaison des langages de politiques

CIM-SPL aurait pu être le langage convenant le mieux à la problématique du pilotage de l'adaptation de la surveillance. Mais aucun outil libre, gratuit et pérenne ne permet actuellement son utilisation. En effet, le logiciel Apache Imperius, seul outil gratuit et libre instrumentant le langage CIM-SPL, n'est plus actif ni maintenu. Le choix doit donc être fait entre XACML et Ponder, les deux autres langages qui se démarquent ici. Comparons-les en fonction des critères établis.

- **Type de politique et niveau d'expression**. XACML est avant tout un langage de politiques de sécurité : son but premier est d'autoriser ou d'interdire l'accès sur réception d'une demande. Toutefois, il est possible de spécifier un ensemble d'actions à réaliser pour chaque politique : la politique est alors dite d'obligation. Le langage peut donc être détourné pour répondre au critère « Politiques de gestion ». Par contre, Ponder a été défini comme un langage de politiques de gestion et permet de spécifier un certain nombre d'actions sur réception d'un événement particulier.

Le choix se porte sur **Ponder**.

- **Caractéristiques principales : stratégies, priorités**. Ponder intègre des mécanismes de priorités, mais pas de stratégies, alors que XACML utilise conjointement stratégies et priorités.
  Le choix se porte sur **XACML**.
- **Outillage et architecture**. XACML possède une architecture de type PDP/PEP/PR spécifique au langage, impliquant une évaluation des politiques selon un protocole très précis. Cette architecture joue grandement en faveur de ce langage. Côté outillage, l'API Java Sun XACML permettrait *a priori* de pouvoir implémenter une architecture XACML, et de faire la liaison entre le client Java SBLIM et le serveur CIM : l'évaluation de politiques XACML serait alors déclenchée sur réception de demandes d'accès (ce qui représente le point faible de cette méthode), et permettrait de lancer des actions particulières : les opérateurs développés grâce à l'API Java SBLIM. Le problème réside ici en la traduction de tout événement en demandes d'accès, en l'utilisation du langage XACML, de grande verbosité, et en l'obligation d'appeler en XACML des méthodes Java, ce qui n'est pas évident au premier abord.
  D'un autre côté, le *Ponder framework* est une grosse infrastructure dépendant de l'environnement (CIM ou LDAP). La dernière utilisation du *framework* avec l'environnement CIM/WBEM remonte à une dizaine d'années, mais démontre de la compatibilité des environnements. Pour pallier à la dépendance entre l'infrastructure Ponder et l'environnement, la deuxième version de Ponder, Ponder2, offre une architecture décorrellée de l'environnement, et une API Java, permettant d'intégrer des politiques Ponder en utilisant directement Java. L'intérêt ici est d'intégrer directement l'infrastructure Ponder2 à l'application Java déjà développée.
  Le choix se porte sur **Ponder**.

Plus orienté *business*, Ponder2 offre en natif les moyens de spécifier des politiques de type ECA (aux niveaux politiques d'obligation et politiques d'historisation), telles qu'elles sont requises dans le cadre de ces travaux. De plus, l'API Java proposée est une réponse technique tout à fait convenable et qui s'intégrerait correctement au *framework* déjà existant.

3.3 Implémentation

Cette section présente un exemple simple d'utilisation choisi pour piloter la surveillance à l'aide de politiques Ponder2 chargées de prendre les décisions d'adaptation. Elle a pour vocation de démontrer la faisabilité de l'utilisation de Ponder2 au sein du *framework* d'autre part.

3.3.1 Exemple simple d'utilisation

3.3.1.1 Quels sont les objectifs de l'exemple ?

L'objectif de haut niveau choisi est de « connaître avec précision l'état courant du réseau lorsqu'il fonctionne correctement ». Ainsi, la qualité des informations doit être maximale pour obtenir une vision la plus réaliste possible de ce système. Mais lorsque le système est dégradé, un deuxième objectif stipule que « toute surcharge de trafic due à l'observation du système doit être évitée, pour laisser le système se décongestionner, tout en cherchant à diagnostiquer au mieux le problème ».

Techniquement, il s'agit d'utiliser les mécanismes de *polling* et d'*event reporting* au mieux de sorte à répondre à tous les objectifs, quel que soit l'état du système :

- Dans l'état normal, un *listener* de notifications surveille un changement de comportement des notifications (des envois sporadiques), et un mécanisme de *polling* interroge une cible quelconque de manière périodique,
- dans l'état dégradé, le *listener* doit détecter un retour à un comportement normal (silencieux), le mécanisme de *polling* arrête d'interroger sa cible pour ne pas encombrer le trafic, et un autre mécanisme de *polling* est lancé pour diagnostiquer la panne.

Ces objectifs de gestion sont des règles qui peuvent être traduites en politiques Ponder2.

3.3.1.2 Comment piloter l'adaptation ?

La figure 3.7 illustre la façon dont cette politique peut être supportée par le *framework*. Ceci permettra à terme de démontrer la faisabilité de la couche « gouvernabilité » avec deux objectifs de gestion traduits en politiques Ponder2.

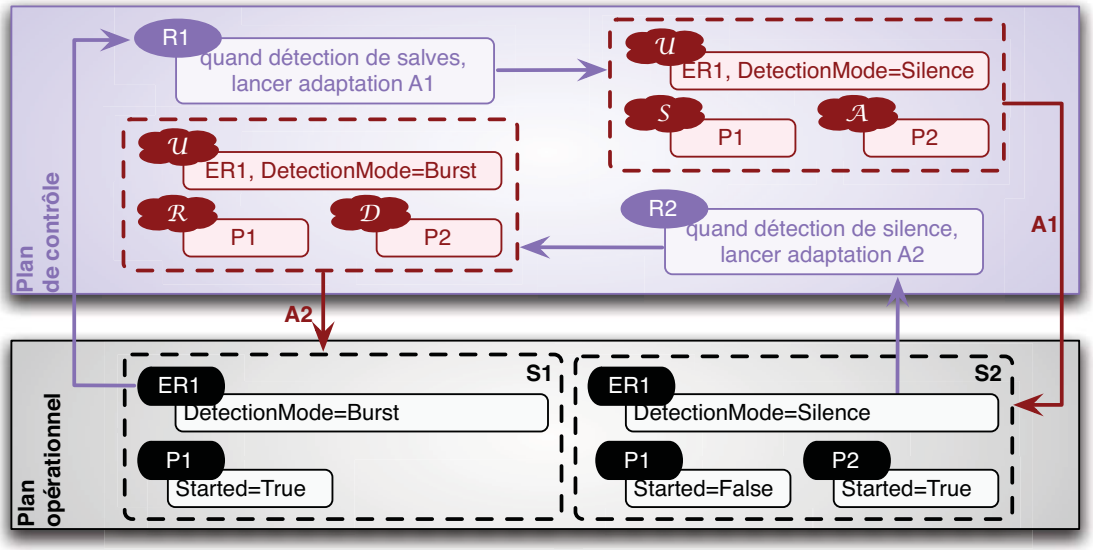

FIGURE 3.7: Cas simple d'utilisation

La figure 3.7 présente les différents plans du *framework* de surveillance adaptative, à savoir le plan opérationnel et le plan de contrôle.

Le rectangle en pointillés de gauche dans le plan opérationnel décrit l'état initial du système, qui peut être considéré ici comme « normal » (stratégie S_1) : un *polling* est positionné pour interroger une cible périodiquement et un mécanisme d'*event reporting* est activé pour détecter un comportement sporadique du système, qui est significatif d'une panne système.

Lorsqu'une salve est détectée, l'évaluation de la règle R_1 est déclenchée pour lancer la première adaptation A_1 permettant de changer de stratégie d'observation :

$$S_1 = \{(ER_1, C_1, running), (P_1, C_2, running)\} \xrightarrow{A_1}$$

$$S_2 = \{(ER_1, C_1', running), (P_1, C_2, pending), (P_2, C_3, running)\}$$

... avec deux paramètres de configuration particuliers à mettre en évidence (les autres ne sont pas importants pour l'exemple) :

- $\langle \texttt{DetectionMode} = \textit{Burst} \rangle \subset C_1$
- $\langle \texttt{DetectionMode} = \textit{Silence} \rangle \subset C'_1$

De ce fait, cette adaptation consiste à suspendre le *polling* interrogeant périodiquement ses cibles (grâce à l'opérateur $\mathscr{S}$), à en ajouter un nouveau chargé de diagnostiquer le problème (grâce à l'opérateur $\mathscr{A}$) et à modifier la configuration du *listener* de notifications pour qu'il détecte une situation de silences (grâce à l'opérateur $\mathscr{U}$). L'adaptation permet alors d'atteindre la stratégie de surveillance S_2 :

$$A_1 = \left\{\mathscr{U}\left(ER_1, silence\right), \mathscr{S}\left(P_1\right), \mathscr{A}\left(P_2, C_3, running\right)\right\}$$

La seconde adaptation A_2 est déclenchée lorsqu'un comportement silencieux est détecté, afin de retourner dans la stratégie initiale de surveillance :

$$S_2 = \left\{\left(ER_1, C'_1, running\right), \left(P_1, C_2, pending\right), \left(P_2, C_3, running\right)\right\} \xrightarrow{A_2}$$
$$S_1 = \left\{\left(ER_1, C_1, running\right), \left(P_1, C_2, running\right)\right\}$$

La règle R_2 permet alors de repositionner le système dans sa configuration de surveillance normale (stratégie S_1) :

$$A_2 = \{\mathscr{U}\left(ER_1, burst\right), \mathscr{R}\left(P_1\right), \mathscr{D}\left(P_2\right)\}$$

Les événements déclencheurs et les actions d'adaptation ont été définis. Il est désormais possible d'opérer une traduction de ces éléments en règles compréhensibles par le *framework*.

