AMSDL : Un langage déclaratif dédié à la surveillance adaptative

Messaoud Aouadj, Thierry Desprats, Emmanuel Lavinal, Michelle Sibilla

AMSDL : Un Langage Déclaratif dédié à la Surveillance Adaptative

Messaoud Aouadj, Thierry Desprats, Emmanuel Lavinal, Michelle Sibilla
University of Toulouse
IRIT UMR 5505
118 Route de Narbonne, F-31062 Toulouse, France
Email :{Firstname.Name}@irit.fr

***Résumé*—De plus en plus d'attentes portent sur la capacité à pouvoir contrôler finement et dynamiquement l'exécution d'une activité de surveillance de réseaux et systèmes communicants. Afin de faciliter la mise en œuvre de plans de contrôle d'une activité de supervision, nous proposons, dans cet article, la définition et une implémentation d'un langage dédié à l'expression de stratégies de surveillance adaptative. Le langage AMSDL (*Adaptive Monitoring Strategy Description Langage*) permet, à des administrateurs de réseaux et de systèmes, ou à des programmeurs de modules logiciels autonomiques, de déclarer à un haut niveau d'abstraction, en plus des ressources à surveiller, la logique de pilotage d'une supervision dynamiquement adaptable aux évolutions des exigences fonctionnelles, informationnelles ou opérationnelles.**

Keywords—adaptive monitoring; domain specific language; policy-based management

I. INTRODUCTION

La supervision (*Monitoring*) est devenue une fonctionnalité de plus en plus importante sur laquelle s'appuient désormais les nouveaux paradigmes de la gestion de réseaux et de systèmes tels que les réseaux autonomiques, les systèmes de communication sensibles au contexte ou encore la virtualisation logicielle des réseaux. En conséquence, de plus en plus d'attentes portent sur la capacité à pouvoir contrôler finement l'exécution d'une activité de surveillance.

La surveillance adaptative peut être définie comme la capacité qu'a une fonction de supervision en cours d'exécution, de décider de, puis d'opérer sans interruption, l'ajustement de son comportement en vue de maintenir son efficacité conformément aux variations de contraintes soit fonctionnelles, soit opérationnelles, et, éventuellement pour améliorer son efficience selon des objectifs d'auto-optimisation. Des précédents travaux [6, 7, 10] ont consisté, selon une approche ascendante, en la définition et à la mise en œuvre d'un plan de contrôle d'une activité de surveillance en cours d'exécution [1]. Il demeure toutefois un réel besoin de fournir aux administrateurs et aux développeurs d'applications de supervision des outils facilitant l'expression de la dynamique à un niveau d'abstraction élevé qui les affranchit des détails des technologies spécifiques sous-jacentes.

Cet article est consacré à la définition et à la mise en oeuvre de AMSDL (*Adaptive Monitoring Strategy Description Langage*), véritable *DSL* (*Domain Specific Langage*) dédié à la programmabilité du pilotage de modules de surveillance adaptative de systèmes communicants. La section II présente les motivations pour un tel langage en particulier à travers un scénario classique de surveillance adaptative et une synthèse de l'état de l'art. Alors que la section III donne un aperçu de l'architecture générale proposée, la section IV présente les éléments principaux de AMSDL. La section V décrit une implémentation de AMSDL réalisée et testée avec succès dans un environnement Java/WBEM. La section VI conclut l'article et ouvre des perspectives pour ce langage dont l'originalité réside dans l'importance accordée à l'expression de l'adaptation.

II. MOTIVATIONS

A. Scénario exemple

Dans cette section, nous présentons un cas simple illustrant une surveillance adaptative. Il constitue un patron de supervision classique à l'établissement d'un diagnostic. Ce scénario est basé sur des adaptations qui concernent à la fois la collecte périodique (*polling*) de propriétés spécifiques au système géré (e.g état opérationnel) et l'observation de la fréquence de réception de notifications particulières (*event reporting*). La stratégie sous-jacente est d'adapter le polling à l'état courant du système géré ou à celui du contexte d'exécution, mais également, quand nécessaire, de bénéficier de données de gestion supplémentaires contribuant à la précision de diagnostic d'incidents.

