AMSDL: a Declarative Language for Adaptive Monitoring Control

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

***Abstract*—More and more requirements are given on the ability to precisely control at run time the achievement of a network and communicating systems monitoring activity. This paper gives an overview of AMSDL (*Adaptive Monitoring Strategy Description Language*), which is a language under development dedicated at the expression of adaptive monitoring strategies. AMSDL will provide both the network managers and the software developers of autonomic modules with the capacity to easily declare, more than the resources to be managed, the logics that will govern the dynamic monitoring behavior according to the variations of functional, informational and operational requirements.**

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

I. INTRODUCTION

Monitoring has become an increasingly important functionality upon which are currently built the novel paradigms of network and system management such as autonomic, context-aware or software defined networking. As a result, more requirements are made on the ability to precisely control at run time the achievement of a monitoring activity.

Adaptive monitoring can be defined as the ability an online monitoring function has to decide and to enforce, without disruption, the adjustment of its behavior for maintaining its effectiveness, with respect to the variations of both functional requirements and operational constraints, and possibly for improving its efficiency according to self-optimization objectives. Previous work consisted, according to a bottom-up approach, in the definition and in the implementation of a control plane of a running monitoring activity [1]. It remains however a real need to supply the administrators and the monitoring applications developers with tools facilitating the consideration of the dynamics, at a high level of abstraction, which free them from details of the underlying specific technologies.

This article is devoted to the definition and to one implementation of AMSDL (*Adaptive Monitoring Strategy Description Language*), which is a DSL (Domain Specific Language) dedicated to the programmability of the control of adaptive monitoring modules. AMSDL is intended to be used mainly at the management system level, but it could also be used in case of self-monitored devices, each time an adaptive monitoring is required. In Section II we present the motivations for such a language through both a classical adaptive monitoring scenario and a brief synthesis of the state of the art. While section III gives an overview of the proposed architecture, section IV presents the main elements of AMSDL. Section V describes one AMSDL implementation we achieved and successfully tested in a Ponder2 and Java/WBEM environment. Section VI concludes the article and opens perspectives for this language the originality of which lies in the importance granted to the expression of adaptation.

II. MOTIVATIONS

A. Illustrative Scenario

This section describes a use case of adaptive monitoring that constitutes a classical pattern in the process of establishing a diagnostics. The proposed scenario is based on monitoring adaptations related to both polling of a managed element's specific properties (e.g. operational status) and to the observation of the frequency of notifications reception (event reporting). The underlying goal is to dynamically adapt the polling activity according to the managed system's health but also, when needed, to benefit of new additional data useful in diagnosing accurately incidents.

Initially and when the monitored system is healthy, a polling of a global operational status of each component of the managed system is active. An event reporting mechanism is also activated to detect any sporadic behavior (we assume here a fault in the managed system when a burst of notifications is received). Following this burst detection an adaptation of the monitoring activity is triggered. It consists in suspending the current polling, launching a new polling mechanism to collect more precise data, and finally in switching the mode of observation of event listening from *burst* to *silence*. When no more notifications are received (*i.e.* silent behavior detected), another adaptation is triggered that aims to turn the system back to its normal monitoring configuration.

Besides, the operational context of the monitoring function can be convenient to a reduction of its execution. For example, decrease the polling frequency in night-period or during some underlying resources overload period, and then restore it in case of situation reversal. In this scenario, the hypothesis is made that a notification of such a situation change can trigger

an adjustment of the polling period that concerns the global operational status.

B. Previous Work

In this section, we briefly mention several works that contributed in defining some specific language that considers the control of network and/or complex systems monitoring.

ANEMONA [2] (A NEtwork MONitoring Application) is a simple language designed for programming network monitoring applications. The language designers have relied on the SNMPv3 framework and on the paradigm of policy based management. In fact, compiling an ANEMONA program will allow to deploy policies and to monitor the corresponding events within the SNMPv3 framework. Its usage exclusively concerns IP networks. M.Bennett and Al [3] described a DSL for the NASA Constellation Launch Control System (LCS) project. The DSL, which was implemented using the Python programming language, provides a set of constructs for specifying and programming test, checkout, and launch processing applications for flight and ground systems. EDBSLang [4] (Event Behaviour Specification and Description Language) is a language for programmable traffic flow monitoring for multi-service self-managing networks. EDBSLang developing was mainly inspired and motivated by the ODM (On-Demand Monitoring) paradigm, which is based on the fundamental principle that the monitoring components must be designed in a way that they can adapt their behavior during execution. This DSL strongly depends on the On-Demand MIB standard. Appeared in 2010, Frenetic [5] is a DSL for the SDN (Software Defined Networking) paradigm. Frenetic defines a sub-query language that allows subscribing to streams of information about the network status. It also allows programmers to control the information they receive by using a collection of operators.