3.3.2 Implémentation de l'exemple : la vision Ponder2

Les règles de pilotage de l'adaptation peuvent être traduites en politiques Ponder2. Cette section explique comment et en fait la démonstration sur l'exemple simple proposé.

3.3.2.1 Vers une intégration Ponder2

L'environnement d'utilisation de Ponder2 est le suivant : le *Ponder framework* lance un processus Ponder en tâche de fond. Outre son rôle de *listener* d'événements, ce processus doit aussi évaluer les politiques Ponder en fonction des événements qui lui parviennent (événements de type « messages » ou selon une période prédéfinie par un *timer*).

Par exemple, lorsque l'événement « réception de salves ! » est reçu par le *listener* Ponder, ce dernier doit évaluer la politique ECA correspondant à cet événement en particulier et appliquer les actions d'adaptation correspondant à la stratégie désirée (la suspension du *polling* existant, la création d'un nouveau *polling* et la modification du mode de détection de l'*event reporting*).

Cette vision se rapproche fortement de l'architecture PDP/PEP/PR décrite précédemment et composée des éléments suivants :

- le PR (*Policy Repository*) permet de stocker les politiques : ici, les politiques sont stockées dans un ensemble de fichiers regroupés dans un répertoire Ponder dédié à cet effet,
- le PDP (*Policy Decision Point*) permet d'évaluer les politiques applicables et de décider les actions à réaliser en fonction de celles-ci : ici, le rôle de PDP est joué par le processus Ponder tournant en tâche de fond,

- le PEP (*Policy Enforcement Point*) est quant à lui chargé d'appliquer les décisions prises par le PDP : l'exécution des politiques est réalisée par le processus Ponder qui lance les opérateurs d'adaptation correspondant à la politique évaluée.

Une quatrième entité apparaît dans le cadre de l'intégration du langage Ponder : le PAP (*Policy Administration Point*). Cette entité, visible dans la figure 3.8, permet de créer à tout moment des politiques de gestion et de les stocker dans le *Policy Repository* de Ponder. Ponder2 supporte en effet l'ajout dynamique de politiques.

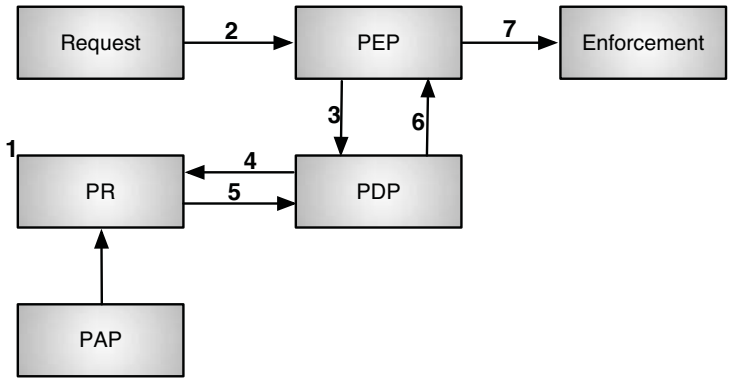

FIGURE 3.8: Architecture PDP/PEP/PAP de Ponder

3.3.2.2 Implémentation

Lorsque l'API Java Ponder2 est utilisée, une politique Ponder2 se trouve liée à une classe Java particulière implémentant le type défini `ManagedObject` du *Ponder framework*. Cette implémentation d'objet géré permet de créer naturellement un lien entre la couche « gouvernabilité » (tout l'environnement Ponder2 et les politiques) et la couche « adaptabilité » (les opérateurs d'adaptation de la configuration et de la portée des mécanismes de surveillance) du prototype qui a été développé intégralement en Java : les méthodes de la classe Java liée à la politique n'ont qu'à appeler les bons opérateurs d'adaptation.

Pour exemple, la figure 3.9 montre les politiques Ponder2 correspondant aux règles précédemment établies (cadre de gauche), appelant les actions `A1` ou `A2` de la classe Java liée (cadre de droite), appelant elle-même les opérateurs d'adaptation de la couche « adaptabilité ».

Les politiques ECA sont en relation directe avec les règles d'adaptation A_1 et A_2 précédemment établies :

- Sur détection d'une salve (**événement** `evBurst`), la politique `R1` est évaluée : celle-ci indique en partie **actions** d'exécuter le code Java de la méthode `A1` de la classe liée. Cette méthode lance alors la modification du mode de détection du *listener* de notifications `ER1`, la suspension du *polling* en cours d'exécution `P1` et la création d'un nouveau *polling* `P2` chargé de diagnostiquer la panne.
- Sur détection d'un silence (**événement** `evSilence`), la politique `R2` est évaluée : celle-ci indique en partie **actions** d'exécuter le code Java de la méthode `A2` de la classe liée pour revenir en mode normal. Cette méthode lance donc la modification du mode de détection du *listener* de notifications `ER1`, la reprise du *polling* `P1` et la suppression du *polling* `P2`.

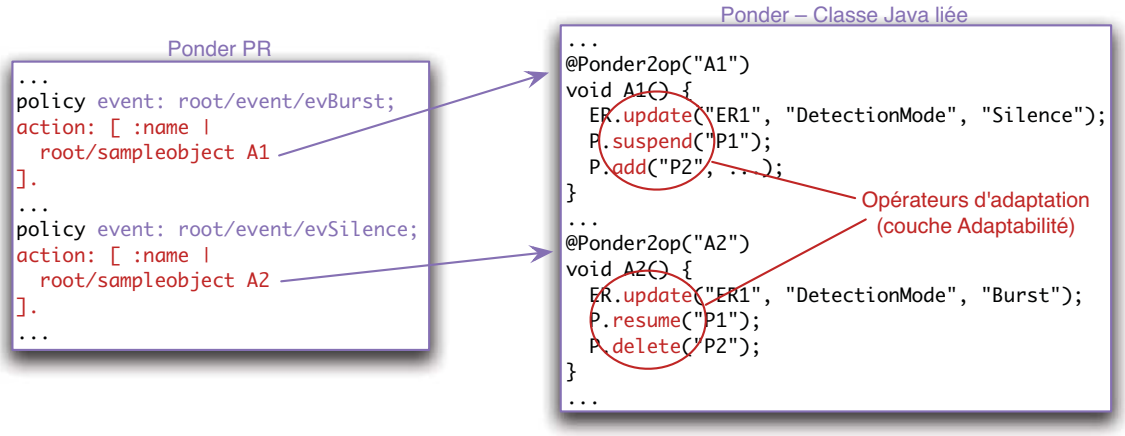

FIGURE 3.9: Politiques Ponder2 et code Java associé

3.3.2.3 Exemple de fonctionnement

Cet exemple ayant été implémenté, il est possible de montrer son fonctionnement général.

Un *listener* de notifications `ER1` est lancé en mode détection de salves, et des notifications sont envoyées aléatoirement. La figure 3.10 montre que le *listener* détecte ce comportement sporadique et génère l'événement `evBurst`.

Le *Ponder framework* (à gauche) prend la relève dès qu'il capture l'événement. Il applique dynamiquement les actions de reconfiguration qui s'imposent : le *polling* `P1` est suspendu, le *polling* `P2` est lancé et le *listener* d'événements commute en mode de détection de silences.

Ces actions correspondent à l'adaptation `A1`. L'adaptation `A2` fonctionne sur le même principe, et est aussi représentée sur la figure 3.10.