Initialement, et lorsque le système supervisé fonctionne sainement, un *polling*, rapatriant le statut opérationnel global de chaque élément composant le système, est actif. Un mécanisme de détection de salve de notifications est également activé : on prend pour hypothèse qu'un incident ayant lieu au sein du système supervisé génère une rafale de notifications. Ainsi, lorsqu'une telle salve est détectée, une adaptation de la surveillance est déclenchée. Cet ajustement consiste à suspendre le *polling* courant, à lancer un nouveau *polling* pour collecter des données plus précises, et finalement à pouvoir détecter un silence quant à la réception des notifications. Une autre adaptation est déclenchée lorsqu'un tel silence est détecté et a pour objectif de ramener la surveillance à sa configuration normale.

Par ailleurs, le contexte opérationnel d'exécution de cette fonction de supervision peut être propice à un allègement de son exécution. Par exemple, diminuer la fréquence des collectes en période nocturne ou lors de surcharge de certaines ressources sous-jacentes, puis la rétablir en cas de retournement de situation. Dans ce scénario, l'hypothèse est faite qu'une notification d'un tel changement de situation peut déclencher un réajustement de la période du *polling* portant sur le statut opérationnel global.

B. *Travaux relatifs aux langages pour la supervision*

Dans ce paragraphe nous décrivons plusieurs travaux, les plus proches des nôtres, qui ont contribué à la définition de langages dédiés (DSLs) au pilotage de la supervision de réseaux et systèmes.

ANEMONA [2] (A NEtwork MONitoring Application) est un langage simple conçu pour la programmation d'applications de supervision de réseaux. Les concepteurs s'appuient sur SNMPv3 et la gestion à base de politiques afin de mieux considérer la complexité des réseaux et des systèmes quant à leur supervision, leur contrôle, et leur gestion distribuée. La compilation d'un programme ANEMONA permet la définition et le déploiement de politiques de surveillance ainsi que la prise en charge des événements SNMPv3 correspondants. Il est à l'usage exclusif du monitoring de réseaux TCP/IP à tel point qu'il inclut dans sa syntaxe des concepts spécifiques à SNMP tels que les agents SNMP ou encore les OIDs.

Afin de réduire les coûts de développement et de maintenance des applications de surveillance et de contrôle des systèmes de lancements spatiaux, M.Bennett *et al.* [3] ont décrit un DSL pour le projet Constellation Launch Control System (LCS) de la NASA. Ce DSL, implémenté en Python, fournit un ensemble de constructeurs pour la spécification de mesures et de fonctions de surveillance et de contrôle des fusées de lancement et des engins spatiaux. Il est difficilement transposable à d'autres types d'infrastructures à superviser.

EDBSLang [4] (Event Behaviour Specification and Description Language) est un langage pour la programmation de la supervision de flux de trafics dans les réseaux multi-services autogérés. Son développement est basé sur le paradigme ODM (On Demand Monitoring) qui repose sur le principe fondamental que les composants de supervision doivent être conçus de façon à ce qu'ils puissent adapter leur comportement durant leur exécution. Ce langage est fortement dépendant de la MIB standard *On-Demand*, puisque des éléments de cette MIB font partie intégrante de la syntaxe de ce langage et en limitent la logique.

Apparu en 2010, Frenetic [5] est un DSL pour le paradigme SDN (Software Defined Networking). Il définit en particulier un langage de requête simple mais assez expressif pour permettre aux programmeurs de spécifier ce qu'ils veulent surveiller au sein d'un réseau sans se soucier comment cela est effectivement réalisé. Frenetic ne concerne que la surveillance de flux de paquets d'un réseau et ne supporte pas l'adaptation de cette surveillance.

Aucun de ces DSLs n'apporte un degré de généricité tel que la sémantique soit indépendante du domaine contrôlé, des mécanismes de surveillance et des protocoles sous-jacents les implémentant ou encore du modèle d'information utilisé. Nous avons conçu AMSDL dans l'objectif de pouvoir décrire à un haut niveau d'abstraction des stratégies de surveillance adaptatives pouvant s'appliquer dans un contexte ouvert de gestion intégrée de systèmes complexes.

III. Architecture Générale

Fig. 1 illustre l'architecture générale qui sous-tend la mise en œuvre de l'adaptation d'une activité de surveillance. Les contributions concernant le pilotage d'une telle activité font l'objet de la partie supérieure de la figure et constituent la nouveauté dans nos travaux : elles facilitent la programmation du plan de contrôle de la supervision.