However, none of these DSLs brings a high level of genericity such as semantics are no dependent of the monitored domain, of the monitoring mechanisms, of the underlying protocols, and of the management information models. Therefore, we designed our DSL in order that, firstly it fits in an integrated management context and it is completely detached from any application domain, and secondly, it allows network operators to express, in an easy and fully declarative way, their various business needs for adaptive monitoring. Furthermore, in order to satisfy these strong syntax constraints, we have chosen to build our language as an external DSL, instead of an internal one that bends and twists a host language like Python. This choice allowed us to have more control on the language's expressiveness and programming logic.

III. General Architecture

Fig. 1 illustrates the general architecture that supports the achievement of the adaptation of a monitoring activity. In the upper part of the figure are the contributions that concern the governance of the monitoring activity: they facilitate the programming of the monitoring control plane.

The monitoring governability, namely the ability to decide if and how a monitoring activity should be adapted, is first supported by the expression of strategies using the dedicated AMSDL language and secondly, by a decision engine, which in the course of the received events, and in accordance to these strategies, can activate the achievement of the adjustments of the monitoring activity. AMSDL was designed to favor the representation, at a high level of abstraction, of the dynamics of a monitoring activity. Adaptation-oriented, AMSDL proposes instructions to declare, beyond the resources that have to be observed for a particular functional management purpose, the logic that governs the evolution of their monitoring according to the changes of business requirements, or state of the managed system, or operational context of execution. To capture such a logic, AMSDL adopts an ECA (Event/Condition/Action)-based style. It allows to declare events, and when they have to trigger adaptation, to associate them to monitoring adjustment instructions.

An AMSDL program can serve as an input to the configuration of an engine automating the decision-making continuum, which controls the running adaptive monitoring activity. In practice, AMSDL high-level instructions will be translated into low-level configurations. In this article, AMSDL instructions are translated by our code generator module into PonderTalk rules and Java code. However, it is important to stress that AMSDL is not dependent on any technology, and can perfectly be used to target different environments by only adapting the code generator module.

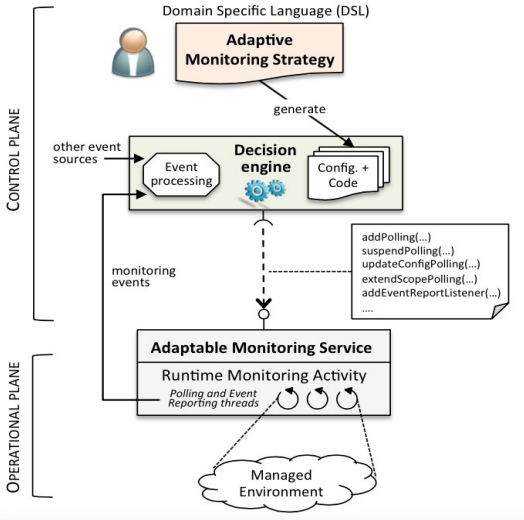

Fig. 1. Architecture générale d'une surveillance adaptative

The adaptability of the monitoring, namely the ability to dynamically modify at run time the behavior of a monitoring activity, is supported by a monitoring service (*Adaptable Monitoring Service* in Fig. 1), which is adaptable via an interface. This interface is the result of five atomic operators (*i.e.*, add, delete, suspend, resume, update), each of them leading to a change of the state of a monitoring activity. The precise definition of these operators is described in [6].

The operational plane, namely the achievement of the monitoring activity itself, is supported by a set of threads, managed by the control plane. These threads are related to basic monitoring mechanisms (e.g. polling, event reporting) that are configurable. The configurability of a monitoring mechanism refers to the ability to initially set and dynamically

modify its scope as well as the parameters that govern its individual behavior [7]. This latest property is *sine qua none* for both adaptability and governability of an adaptive monitoring.