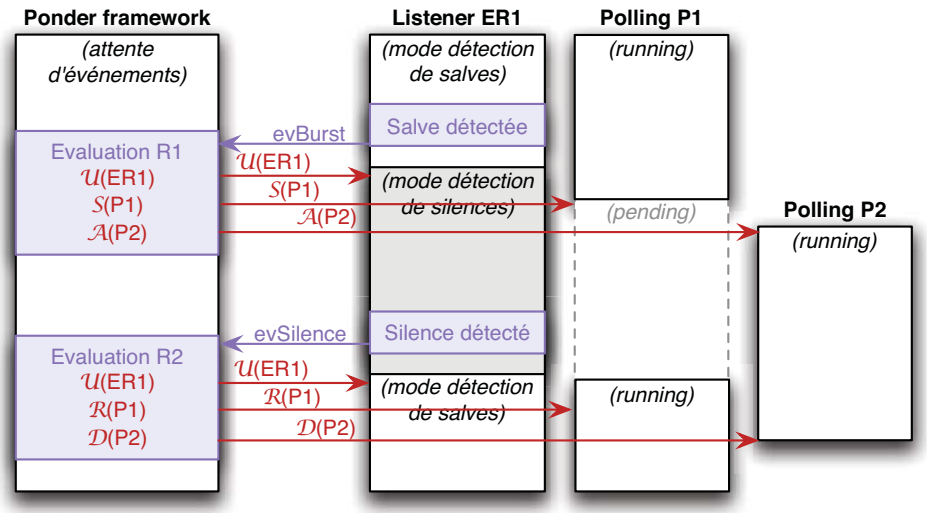

FIGURE 3.10: Fonctionnement du pilotage de l'adaptation à l'aide de politiques Ponder2

Cet exemple démontre bien la faisabilité de l'approche et la possibilité d'intégrer le langage de politiques Ponder2 au sein du *framework* de surveillance adaptative. Il est nécessaire maintenant

d'étudier le fonctionnement du *framework* au sein d'un cas d'étude concret, cohérent et réaliste.

3.4 Conclusion

La « gouvernabilité » est entièrement située dans le plan de contrôle du service de surveillance adaptative, comme le rappelle la figure 3.11.

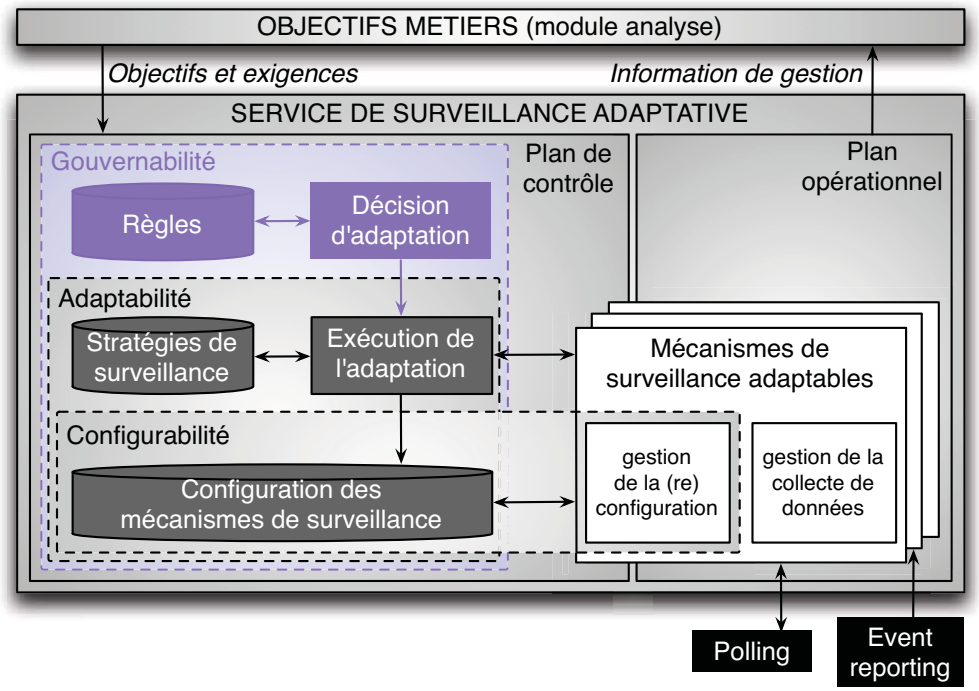

FIGURE 3.11: Le module de « gouvernabilité »

Elle se définit comme suit :

Définition

La **gouvernabilité** est la capacité de l'infrastructure à pouvoir détecter un besoin d'adaptation de la surveillance et à lancer opérationnellement cette adaptation.

Ainsi, la « gouvernabilité » permet de déterminer si et quand l'activité de surveillance doit être ajustée. Elle permet, en fonction de la détection d'événements représentatifs d'un état ou d'un comportement particulier de la surveillance elle-même, du système géré ou de variations de besoins ou de contraintes, de lancer des stratégies d'adaptation de la surveillance, composées d'un ensemble d'actions d'adaptation. Celles-ci sont matérialisées par les opérateurs d'adaptation de la capacité d'« adaptabilité » et qui agissent sur la configuration et/ou la portée des mécanismes de surveillance, définies au niveau « configurabilité ».

Comme démonstration de faisabilité et de cohérence de l'approche, de premiers exemples ont été implémentés à l'aide du paradigme de la gestion à base de politiques. Cette dernière s'est révélée un candidat pertinent, l'utilisation des règles permettant de déterminer les événements déclencheurs de l'adaptation, les conditions autorisant (ou non) l'adaptation, et les actions permettant de réaliser

effectivement l'adaptation de la surveillance. Le *Ponder framework* a notamment été choisi pour réaliser ce pilotage de l'adaptation à l'aide de politiques Ponder2, de sorte que le prototype proposé permet de valider la faisabilité de l'approche et apporte une solution concrète, réutilisable, modulaire et qui s'inscrit correctement dans un contexte de gestion intégrée.

Cette partie propose essentiellement un début de réflexion concernant de futurs travaux pouvant porter sur le pilotage de l'adaptation de la surveillance. Une des pistes importantes à suivre concerne notamment la détermination de l'intelligence nécessaire à introduire dans le prototype pour réaliser la correspondance entre les « variations opérationnelles, fonctionnelles ou besoins d'optimisation » et les stratégies d'adaptation qui doivent en résulter.

Cas d'étude général 4

L'objectif de ce cas d'étude est d'observer le comportement du service de surveillance adaptative lorsque les décisions de haut niveau proviennent de variations de contraintes. Ces dernières précisent un ensemble de bornes de valeurs qui sont des contraintes à respecter impérativement par le système de surveillance lorsque celui-ci doit s'adapter.

Les contraintes prises en exemple dans ce cas d'étude correspondent chacune à une des catégories de contraintes définies dans la première partie :

- une variation des objectifs de l'analyse : des besoins métiers ou objectifs de haut niveau peuvent être intégrés dynamiquement au module d'analyse, et la surveillance doit donc être ajustée pour toujours répondre aux objectifs de l'analyse, quels qu'ils soient,
- une variation des contraintes opérationnelles : une entité du système géré est en panne et il faut réduire la portée de la surveillance pour détecter l'origine de la panne, ou encore surveiller de nouvelles entités dans le système...
- un besoin de « *self-optimization* » : les données collectées par la surveillance répondent aux deux premières contraintes de variations. Mais la surveillance est tellement intrusive dans le système géré que celui-ci n'est pas optimal : la surveillance doit être optimisée pour être efficace mais sans perturber le bon fonctionnement du système sous-jacent.

Ainsi, nous pouvons montrer que quel que soit le type de contrainte, le *framework* peut s'adapter pour respecter chacune d'entre elles. Nous avons donc choisi de travailler sur la qualité de l'information (fraîcheur), la consommation et un objectif métier (une stratégie d'observation).

4.1 Description de l'état initial

Positionné dans un contexte de gestion intégrée, on suppose que, dans l'état initial, le système à surveiller est constitué de quatre équipements. Les contraintes suivantes sont positionnées :

- tous les équipements du système géré doivent être observés ;
- la fraîcheur des données : celle-ci ne doit pas dépasser 10 ms ;
- la consommation : celle-ci ne doit pas dépasser 10 requêtes toutes les 6 ms.

Ainsi, un *polling* a été positionné sur chaque équipement, chacun étant considéré comme une cible individuelle, de sorte à interroger périodiquement tous les équipements du système géré. La période choisie est par ailleurs de 6 ms, de sorte que la fraîcheur des données est de 6 ms < 10 ms et la consommation est de 4 requêtes toutes les 6 ms. Cet état est visible sur la figure 4.1.