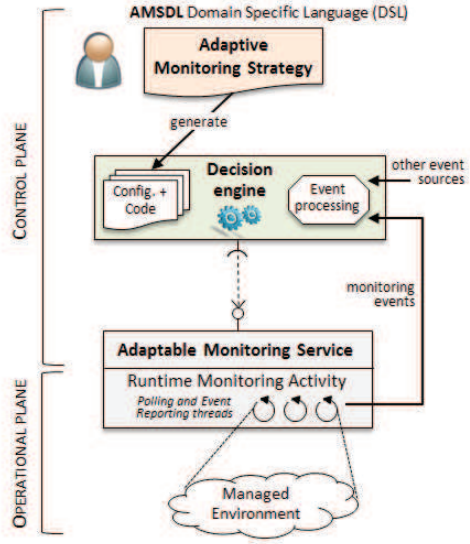

Fig. 1. Architecture générale d'une surveillance adaptative

La gouvernabilité de la surveillance, c'est-à-dire la capacité à décider si et comment une activité de surveillance doit être adaptée, est supportée à la fois par l'expression de stratégies à l'aide du langage dédié AMSDL et par un moteur décisionnel qui, au fil des événements reçus, et en conformité avec ces stratégies, peut déclencher la réalisation d'ajustements du comportement de la surveillance. AMSDL a été conçu de façon à privilégier la représentation, à un haut niveau d'abstraction, de la dynamique d'une activité de supervision. Orienté adaptation, AMSDL propose des instructions permettant de déclarer, au-delà des ressources à surveiller pour un besoin fonctionnel de gestion particulier, la logique gouvernant l'évolution de leur surveillance en fonction des changements des exigences métier, ou de ceux de l'état du système géré ou de celui du contexte opérationnel d'exécution. Pour capturer cette logique, le langage adopte un style inspiré des politiques de type ECA (*Event/Condition/Action*) en permettant de déclarer des événements, et lorsqu'ils peuvent être déclencheurs d'adaptation, de leur associer des instructions d'ajustement de la surveillance. Le programme AMSDL obtenu peut alors servir d'entrée à la configuration d'un moteur automatisant le continuum décisionnel qui contrôle une surveillance adaptative.

L'adaptabilité de la surveillance, c'est-à-dire la capacité de modifier dynamiquement et sans interruption le comportement d'une activité de supervision, est supportée par un service de surveillance adaptable via une interface d'opérations (*Adaptable Monitoring Service* dans Fig. 1). Cette interface est issue de cinq opérateurs atomiques, chacun conduisant à un changement de l'état de la configuration d'une activité de surveillance. Un mécanisme de surveillance de base peut être créé et configuré grâce à l'opérateur d'adjonction $\mathcal{A}$ (*add*). Un mécanisme existant peut être définitivement supprimé, via l'opérateur $\mathcal{D}$ (*delete*), ou bien suspendu puis relancé grâce respectivement aux opérateurs $\mathcal{S}$ (*suspend*) et $\mathcal{R}$ (*resume*). Enfin, la configuration concernant les paramètres qui gouvernent son comportement individuel, ainsi qu'éventuellement la liste des éléments qu'il observe, peuvent être modifiées grâce à l'opérateur $\mathcal{U}$ (*update*). La définition précise de ces opérateurs est décrite dans [6]. Fig. 2 montre un extrait de l'interface d'un service adaptable dont les opérations offertes sont dérivées de ces définitions.

```
module adaptableMonitoringService {
…
// Adaptable Polling Management
interface PollingService {
    addPolling(…);
    deletePolling(…);
    suspendPolling(…);
    resumePolling(…);
    updateConfigurationPolling(…)
    extendScopePolling(…);
    reduceScopePolling(…);
    };
// Adaptable Event Reporting Management
interface EventReportingService {
    addEventReportingListener(…)
    deleteEventReportingListener(…)
    updateConfigurationEventReportingListener(…)
    };
...
}
```

Fig. 2. Extrait de la définition de l'interface d'un service de surveillance adaptable

Le plan opérationnel, c'est-à-dire la réalisation de la fonctionnalité de supervision elle-même, est supporté par un ensemble de *threads* dont le comportement est géré par le plan de contrôle. Ces *threads* sont relatifs à des mécanismes de surveillance de base tels que *polling* et autres *event reporting* dits configurables. La configurabilité d'un mécanisme de surveillance de base fait référence à la capacité d'initialiser et de modifier dynamiquement sa portée ainsi que les paramètres gouvernant son comportement individuel [7]. Cette propriété est *sine qua none* à l'adaptabilité et à la gouvernabilité d'une activité de surveillance.

IV. AMSDL : UN DSL DEDIE A LA SURVEILLANCE ADAPTATIVE

C'est pour simplifier l'expression de politiques de contrôle d'une activité de supervision que le langage déclaratif AMSDL a été conçu. Il permet aux programmeurs de modules de supervision d'exprimer à un haut niveau d'abstraction une stratégie évolutive de surveillance en fonction de besoins métiers et de contraintes opérationnelles importantes à considérer. Les programmeurs se concentrent ainsi sur des questions essentielles telles que : Quoi Surveiller ? Comment organiser le domaine et les modalités de surveillance ? Pourquoi, quand et comment faire varier la surveillance ? Le niveau de généricité proposé est important car il affranchit les programmeurs des détails des environnements d'implémentation cibles, en restant le plus agnostique possible du moteur décisionnel, du système de supervision et du système géré.

Le listing reporté dans Fig. 3 est un extrait de l'utilisation d'AMSDL pour spécifier le plan de contrôle de la fonction de supervision décrite dans le scénario présenté en Section II. On y distingue les trois principales parties selon lesquelles un programme AMSDL est structuré. La première est dédiée à la déclaration des ressources, la seconde à celle des événements et la troisième à la dynamique des activités de surveillance.

```
// Monitored Resources (and Group) declaration part
MonitoredResource MyResource1 = create ("…");
MonitoredResource MyResource2 = create ("…");
MonitoredResource MyResource3 = create ("…");
// Event declaration part
Event IncidentBurstDetectedEv;
Event IncidentSilenceDetectedEv;
Event StressedOperationalContextEv;
Event UnstressedOperationalContextEv;
// Monitoring Strategy declaration part
MonitoringStrategy caseStudy {
   // Profile declaration part
   PollingProfile Hight_Accuracy {
     QoI : Accuracy = 5000,
     Completeness = "EnabledState" , Timeliness = 800 ;
     StopCondition : UnproductiveRequestThreshold = 5;
   }
   PollingProfile Low_Accuracy {
     QoI : Accuracy = 10000,
     Completeness = "EnabledState" , Timeliness = 800 ;
     StopCondition : UnproductiveRequestThreshold = 5;
   }
   EventReportingProfile Detect_Burst {
     Detect : Burst;
     DetectionCondition : DetectionInterval =10000 and
     OccurenceThreshold = 3;
   }
   EventReportingProfile Detect_Silence {
     Detect : Silence;
     DetectionCondition : DetectionInterval =20000 ;
   }
   // Monitoring instructions declaration part
Initial {
  poll MyResource1 accordingTo Hight_Accuracy ;
  listenTo MyResource2 accordingTo Detect_Burst ;
  }
Adaptation StartIncidentDiagno {
  suspend polling MyResource1 according to Hight_Accuracy ;
  poll MyResource3 accordingTo Hight_Accuracy ;
  listenTo MyResource2 accordingTo Detect_Silence
     insteadOf Detect_Burst;
  }
Adaptation StopIncidentDiagno {
  stop polling MyResource3 accordingTo Hight_Accuracy;
  resume polling MyResource1 accordingTo Hight_Accuracy;
  listenTo MyResource2 accordingTo Detect_Burst
     insteadOf Detect_Silence;
  }
Adaptation IncrPollPeriod {
  poll MyResource1 accordingTo Low_Accuracy;
  insteadOf Hight_Accuracy;
  }
Adaptation DecrPollPeriod {
  poll MyResource1 accordingTo Hight_Accuracy;
  insteadOf Low_Accuracy;
  }

   // Adaptation strategy declaration part
when IncidentBurstDetectedEv apply StartIncidentDiagno;
when IncidentSilenceDetectedEv apply StopIncidentDiagno;
when StressedOperationalContextEv apply IncrPollPeriod;
when UnstressedOperationalContextEv apply DecrPollPeriod;
 }
```

Fig. 3. Exemple d'un programme en AMSDL

Au début du programme, les administrateurs doivent spécifier l'objet de la supervision en utilisant les instructions `MonitoredResource` et éventuellement `MonitoredGroup` dont la syntaxe est précisée dans Fig. 4. Ces instructions permettent respectivement d'associer un identificateur à une ressource physique ou logique du système supervisé, ou à un regroupement de ces ressources opéré pour des besoins fonctionnels ou de performance. Ces identificateurs sont utilisés dans la suite du programme en association avec des instructions de supervision. Il est également possible de spécifier l'ajout ou le retrait de ressources d'un groupe.

```
MonitoredResource <resourceID> = create(<path>);

MonitoredGroup <groupID>
<groupID>.increaseScope(<resourceID>); // Ajout ress
<groupID>.decreaseScope(<resourceID>); // Retrait ress
```