Cet état est représenté formellement par la stratégie S_0 :

$$S_0 = \{(P_{D1}, \text{Cfg6}, \text{running}), (P_{D2}, \text{Cfg6}, \text{running}), (P_{D3}, \text{Cfg6}, \text{running}), (P_{D4}, \text{Cfg6}, \text{running})\}$$

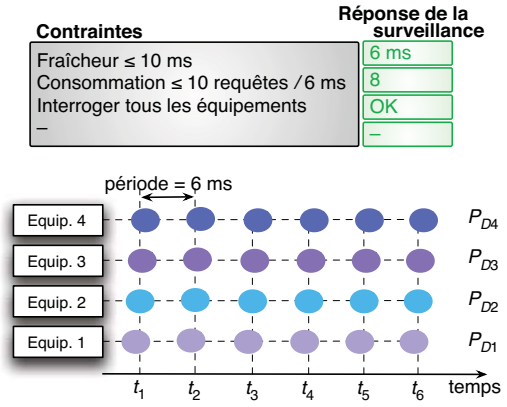

FIGURE 4.1: Etat initial de la surveillance

... avec :

- Dn avec $n = 1..4$ les équipements 1, 2, 3 et 4 ;
- Cfg6 une configuration de *polling* dont la période est de 6 ms.

Notons que sur la figure, un petit décalage temporel apparaît à chaque t_i, car toutes les opérations de *polling* ne peuvent avoir lieu exactement au même moment. Mais à chaque t_i, plus ou moins ce petit delta temporel que l'on considère comme négligeable, tous les *pollings* font leur demande de consultation sur l'équipement. Les informations récupérées sur les équipements sont nécessaires pour établir un état de la qualité de service.

4.2 Adaptation due à un ajout d'équipement

Un équipement est ajouté. La contrainte indiquant que tous les équipements doivent être interrogés n'est plus respectée. La surveillance doit s'adapter pour respecter toutes les contraintes :

- la fraîcheur des données ne doit pas dépasser 10 ms ;
- la consommation ne doit pas dépasser 10 requêtes toutes les 6 ms ;
- tous les équipements du système géré doivent être observés.

Pour résoudre ce problème, le système de surveillance adaptative ajoute un *polling* sur le cinquième équipement, comme le montre la figure 4.2. La contrainte est à nouveau respectée. Mais la consommation augmente à 10 requêtes toutes les 6 ms, ce qui correspond à la valeur maximum autorisée par la contrainte. Mais comme cette dernière est respectée, cette solution est retenue, car aucun événement indiquant un dépassement de seuil de la consommation n'a été émis.

Ce nouvel état est représenté formellement par la stratégie S_1 :

$$S_1 = \{(P_{D1}, \text{Cfg6}, \text{running}), (P_{D2}, \text{Cfg6}, \text{running}), (P_{D3}, \text{Cfg6}, \text{running}), (P_{D4}, \text{Cfg6}, \text{running}), (P_{D5}, \text{Cfg6}, \text{running})\}$$

... avec :

- Dn avec $n = 1..5$ les équipements 1, 2, 3, 4 et 5 ;
- Cfg6 une configuration de *polling* dont la période est de 6 ms.

Pour passer de la stratégie initiale S_0 à la stratégie courante S_1, l'adaptation suivante a été réalisée :

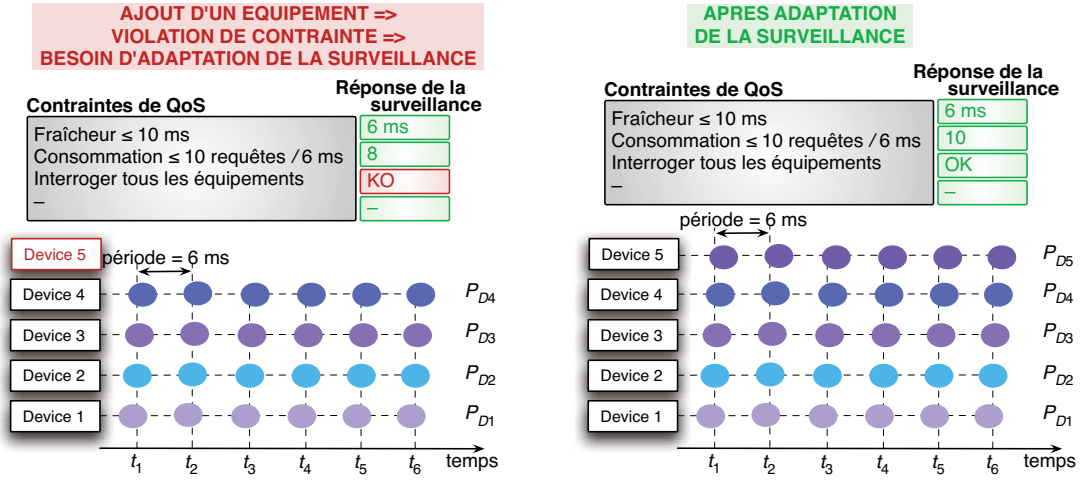

FIGURE 4.2: Adaptation de la surveillance adaptative due à l'ajout d'un équipement

$$S_0 \xrightarrow{A_1} S_1$$
avec
$$A_1 = \{\mathscr{A}\,(P_{D5}, \text{Cfg6}, \text{running})\}$$

En se basant sur les résultats du chapitre précédent, le temps d'exécution théorique de cette adaptation serait de 500 ms. Cette durée ne comprend pas la prise de décision de cette adaptation.

Cette adaptation est décidée par le niveau de gouvernabilité au moyen de la règle de la figure 4.3.

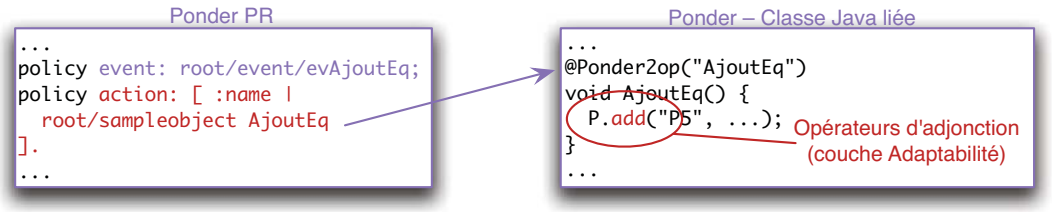

FIGURE 4.3: Politique Ponder correspondant à l'ajout d'un équipement

Lorsque cette étude de cas a été expérimentée (dans les mêmes conditions qu'au chapitre précédent, mais avec le serveur CIM contenant toutes les classes ayant permis de faire remonter nos premiers résultats), l'exécution opérationnelle de cette adaptation a été réalisée en 642 ms, ce qui est un peu supérieur au temps d'exécution théorique. Ceci s'explique par la présence d'un plus grand nombre d'objets dans le serveur CIM. En effet, plus il y a d'objets enregistrés au sein du *CIM Repository*, plus les temps d'exécution seront augmentés, du fait de l'utilisation des *paths* CIM (qui impliquent de parcourir l'ensemble des objets et de comparer chaque *path* de l'objet parcouru avec le *path* recherché). Il paraît logique que les temps d'exécution réels soient donc toujours supérieurs aux temps d'exécution théoriques en condition « meilleur cas ».

4.3 Adaptation due à la modification d'une contrainte

La consommation passe de 10 requêtes toutes les 6 ms à 8 toutes les 6 ms. Il s'agit d'une modification simulée par le changement d'une valeur conditionnant l'envoi d'un événement de dépassement de seuil pour la consommation. Le seuil est dépassé et la contrainte est violée. La surveillance doit s'adapter pour respecter les nouvelles contraintes :

- la fraîcheur des données ne doit pas dépasser 10 ms ;
- la consommation ne doit pas dépasser 8 requêtes toutes les 6 ms ;
- tous les équipements du système géré doivent être observés.

Deux cas vont être illustrés ici pour montrer que l'adaptation de la surveillance ne donne pas les mêmes effets en fonction de ce qui lui est demandé. Deux cas peuvent être mis en évidence :

- On demande au système de surveillance de se positionner en « *best effort* » au niveau de la rapidité d'adaptation (l'adaptation ne pourra pas être optimale) ;
- On demande au système de surveillance de se positionner en « *best effort* » au niveau de son optimisation (l'adaptation ne pourra pas être effectuée rapidement).

Le choix d'un cas ou de l'autre correspond également à une contrainte pour déterminer quelle option est la meilleure dans le contexte courant. Ceci doit être défini au niveau d'objectifs métiers.

Pour la suite, les deux cas vont être détaillés, pour montrer les effets d'une telle décision.

4.3.1 *Best effort* : Rapidité d'adaptation

Pour s'adapter rapidement, il est préférable de réaliser le moins d'actions d'adaptations possibles, et que ces actions soient réalisables rapidement. Ainsi, il est décidé d'augmenter la période de chaque *polling*, comme cela est visible sur la figure 4.4.