Fig. 4. Déclaration de ressources en AMSDL

Les événements sont les catalyseurs des adaptations des activités de surveillance. Le programmeur doit définir les événements impliqués dans la seconde partie du programme. AMSDL propose une version minimale de définition d'un événement grâce à l'instruction `Event` définissant un identificateur d'événement. On retrouve ainsi, dans Fig. 2, la définition de quatre événements correspondant aux changements de situations décrits dans le scénario exemple. Fig. 5 montre d'autres éléments syntaxiques d'AMSDL qui permettent, si besoin, de définir des attributs d'un événement (*e.g.* des données transportées par une notification, des horaires pour un événement temporel), de spécifier la source associée à sa production, ou encore de spécialiser des événements par évaluation d'une expression booléenne, moyen simple d'exprimer les deux premiers niveaux d'une politique ECA.

```
Event <eventID> = {
      Attributes : <data>;
      TriggeredBy : <path>;
}
<childEventID> is_a <fatherEventId> if (<condition)>
```

Fig. 5. Déclaration d'événements

La dernière partie porte sur la spécification de la stratégie dynamique devant piloter l'activité de supervision. Conformément à la syntaxe reportée dans Fig.6, ce bloc est lui-même structuré en trois parties concernant successivement la définition de profils de modalités de surveillance, la spécification d'instructions de surveillance et l'expression de la dynamique d'adaptation.

```
MonitoringStrategy <strategyID> {
       // profiles declarations
       Initial {:
       // poll and listenTo instructions
       }       Adaptation <adaptationID> {
       // poll,listenTo, stop, resume and insteadOf
          instructions
       }
…
       when <eventID> apply <adaptationID>;
…
}
```

Fig. 6. Structure de la définition d'une stratégie de surveillance

La notion de profil offre une expression de haut niveau de la configurabilité des mécanismes de base de supervision. Par ce biais, l'administrateur peut indiquer des exigences quant au fonctionnement de ces mécanismes. Ainsi, pour le moment, deux types de profils sont définis, l'un pour le *polling*, l'autre pour l'*event reporting*. Leur syntaxe est respectivement présentée dans Fig. 7 et Fig. 8. Pour un *polling*, peuvent être essentiellement spécifiés des éléments relatifs à des critères de Qualité de l'Information (QoI) collectée (tels que la fraîcheur, la précision ou encore la complétude de l'information) ou des modalités paramétrables d'arrêt de la collecte. On remarque dans l'exemple du programme la déclaration de deux profils de polling dont la différence essentielle réside dans la valeur de la période séparant deux opérations de consultation successives.

```
PollingProfile <profileID> {
       QoI :
                Accuracy = <integer>;
                Timeliness = <integer>;
                Completeness = <string>;
                …
       StopCondition :
                Duration | Quantity = <integer>;
                …
}
```

Fig. 7. Extrait de la syntaxe de la définition de profil d'un *Polling*

Un profil d'*Event reporting* permet de déclarer des modalités de détection de caractéristiques particulières de réception d'événements. Il est possible de configurer l'observation de l'arrivée d'événements soit en mode *salve*, en mode *silence* ou encore en mode *isochrone* tout en indiquant les paramètres temporels conditionnant la détection. Dans l'exemple reporté dans Fig. 3, deux profils sont déclarés. Le premier spécifie que la détection d'une salve a lieu dès qu'au moins 3 événements sont reçus durant une fenêtre temporelle de 10 secondes. Le second définit un silence dès lors que pendant 20 secondes aucun événement n'a été reçu.

```
EventReportingProfile <profileID> {
       Detect :
                <Burst | Silence | Heartbeat>;

       DetectionCondition :
                DetectionInterval = <integer> and
                OccurenceThreshold = <integer> and
                TemporalAproximation = <integer> ;
}
```

Fig. 8. Extrait de la syntaxe de la définition de profil d'un *Event Reporting*

La partie d'un programme AMSDL consacrée à la déclaration des instructions de surveillance comprend obligatoirement une clause introduite par le mot-clé `Initial` et possiblement différents blocs d'instructions d'adaptation déclarés à l'aide du mot-clé `Adaptation` (pour des raisons de place, nous ne détaillerons pas la syntaxe de ces instructions).

La clause `Initial` décrit l'état initial d'une activité de surveillance. Il s'agit d'indiquer quelles opérations de *polling* et d'*event reporting* doivent être initialement lancées tout en mentionnant le profil comportemental qui leur est associé ainsi que la ressource cible de la surveillance.

Chaque bloc `Adaptation` permet de déclarer et de définir un ajustement de la surveillance. Les instructions autorisées permettent de suspendre, reprendre ou stopper définitivement un mécanisme, ou de changer son profil. Les quatre adaptations définies dans Fig. 3 correspondent respectivement aux quatre ajustements de l'activité de la surveillance décrits dans le scénario exemple de la section II.

Enfin, la dernière partie de la déclaration d'une activité de surveillance permet d'exprimer la stratégie d'adaptation en associant à des événements déclencheur, des instructions d'adaptation telles que définies dans les clauses précédentes. La syntaxe, naturellement bâtie autour des mots-clés `when` et `apply`, est fidèle au style des politiques de type ECA.

V. PROTOTYPAGE ET TEST D'UN COMPILATEUR

La définition du langage AMSDL n'impose pas d'implémentation spécifique. En guise de preuve de concept, nous avons développé un compilateur qui est capable, après avoir analysé un fichier source d'un programme écrit en AMSDL, de générer des règles ECA et du code Java pour le moteur de gestion à base de politiques Ponder2 [8]. Les fichiers générés ont été testés au sein d'un environnement d'exécution comprenant le moteur Ponder2 et un service de surveillance adaptable développé sous la forme d'un client WBEM [9].

L'architecture du compilateur AMSDL développé est bâtie sur trois principaux modules (cf. Fig. 9) : un analyseur lexical, un analyseur syntaxique et un générateur de code. Les deux analyseurs sont génériques (développés respectivement grâce aux outils *lex* et *yacc*) alors que le générateur peut être spécialisé pour une implémentation particulière de l'interface du service de surveillance adaptable. Le générateur de code développé dans le prototype est dédié à l'utilisation d'un moteur de règles ECA Ponder2. Deux fichiers sont alors générés. Le premier contient les règles écrites en langage `PonderTalk` représentant les politiques de surveillance adaptative dérivées de la stratégie AMSDL. Le second fichier est une classe Java dévolue à l'implémentation de la clause Action des règles ECA. Plus précisément, cette classe Java contient l'équivalent des clauses `Initial` et `Adaptation` du programme AMSDL compilé. Ces expressions prennent la forme d'invocations Java RMI des méthodes correspondantes du service de surveillance adaptable (Fig. 2). Nous avons développé une implémentation Java/WBEM de cette interface RMI. Les détails de l'implémentation de ce service de surveillance adaptable, utilisant l'API SBLIM et un Serveur `OpenPegasus` [9] peuvent être trouvées dans [10].

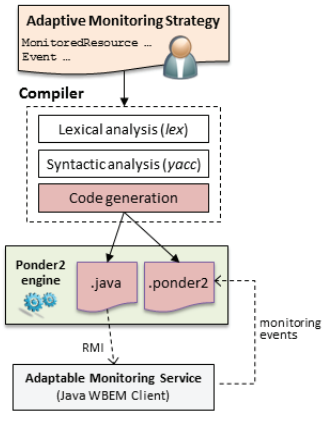

Fig. 9. Architecture du compilateur AMSDL vers Ponder2

Fig. 10 présente un extrait des fichiers générés suite à la compilation du fichier AMSDL décrit dans Fig. 3. L'extrait du fichier Ponder2 est le résultat de la compilation des éléments AMSDL correspondant à l'évolution de la surveillance lorsqu'une situation d'incident est détectée suite à l'occurrence d'une salve de notifications. Ce code `PonderTalk`, représentant une politique interprétable par le moteur de règles Ponder2, est automatiquement généré à partir de la première des quatre clauses `when … apply` déclarée dans la partie stratégie du fichier AMSDL.