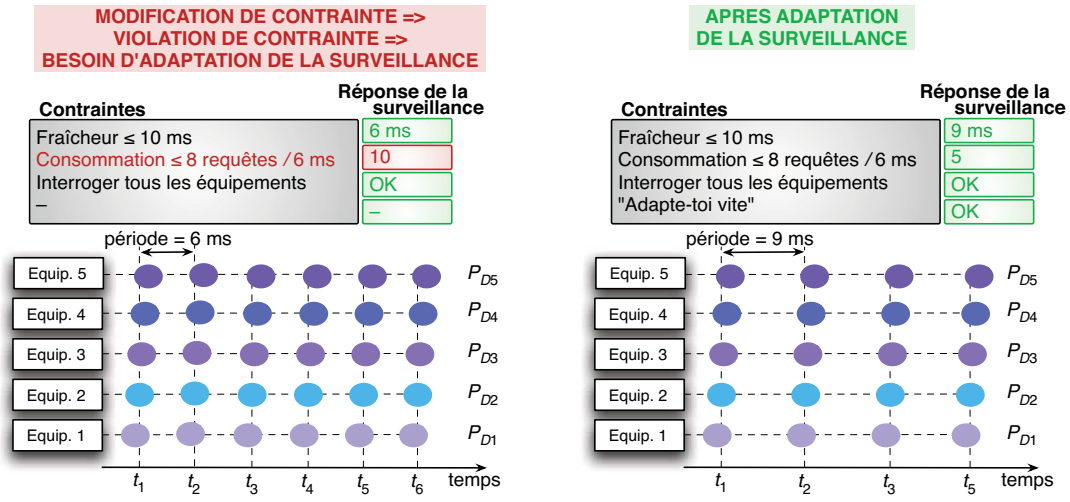

FIGURE 4.4: Adaptation rapide de la surveillance

La période de *polling* a été passée de 6 ms à 9 ms, ce qui permet de continuer à respecter la contrainte de fraîcheur (inférieure à 10 ms), tout en réduisant la consommation à 5 requêtes toutes les 6 ms. Quant à la durée d'adaptation, elle est théoriquement réalisée en $5 \times 149 = 745$ ms environ (mise à jour de chacun de cinq *pollings*). Cette solution est retenue, mais si le nombre d'équipements

à surveiller augmente encore, la surveillance devra peut-être être adaptée encore une fois.

Ce nouvel état de la surveillance est représenté formellement par la stratégie S_2 :

$$S_2 = \{(P_{D1}, \text{Cfg9}, \text{running}), (P_{D2}, \text{Cfg9}, \text{running}), (P_{D3}, \text{Cfg9}, \text{running}), (P_{D4}, \text{Cfg9}, \text{running}), (P_{D5}, \text{Cfg9}, \text{running})\}$$

... avec :

- Dn avec $n = 1..5$ les équipements 1, 2, 3, 4 et 5 ;
- Cfg9 une configuration de *polling* dont la période est de 9 ms.

Pour passer de la stratégie S_1 à la stratégie S_2, l'adaptation suivante a été réalisée :

$$S_1 \xrightarrow{A_2} S_2$$

avec

$$A_2 = \{\mathscr{U}(P_{D1}, \text{Cfg9}), \mathscr{U}(P_{D2}, \text{Cfg9}), \mathscr{U}(P_{D3}, \text{Cfg9}), \mathscr{U}(P_{D4}, \text{Cfg9}), \mathscr{U}(P_{D5}, \text{Cfg9})\}$$

L'adaptation « *best effort* » au niveau de la rapidité est possible : le temps d'adaptation est de moins d'une seconde pour ces cinq équipements et la solution proposée fonctionne même si elle n'est pas optimale.

Cette adaptation est décidée par le niveau de gouvernabilité au moyen de la règle de la figure 4.5.

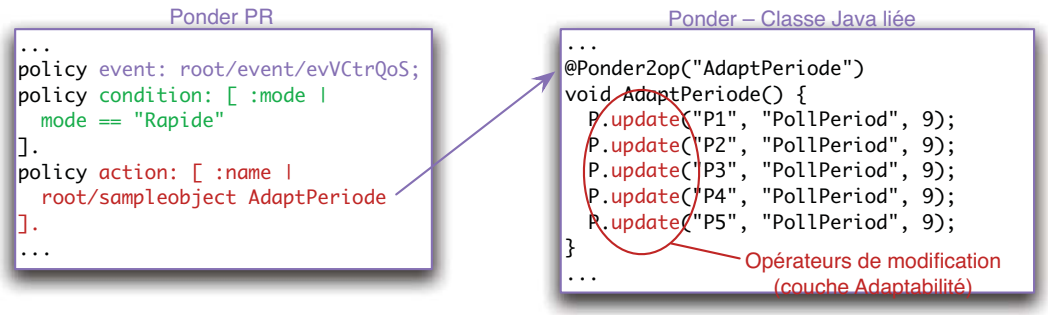

FIGURE 4.5: Politique Ponder d'adaptation en mode « adaptation rapide »

Dans les conditions expérimentales de cette étude de cas, l'exécution opérationnelle de cette adaptation a été réalisée en 965 ms, ce qui est encore une fois supérieur au temps d'exécution théorique.

4.3.2 *Best effort* : optimisation

Plutôt que d'avoir cinq *pollings* (un pour chaque équipement), le système de surveillance adaptative décide de n'en avoir qu'un seul qui interrogera successivement chaque équipement, car cette configuration est plus performante théoriquement en terme de charge sur le *manager* (du fait de la réduction du nombre de mécanismes en cours d'exécution). Toutefois, cette configuration est plus longue à mettre en place. Lorsque l'adaptation n'a pas besoin d'être réalisée au plus vite, la surveillance peut s'optimiser dans ce sens.

Pour ce faire, le système de surveillance passe d'un mode de collecte individuel (un *polling* par équipement) à un mode de collecte collectif (un *polling* interrogeant tous les équipements successivement). Ceci permet dans notre cas de réduire à 6 le nombre des requêtes. Mais pour garder

la même fraîcheur (5 ms), il est impératif de diviser la période à 1 ms, comme cela est visible sur la figure 4.6.

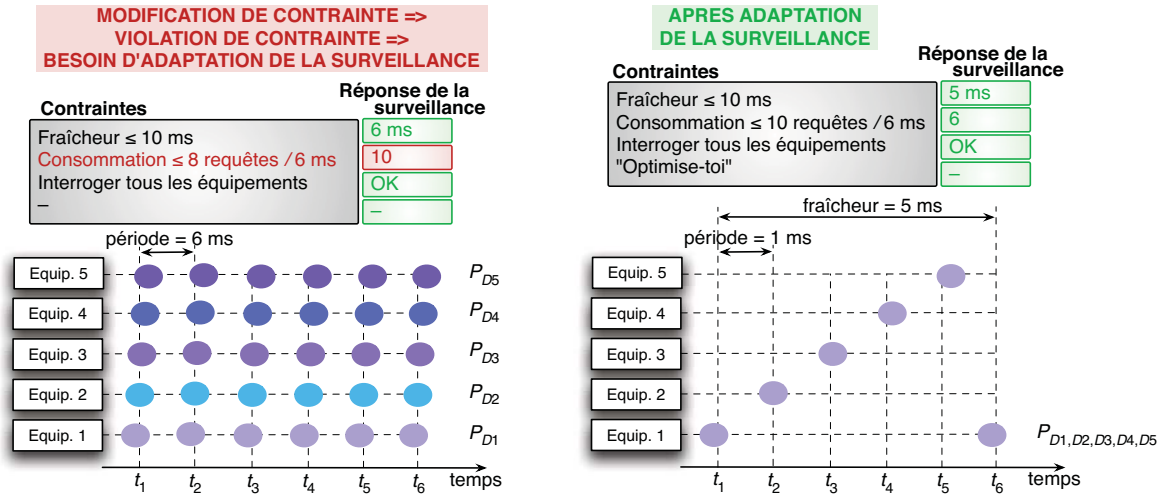

FIGURE 4.6: Adaptation optimisée de la surveillance

Cet nouvel état de la surveillance est représenté formellement par la stratégie S_3 :

$$S_2 = \{(P_{D1}, \text{Cfg6}, \text{pending}), (P_{D2}, \text{Cfg6}, \text{pending}), (P_{D3}, \text{Cfg6}, \text{pending}), (P_{D4}, \text{Cfg6}, \text{pending}), (P_{D5}, \text{Cfg6}, \text{pending}), (P_{D1,D2,D3,D4,D5}, \text{Cfg1}, \text{running})\}$$

... avec :

- Dn avec $n = 1..5$ les équipements 1, 2, 3, 4 et 5 ;
- Cfg6 une configuration de *polling* dont la période est de 6 ms ;
- Cfg1 une configuration de *polling* dont la période est de 1 ms.

Pour passer de la stratégie S_1 à la stratégie S_3, l'adaptation suivante a été réalisée :

$$S_1 \xrightarrow{A_3} S_3$$

avec

$$A_3 = \{\mathscr{S}(P_{D1}), \mathscr{S}(P_{D2}), \mathscr{S}(P_{D3}), \mathscr{S}(P_{D4}), \mathscr{S}(P_{D5}), \mathscr{A}(P_{D1,D2,D3,D4,D5}, \text{Cfg1}, \text{running})\}$$

L'adaptation de la surveillance est plus complexe dans ce cas-ci dans la mesure où elle demande de réaliser un plus grand nombre d'actions d'adaptation, dont certaines ont une durée d'exécution élevée. Cette adaptation consiste en la suspension de 5 pollings (545 ms) et en l'ajout d'un nouveau polling ayant 6 cibles collectives (1168 ms). L'adaptation est réalisée en 1713 ms (soit presque deux secondes).

Cette adaptation est décidée par le niveau de gouvernabilité au moyen de la règle de la figure 4.7. L'événement déclencheur de l'adaptation est bien exactement le même que pour le mode « adaptation rapide ». Toutefois, c'est la condition qui va influencer la prise de décision et impliquer un type d'adaptation optimisé.

Dans les conditions expérimentales de cette étude de cas, l'exécution opérationnelle de cette adaptation a été réalisée en 2058 ms, ce qui est supérieur une fois encore au temps d'exécution théorique.

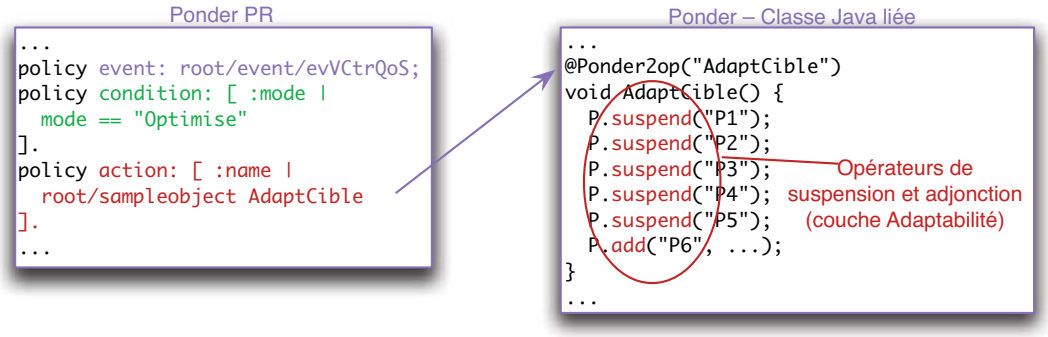

FIGURE 4.7: Politique Ponder d'adaptation en mode « adaptation optimisée »

4.4 Bilan

Selon la décision, les résultats au niveau de l'exécution des mécanismes de surveillance ne sont pas les mêmes. Des règles doivent être rédigées de manière précise et cohérente de sorte à pouvoir déterminer quelles sont les actions à entreprendre en fonction de chaque cas particulier. Dans tous les cas, la surveillance offre les moyens de s'adapter en fonction de chacun des objectifs déterminés. Reste à intégrer l'ensemble de l'intelligence permettant de faire le *mapping* entre les objectifs de haut niveau (ici issus de la décision provenant des variations de contraintes), et les opérations d'adaptation des mécanisme de surveillance.

De plus, cette étude montre qu'il faut délimiter le domaine d'intervention des différents systèmes impliqués : ici, ce sont les variations de contraintes qui impliquent de réduire le nombre de requêtes envoyées par les mécanismes de surveillance, mais il pourrait aussi être envisagé que ce soit le système d'auto-optimisation de la surveillance qui puisse lui-même intégrer une certaine quantité d'intelligence pour déterminer si elle doit s'optimiser ou s'adapter pour répondre aux besoins avant même qu'ils ne soient énoncés.

Quatrième partie

Conclusion générale

Conclusion générale

La gestion intégrée permet de gérer des systèmes complexes comme une seule et même entité. Peu importent l'hétérogénéité des composants et des protocoles, la taille du système, le nombre de composants... Le système géré, dans son ensemble, est considéré comme une boîte noire qu'il faut surveiller, contrôler, adapter si besoin. La surveillance intégrée a un rôle prédominant dans la gestion de tels systèmes car elle permet de faire remonter toutes les informations significatives d'une panne, d'un dysfonctionnement sur le système. Cette surveillance devait être automatisée de sorte à devenir « adaptative ». Mais comment ?

La surveillance s'opère au moyen de deux mécanismes, le *polling* et l'*event reporting*, qui doivent être rendus configurables. De ce fait, la surveillance peut gouverner son propre comportement opérationnel et s'adapter à tout type de contraintes : internes (en provenance de la connaissance interne de la surveillance, par exemple pour des besoins d'optimisation), ou externes (en provenance du système géré dont les objectifs de haut niveau ont pu être modifiés). Il est donc nécessaire de définir un *framework* dont les capacités permettent d'opérer des changements sur les paramètres de configuration (fréquence d'interrogation...) ou la portée (ajout ou suppression de cibles interrogeables) des mécanismes de surveillance en fonction des besoins apparaissant en temps réel.

Le *framework* modélisé, conceptualisé et implémenté présente une interface clairement définie grâce à une conception générique pouvant être spécialisée, en fonction de tout besoin, sur n'importe quel type d'environnements technologiques (toutefois pour démontrer la faisabilité de l'approche dans sa globalité, l'environnement CIM/WBEM a été choisi pour donner un exemple d'implémentation du prototype, lui-même développé en Java). Le *framework* est ainsi constitué de trois capacités interconnectées permettant de rendre la surveillance adaptative.

Tout d'abord, la **Configurabilité** est la capacité d'initialiser et de modifier dynamiquement la portée et les valeurs des paramètres gouvernant le comportement des mécanismes de surveillance. Les informations de surveillance sont ici stockées dans le serveur CIM de l'outil Open Pegasus, et peuvent être récupérées par les mécanismes pour gouverner le comportement de leur exécution opérationnelle. Cette séparation entre configuration et portée des mécanismes d'une part et l'exécution opérationnelle d'autre part rend possible la reconfiguration et l'ajustement du comportement global de la surveillance. Encore faut-il apporter l'interface permettant de modifier ces valeurs de paramètres et de portée. C'est à ce niveau que l'adaptabilité (deuxième capacité) entre en jeu.

L'**Adaptabilité** est la capacité de modifier dynamiquement et sans interruption le comportement d'une activité de surveillance. Cette couche présente ainsi un ensemble d'opérateurs basiques d'adaptation ($\mathscr{AUDRS}$) – développés en Java grâce à l'API SBLIM (de sorte à créer un client CIM) – permettant certes de modifier les valeurs des paramètres de configuration et la portée des mécanismes définis au niveau configurabilité, mais plus généralement d'agir sur le cycle de vie de ces mécanismes en permettant aussi leur création, suspension, reprise ou suppression. Elle offre

ainsi une interface support à la programmation de modifications du comportement d'une activité de surveillance opérée par un ensemble de mécanismes de base. Dit autrement, elle permet la réalisation effective des actions retenues pour opérer l'adaptation d'une stratégie de surveillance en concrétisant le changement de comportement global de l'activité de surveillance concernée. Reste désormais à automatiser la prise de décision d'adaptation en fonction des variations des contraintes et des besoins d'optimisation.

La **Gouvernabilité**, enfin, est la capacité de l'infrastructure à pouvoir détecter un besoin d'adaptation de la surveillance et à lancer opérationnellement cette adaptation. Cette dernière capacité offre les moyens de déterminer si et comment temporellement l'activité de surveillance doit être ajustée en lançant des stratégies d'adaptation de la surveillance, composées d'un ensemble d'actions d'adaptation, matérialisées par les opérateurs d'adaptation de la couche d'adaptabilité. Pour déterminer un besoin d'adaptation, il faut prendre en compte tous les événements potentiellement déclencheurs d'une adaptation de la surveillance. Ainsi, le paradigme de la gestion à base de politiques a été étudié et choisi car il présente des politiques de types ECA : ces règles permettent de définir les événements déclencheurs de l'adaptation, les conditions autorisant (ou non) l'adaptation, et les actions permettant de réaliser effectivement l'adaptation de la surveillance. Le *Ponder framework* a ainsi été choisi pour piloter l'adaptation de la surveillance à l'aide de politiques Ponder2, de sorte que le prototype proposé permet de valider la faisabilité de l'approche et apporte une solution concrète, réutilisable, modulaire et qui s'inscrit correctement dans notre contexte de gestion intégrée.

En conclusion, la solution proposée répond à la problématique. En effet, bien qu'elle ait été implémentée dans un environnement technologique spécifique (CIM/WBEM) à des fins de démonstration de faisabilité et d'évaluation de performances, il suffit de reprendre la conception, les modèles et la formalisation pour réimplémenter la solution au sein d'un autre environnement et avec l'utilisation de protocoles différents. La solution est donc générique (tout au moins sa conception). Le système géré reste quant à lui une boîte noire : la solution peut être positionnée dans un contexte de gestion intégrée. De plus, l'interface de la solution a été définie et permet la réutilisation pour spécialisation ou non, en partie ou en totalité (les modules de conception permettent en effet de ne considérer qu'une seule, deux ou les trois capacités du framework (configurabilité, adaptabilité, gouvernabilité).