Le second fichier généré est un fichier Java qui contient la définition d'une classe particulière qui supporte l'implémentation du type prédéfini Ponder2 `ManagedObject`. Cette classe inclut des annotations `@Ponder2op` qui permettent de lier un message PonderTalk à la méthode java qui lui est associée. Par exemple, le message `PonderTalk StartIncidentDiagno` est associé à la méthode Java `StartIncidentDiagno()` dont la définition consiste en l'invocation Java RMI de trois opérations du service de surveillance adaptable. Il en serait de même pour les autres règles issues de chaque clause AMSDL `Adaptation`. De la même façon, le message `create` est associé au constructeur de la classe, dont la définition consiste à invoquer les opérations du service de surveillance adaptable correspondant aux instructions AMSDL déclarées dans la clause `Initial`.

```
pondertalk := root load: "PonderTalk".
[...]
root/factory at: "caseStudyObject" put: (root load: "MyCaseStudyObject").
so := root/factory/caseStudyObject create.
root at: "caseStudyObject" put: so.

event := root/factory/event create: #( "name" ).
root/event at: "IncidentBurstDetectedEv" put: event.

policy := root/factory/ecapolicy create.
policy event: root/event/IncidentBurstDetectedEv;
action: [ :name |
  root/caseStudyObject StartIncidentDiagno
].
root/policy at: "IncidentBurstDetectedEv" put: policy.
policy active: true.
```

a) Extrait fichier Ponder2

```
[...]
public class MyCaseStudyObject implements ManagedObject {
  [...]
  @Ponder2op("create")
  public MyCaseStudyObject() {
    [...]
    remoteAms = (AdaptableMonitoringService) Naming.lookup("myAMS");
    remoteAms.addPolling("EndSystem1.HighAccuracy.polling", 800, 2, …);
    remoteAms.addEventReporting("EndSystem2.DetectBurst.EventReport", 2,…);
  }

  @Ponder2op("StartIncidentDiagno")
  void StartIncidentDiagno() {
    remoteAms.suspendPolling("EndSystem1.HighAccuracy.polling");
    remoteAms.addPolling("EndSystem3.HighAccuracy.polling", 800, 2, …);
    remoteAms.updateEventRepporting("EndSystem2.DetectBurst.EventReport", 1,…);
  }
}
```

b) Extrait classe Java

Fig. 10. Extraits des fichiers générés par le compilateur

Un tel compilateur évite ainsi à un programmeur ou à un administrateur de devoir maîtriser les langages d'expression de politiques et de programmation sous-jacents. Par ailleurs, nous avons pu noter une réduction du nombre de lignes de code à écrire par le programmeur de l'ordre de 70%. Enfin, la conception modulaire du compilateur permet le développement de nouveaux générateurs de code spécifiques à d'autres moteurs de règles ECA ou à des services de surveillance adaptables particuliers.

Plusieurs études de cas ont été testées sur le prototype. En particulier le scénario de la section II (décrit en AMSDL dans Fig. 3) a été expérimenté, de la phase de compilation jusqu'à l'exécution réelle des adaptations par le client de surveillance WBEM (une fois ces dernières déclenchées par le moteur Ponder2 suite à la réception d'événements appropriés). Les fichiers obtenus à l'issue de la compilation n'ont pas nécessité d'intervention supplémentaire du programmeur, validant ainsi notre prototype de compilateur ainsi que la faisabilité de notre démarche.

Fig 11. illustre sous forme de diagrammes de séquence UML les interactions entre le moteur de politiques Ponder2 et le service de surveillance adaptable. La première partie du diagramme intitulée Initial représente l'exécution par le moteur Ponder2 de la partie du constructeur de la classe `MyCaseStudyObject` reportée dans Fig. 10b. Conformément à la spécification du scénario exemple : un *polling*, rapatriant le statut opérationnel global de chaque élément composant le système, ainsi qu'un *listener* configuré en mode de détection de salve de notifications, sont activés, sous forme de *threads* Java, par le service de surveillance adaptable. La seconde partie du diagramme illustre la réaction du plan de contrôle de la surveillance lorsqu'une salve de notifications est détectée par le *listener*. Celui-ci génère un événement PonderTalk `IncidentBurstDectectedEv` qui est envoyé au moteur Ponder2. La règle ECA décrite dans Fig. 10a est alors évaluée et entraine l'exécution de la méthode `StartIncidentDiagno` de la classe `MyCaseStudyObject` reportée dans Fig. 10b. Le service adaptable suspend alors le *Polling* P1, lance le *Polling* P2 et place le *Listener* en mode détection de silence.