Du point de vue « apport » de la solution, elle permet une adaptation dynamique de la surveillance en cours d'exécution et sans interruption du système. Les décisions d'adaptation sont, elles, exprimées par des règles et ces dernières peuvent être intégrées au framework à tout moment.

L'ensemble des propriétés d'une solution de surveillance adaptative définie en partie II est donc couvert par le *framework* proposé. Toutefois, un ensemble de perspectives est à mettre en évidence. Le *framework* doit être testé dans un environnement plus contraint (réel) afin d'être éprouvé et de déterminer ses limites (échelle, implémentation au sein d'autres environnements, intégration avec d'autres protocoles, composants, technologies). De plus le niveau gouvernabilité doit être enrichi. Au-delà de ces aspects, une autre perspective intéressante serait de consolider l'observabilité d'une activité de surveillance. En effet, selon une vision autonomique et dans un objectif d'obtenir une activité de surveillance capable de s'auto-optimiser, il est nécessaire d'intégrer une connaissance nécessaire à la mise en place de cette propriété.

Dans l'état actuel du *framework*, nous avons, dans les modèles d'information de gestion qui ont été définis en CIM pour la configurabilité, spécialisé le *pattern* CIM de Statistiques pour les opérations de *polling*. Il s'agit là d'une brique informationnelle de base pouvant être utilisée pour la construction d'une base de connaissances supportant des stratégies d'auto-observation, d'auto-évaluation et d'auto-optimisation.

Enfin, en poursuivant l'approche *bottom-up* suivie pour la conception de ce cadriciel, de nouveaux niveaux d'abstraction peuvent être proposés pour faciliter l'expression de règles, d'objectifs, de contraintes et de besoins métiers. Il s'agit donc d'enrichir et d'améliorer l'outillage du niveau gouvernabilité. La définition et l'implémentation d'un DSL (*Domain Specific Language*) spécifique à la programmation de solutions de surveillance adaptative de réseaux et de systèmes complexes paraît être une piste prometteuse. Un tel langage pourra apporter un niveau d'abstraction supplémentaire au niveau conceptuel, mais s'appuyer pour l'implémentation de ses instructions sur le cadriciel proposé dans cette thèse.

Publications

Conférences et *workshops* internationaux

- Audrey Moui, Thierry Desprats, Emmanuel Lavinal, Michelle Sibilla, "**Managing polling adaptability in a CIM/WBEM infrastructure**" (regular paper), in *International DMTF Academic Alliance Workshop on Systems and Virtualization Management : Standards and New Technologies (SVM 2010)*, Niagara Falls (CA), 29/10/2010–29/10/2010, IEEExplore digital library, pp.1–6, décembre 2010.
- Audrey Moui, Thierry Desprats, "**Towards Self-Adaptive Monitoring Framework for Integrated Management**" (short paper), in *IFIP International Conference on Autonomous Infrastructure, Management, and Security (AIMS 2011)*, Nancy, France, 13/06/2011-17/06/2011, Isabelle Chrisment, Alva Couch, Rémi Badonel, Martin Waldburger (Eds.), Springer, pp.160–163, 2011.
- Audrey Moui, Thierry Desprats, Emmanuel Lavinal, Michelle Sibilla, "**Information Models for Managing Monitoring Adaptation Enforcement**" (regular paper), in *International Conference on Adaptive and Self-Adaptive Systems and Applications (ADAPTIVE 2012)*, Nice (France), 22–27 Juillet 2012.
- Audrey Moui, Thierry Desprats, Emmanuel Lavinal, Michelle Sibilla, "**A CIM-Based Framework to Manage Monitoring Adaptability**" (short paper), in *International Conference on Network and Service Management (CNSM 2012)*, Las Vegas (USA), 22–26 Octobre 2012.

Rapports

- Audrey Moui, Thierry Desprats, Michelle Sibilla, « **Adaptation de la surveillance : besoins métier, état de l'art et perspectives** » (Rapport de contrat), *IMAP/ADCN 2G-WP1.2.3-1*, IRIT, juillet 2010.
- Audrey Moui, Thierry Desprats, Michelle Sibilla, « **Gouvernance et adaptabilité de la surveillance** » (Rapport de contrat), *IMAP/ADCN 2G*, IRIT, juillet 2011.
- Audrey Moui, Thierry Desprats, Michelle Sibilla, « **Adaptation d'une stratégie de surveillance : tests et évaluation d'un démonstrateur** » (Rapport de contrat), *IMAP/ADCN 2G*, IRIT, juillet 2012.

Bibliographie

[Abid09] Z. Abid, S. Chabridon, D. Conan, "**A Framework for Quality of Context Management**", in *QuaCom*, 120–131, 2009.

[Agarwala06] Sandip Agarwala, Yuan Chen, Dejan Milojicic and Karsten Schwan, "**QMON : QoS- and Utility-Aware Monitoring in Enterprise Systems**", in *ICAC'06 : Proceedings of the 2006 IEEE International Conference on Autonomic Computing*, pp.124–133, June 2006.

[Agrawal05a] Dakshi Agrawal, James Giles, Kang-Won Lee and Jorge Lobo, "**Autonomic Computing Expression Language**", *IBM developerWorks*, February 2005.

[Agrawal05b] Dakshi Agrawal, Kang-Won Lee and Jorge Lobo, "**Policy-Based Management of Networked Computing Systems**", in *IEEE Communications Magazine*, 43(10), October 2005.

[Agrawal07] Dakshi Agrawal, Seraphin B. Calo, Kang-Won Lee and Jorge Lobo, "**Issues in Designing a Policy Language for Distributed Management of IT Infrastructures**", in *Integrated Network Management*, pp.30–39, 2007.

[Apache] Apache Imperius Incubator, "**SPL Language Reference**", [en ligne : `http://incubator.apache.org/imperius/docs/spl_reference.html`], page consultée en Décembre 2010.

[Binzenhofer06] Andreas Binzenhöfer, Kurt Tutschku, Björn auf dem Graben, Markus Fiedler and Patrick Thiran, "**A P2P-Based Framework for Distributed Network Management**", in *EuroNGI*, pp.198–210, March 2006.

[Boutaba07] Raouf Boutaba and Issam Aib, "**Policy-based Management : A Historical Perspective**", *Journal of Network and Systems Management*, 15(4), pp.447–480, 2007.

[Brownlee99] N. Brownlee, "**SRL : A Language for Describing Traffic Flows and Specifying Actions for Flow Groups**", *IETF Internet draft*, August 1999.

[Cantieni06] Gion Reto Cantieni, Gianluca Iannaccone, Chadi Barakat, Christophe Diot and Patrick Thiran, "**Reformulating the Monitor Placement Problem : Optimal Network-Wide Sampling**", in *CoNXT'06 : Proceedings of the 2006 ACM CoNEXT Conference*, pp.1–12, March 2006.

[Chaparadza06] Ranganai Chaparadza, "**Introducing On-Demand MIBs to Traffic Engineering**", in *14th IEEE International Conference on Networks ICON'06*, 1, pp.1–6, September 2006.

[Chilukuri06] Sampath Chilukuri, "**Create Autonomic Computing Policies using Simplified Policy Language : Streamline Policy Creation with Policy Management for Autonomic Computing**", *IBM developerWorks*, February 2006.

[Chomicki00] Jan Chomicki, Jorge Lobo and Shamim Naqvi, "**A Logic Programming Approach to Conflict Resolution in Policy Management**", in. *7th Conf. on Principles of Knowledge Representation and Reasoning KR2000*, pp.121–132, 2000.

[Chung04] L. Chung and N. Subramanian, "**Adaptable Architecture Generation for Embedded Systems**", *Journal of Systems and Software*, 71(3), (2004), pp. 271–295.

[CIM12] DMTF, "**Common Information Model (CIM) Infrastructure**" specification v.2.6.0 [online], (2010) [retrieved : May, 2012], http ://www.dmtf.org/standards/cim.

[Clark89] David D. Clark, "**Policy Routing in Internet Protocols**", *IETF Network Working Group, RFC 1102*, May 1989.

[Damianou01] Nicodemos Damianou, Naranker Dulay, Emil Lupu and Morris Sloman, "**The Ponder Policy Specification Language**", in *Proceedings of Policy'01 Workshop on Policies for Distributed Systems and Networks*, pp.17–28, Jan. 2001.

[Derbel09] Hajer Derbel, Nazim Agoulmine and Mikaël Salaün, "**ANEMA : Autonomic Network Management Architecture to Support Self-Configuration and Self-Optimization in IP Networks**", *Computer Networks*, 53, pp.418–430, 2009.

[Dilman01] Mark Dilman and Danny Raz, "**Efficient Reactive Monitoring**", in *INFOCOM 20th Annual Joint Conference of the IEEE Computer and Communications Society*, 2, pp.1012–1019, April 2001.

[DMTF03b] DMTF, "**CIM Policy Model**", *White Paper CIM version 2.7*, version 2.7.0, 18 June 2003.

[DMTF09] DMTF, "**CIM Simplified Policy Language (CIM-SPL)**", *DMTF Specification DSP0231*, version 1.0.0, 14 July 2009.

[Duarte08] E.P. Duarte Jr, M.A. Musicante, H.H. Fernandes, "**ANEMONA : a Programming Language for Network Monitoring Applications**", *International Journal of Network Management*, 18(4), August 2008.

[Durai11] B. Durai, T. A. Gonsalves and K. M. Sivalingam, "**Adaptive Push-Based Data Collection Method for Online Performance Monitoring**", *2011 National Conference on Communications (NCC)*, pp.1–5, January 2011.

[Godik03] S. Godik, T. Moses, "**eXtensible Access Control Markup Language (XACML) version 1.0**", *OASIS, XACML Technical Committee*, 18 February 2003.

[Han04] Qi Han, Sebastian Gutierrez-Nolasco and Nalini Venkatasubramanian, "**Reflective Middleware for Integrating Network Monitoring with Adaptive Object Messaging**", *IEEE Network*, 18(1), pp.56–65, January-February 2004.

[Hernandez01] Edwin A. Hernandez, Matthew C. Chidester and Alan D. George, "**Adaptive Sampling for Network Management Applications**", *Journal of Network and Systems Management*, 9(4), pp.409–434, 2001.

[IBM03] IBM, "**An Architectural Blueprint for Autonomic Computing**". *Technical report*, IBM, 2003.

[ISO10040] ISO/IEC 10040 :1998, "**Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Aperçu général de la gestion-systèmes**", Geneva, Switzerland, [en ligne : `http://www.iso.org/iso/fr/home/store/catalogue_tc/catalogue_detail.htm?csnumber=24406`], page consultée en Décembre 2012.

[ISO10164] ISO/IEC 10164 :1993, "**Information technology : open systems interconnection**" [en ligne : `http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=28566`], page consultée en Avril 2013.

[Jiao00] Jia Jiao, Shamim Naqvi, Danny Raz and Binay Sugla, "**Toward Efficient Monitoring**", *IEEE Journal on Selected Areas in Communications*", 18(5), pp.723–732, May 2000.

[Kephart03] Jeffrey O. Kephart and David M. Chess, "**The Vision of Autonomic Computing**", *Computer*, 36(1), pp.41–50, January 2003.

[Kephart07] Jeffrey O. Kephart and Rajarshi Das, "**Achieving Self-Management via Utility Functions**", *IEEE Internet Computing*, 11(1), pp.40–48, January-February 2007.

[Laborde08] Romain Laborde et Thierry Desprats, « **Gestion de conditions stables dans XACML : intérêt d'une approche par notification** », *Gestion de REseaux et de Services (GRES)*, pp.161–168, Décembre 2008.

[Lahmadi08] Abdelkader Lahmadi, « **Performances des fonctions et architectures de supervision de réseaux et de services** », *PhD Thesis*, Université de Nancy II, Janvier 2008.

[LeDuc10] B. Le Duc, P. Collet, J. Malenfant, N. Rivierre, "**A QoI-Aware Framework for Adaptive Monitoring**", *2nd International Conference on Adaptive and Self-Adaptive Systems and Applications (ADAPTIVE 2010)*, 133–141, November 2010.

[Lobo99] Jorge Lobo, Randeep Bhatia and Shamim Naqvi, "**A Policy Description Language**", in *AAAI'99/IAAI'99 : Proceedings of the sixteenth national conference on Artificial Intelligence and the eleventh Innovative Applications of Artificial Intelligence conference*, pp.291–298, 1999.

[Lymberopoulos04] L. Lymberopoulos, E. Lupu and M. Sloman, "**Ponder Policy Implementation and Validation in a CIM and Differentiated Services Framework**", *IFIP/IEEE Network Operations and Management Symposium NOMS 2004*, April 2004.

[Massie04] Matthew L. Massie, Brent N. Chun and David E. Culler, "**The Ganglia Distributed Monitoring System : Design, Implementation, and Experience**", *Parallel Computing*, 30(7), pp.817–840, 2004.

[Miao07] Kai X. Miao, Henning Schulzrinne, Vishal Kumar Singh and Qianni Deng, "**Distributed Self Fault-Diagnosis for SIP Multimedia Applications**", *MMNS'07 : Proceedings of the 10th IFIP/IEEE International Conference on Management of Multimedia and Mobile Networks and Services*, 4787, pp.187–190, 2007.

[Moffett91] J. D. Moffett and M.S. Sloman, "**The Representation of Policies as System Objects**", in *Proceedings of the Conference on the Organizational Computer Systems (COCS), ACM SIGOIS Bulletin*, 12(2–3), pp. 171–184, Nov. 1991.

[Moghe98] Pratyush Moghé and Michael H. Evangelista, "**RAP - Rate Adaptive Polling for Network Management Applications**", in *Proceedings of IEEE Network Operation and Management Symposium NOMS'98*, pp.395–399, 1998.

[Nagios] Nagios, *The Industry Standard in IT Infrastructure Monitoring*, [en ligne : `http://www.nagios.com/`], page consultée en Février 2011.

[Nossik98] M. Nossik, F. Welfeld and M. Richardson, "**PAX-PDL : a Non-Procedural Packet Description Language**", *Technical report*, Sept. 30, 1998.

[OpenPegasus] The Open Group, "**Open Pegasus : C++ CIM/WBEM Manageability Services Broker**", [en ligne : `https://collaboration.opengroup.org/pegasus/`], page consultée en Décembre 2012.

[Ouda10] A. Ouda, H. Lutfiyya, M. Bauer, "**Automatic Policy Mapping to Management System Configurations**", in *2010 IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY)*, 87–94, 2010.

[Park06] Jong-Geun Park et al., "**A method for representing and transporting CIM operations using binary XML in the WBEM architecture**", *8th International Conference of Advanced Communication Technology*, 20–22, 2006.

[Prieto07] Alberto Gonzalez Prieto and Rolf Stadler, "**A-GAP : an Adaptive Protocol for Continuous Network Monitoring with Accuracy Objectives**", *IEEE Transactions on Network and Service Management (IEEE TNSM)*, 4(1), pp.2–12, June 2007.

[Prieto09] A.G. Prieto, R. Stadler, "**Controlling Performance Trade-offs in Adaptive Network Monitoring**", *Proceedings of IM 2009*, 359–366, 2009.

[Roxburgh11] D. Roxburgh, D. Spaven, C. Gallen, "**Monitoring as an SLA-Oriented Consumable Service for SaaS Assurance : a Prototype**", *2011 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 925–939, 2011.

[Samaan09] Nancy Samaan and Ahmed Karmouch, "**Towards Autonomic Network Management : an Analysis of Current and Future Research Directions**", *Communications Surveys and Tutorials, IEEE*, 11(3), pp.22–36, August 2009.

[Sloman94] Morris Sloman, "**Policy Driven Management for Distributed Systems**", *Journal of Network and Systems Management*, 2(4), pp.333–360, 1994.

[Sterritt05] Roy Sterritt, David W. Bustard, Darren Gunning and Phillip Henning, "**Autonomic Communications and the Reflex Unified Fault Management Architecture**", *Advanced Engineering Informatics*, 19(3), pp.189–198, 2005.

[Stone01] Gary N. Stone, Bert Lundy and Geoffrey G. Xie, "**Network Policy Languages : a Survey and a New Approach**", *IEEE Network*, pp.10–21, February 2001.

[Virmani00] Aashu Virmani, Jorge Lobo and Madhur Kohli, "**Netmon : Network Management for the SARAS Softswitch**", in *Network Operations and Management Symposium NOMS2000*, pp.803–816, 2000.

[Walsh04] William E. Walsh, Gerald Tesauro, Jeffrey O. Kephart and Rajarshi Das, "**Utility Functions in Autonomic Systems**", in *ICAC : Proceedings of the IEEE International Conference on Autonomic Computing*, pp.70–77, 2004.

[WBEM12] DMTF, "**Web-Based Enterprise Management**" specifications [online], (2012) [retrieved : May, 2012], http ://www.dmtf.org/standards/wbem.

[Yang09] Gang Yang, Kaibo Wang, Xingshe Zhou, "**An Adaptive Resource Monitoring Method for Distributed Heterogeneous Computing Environment**", *2009 IEEE International Symposium on Parallel and Distributed Processing with Applications*, pp.40–44, 2009.

[Yavatkar00] R. Yavatkar, D. Pendarakis and R. Guerin, "**A Framework for Policy-based Admission Control**", *RFC 2753*, January 2000.