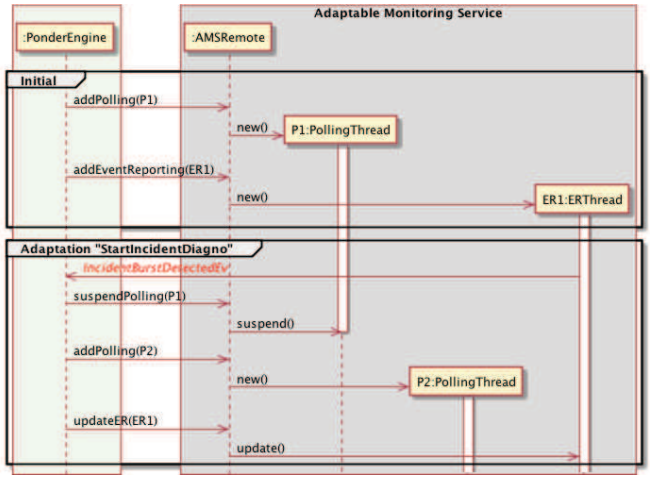

Fig. 11. Diagrammes de séquence de l'initialisation de la stratégie de surveillance et de l'une de ses adaptations

VI. Conclusion et Perspectives

La nécessité sans cesse croissante de pouvoir finement piloter la réalisation d'une fonction de supervision exige de doter les administrateurs et les développeurs d'applications de supervision d'outils leur permettant d'exprimer cette gouvernance. Les efforts déjà réalisés aboutissent souvent à des solutions très dépendantes des mécanismes de surveillance ou des systèmes supervisés. AMSDL est un langage déclaratif générique dont l'originalité réside dans la mise en avant de la dynamique inhérente au plan de contrôle d'une surveillance se voulant adaptative. La version de AMSDL décrite dans cet article peut être facilement enrichie pour intégrer l'expression du contrôle de nouveaux mécanismes de surveillance tel que l'échantillonnage. Par ailleurs, l'architecture modulaire proposée apporte une ouverture possible vers des services et plateformes de supervision standards ou industriels. Des travaux de développement des générateurs de code permettant leur pilotage sont envisagés afin d'assoir la généricité de AMSDL et de poursuivre l'expérimentation sur des cas réels.

References

[1] A. Moui, T. Desprats, "Towards self-adaptive monitoring framework for integrated management", 5th Int. Conf. on Autonomous Infrastructure, Management, and Security (AIMS), Nancy, France, June 13-17, 2011.

[2] E.P. Duarte Jr, M.A. Musicante, H.H. Fernandes, "ANEMONA : a programming language for network monitoring applications", International Journal of NetworkManagement, 18(4), August 2008

[3] M. Bennett, R. Borgen, K. Havelund, M. Ingham, D. Wagner, "Development of a prototype Domain-Specific Language for monitor and control systems", Aerospace Conference, 2008 IEEE , vol., no., pp.1,18, 1-8 March 2008

[4] R. Chaparadza, "A composition language for programmable traffic flow monitoring in multi-service self-managing networks," Design and Reliable Communication Networks, 2007. DRCN 2007. 6th International Workshop on , vol., no., pp.1,8, 7-10 Oct. 2007

[5] N. Foster, A. Guha, M. Reitblatt, A. Story, M.J. Freedman, N.P. Katta, C. Monsanto, J.; Reich, J. Rexford, C. Schlesinger, D. Walker, R. Harrison, "Languages for software-defined networks," Communications Magazine, IEEE , vol.51, no.2, pp.128,134, February 2013

[6] A. Moui, T. Desprats, E. Lavinal, M. Sibilla. "Information models for managing monitoring adaptation enforcement", Int. Conf. on Adaptive and Self-adaptive Systems and Applications (ADAPTIVE), Nice, 22/07/2012-27/07/2012

[7] A. Moui, T. Desprats, E. Lavinal, M. Sibilla, "Managing polling adaptability in a CIM/WBEM infrastructure", Systems and Virtualization Management (SVM), 4th Int. DMTF Academic Alliance Workshop on , vol., no., pp.1,6, 25-29 Oct. 2010

[8] Imperial College London, "Ponder2Project", [http://ponder2.net/], page consultée en Août 2014

[9] The Open Group, "Open Pegasus : C++ CIM/WBEM Manageability Services Broker", [www.opengroup.org/software/openpegasus], page consultée en Août 2014

[10] A. Moui, T. Desprats, E. Lavinal, M. Sibilla, "A CIM-based framework to manage monitoring adaptability," Network and service management (CNSM), 8th Int. Conference , vol., no., pp.261,265, 22-26 Oct. 2012