IV. AMSDL: A DSL DEDICATED TO ADAPTIVE MONITORING

It is for making easier the expression of monitoring control policies that the AMSDL declarative language was designed. AMSDL allows the monitoring application programmers to specify with a high level of abstraction their adaptive monitoring strategy, considering important management business requirements and operational constraints, and regardless of the underlying management technologies used. The developers are more focused on essential issues such as: What to monitor? How to organize the monitoring mode and domain? Why, when and how to change the monitoring? AMSDL proposes a high level of genericity while hiding to the developers the implementation technology details and being the most agnostic as possible of the decision-making engine, the monitoring system and the managed system.

An AMSDL program is composed of three main parts (described in the following paragraphs): the first one is for the declaration of the monitored resources, the second one concerns the events and the third one is dedicated to the monitoring activity dynamics. As an example, the listing in Fig. 3 gives an extract of the AMSDL specification of the control plane of the monitoring function described in the scenario presented in Section II.

```
// 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. 2. Example of an AMSDL program

At the beginning of the program, the administrators have to specify the object of the monitoring. They can use the instructions `MonitoredResource` and `MonitoredGroup`. The resource declaration statements allow programmers to bind an identifier to a physical or a logical resource that belongs to the monitored environment, thus making it easier to identify and manipulate within the monitoring program.

Events are the catalysts of the adaptations of the monitoring activity. In the second part of a program, the programmer has to define the events that are involved. AMSDL proposes a minimal version for event definition thanks to the instruction `Event`, which defines an event identifier. Fig. 2 shows the definition of the four events that match to all the situation changes as described in the illustrative scenario. Fig. 3 shows other AMSDL syntactic elements that allow, if necessary, to define attributes of an event (e.g. conveyed data within a notification, hours in case of temporal event), or to specify the source that produces the event, or also to specialize events by adding the definition of a boolean expression as a simple way to express the two first levels of an ECA policy rule.

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

Fig. 3. Events declaration syntax

The last part of an AMSDL program is devoted to the specification of the dynamic strategy that must govern the monitoring activity. According to the syntax postponed in Fig. 4, this block is structured itself in three parts concerning successively the definition of profiles of monitoring modes, the specification of monitoring instructions and the expression of the dynamics of adaptation.

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

Fig. 4. Monitoring strategy definition

The *profile* concept offers the programmers high level way of expression of the configurability of the basic monitoring mechanisms. In the current version of AMSDL, only two types of profiles are defined, one for the polling mechanism, the second one for event reporting mechanism. Their syntax is

presented in Fig. 5. For *polling*, criteria concerning the quality of the collected information (QoI) can be specified (such as freshness, accuracy, etc) as well as settings concerning the stopping mode of the collect. An *Event reporting* profile allows defining the modalities of detection of particular characteristics of events reception. It is possible to configure the observation of the arrival of events in *burst* mode, in *silence* mode, or even in *heartbeat* mode while providing the temporal parameters that control the detection. In the example reported in Fig. 2, four profiles are defined: two for *polling* with a major difference being in the temporal duration between two successive getting operations, and two for *Event reporting*, the first one defines that a *burst* is detected when at least 3 events are received during a temporal window of 10s, and the second one defines a *silence* when no event is received during 20s.

```
PollingProfile <profileID> {
       QoI :    Accuracy = <integer>;
                Timeliness = <integer>; //ms
                Completeness = <string>;
       StopCondition :
                Duration | Quantity = <integer>; //ms }
EventReportingProfile <profileID> {
       Detect : <Burst | Silence | Heartbeat>;
       DetectionCondition :
                DetectionInterval = <integer> and //ms
                OccurenceThreshold = <integer> and
                TemporalAproximation = <integer> ;  }
```

Fig. 5. Extract of the syntax for the definition of monitoring profiles

The part of an AMSDL program dedicated to the declaration of monitoring instructions necessarily includes a clause introduced by the keyword `Initial`. It then possibly contains various blocks of adaptation instructions declared thanks to the keyword `Adaptation`. The `Initial` clause describes the initial state of a monitoring activity. It mentions what are the polling and event reporting operations that have to be initially launched, as well as their respective behavioral profiles and their associated monitored resource. Each `Adaptation` bloc allows to declare and to define an adjustment of the monitoring via instructions for suspending, resuming or definitively stopping a monitoring mechanism or for changing a profile (`insteadOf`). The adaptations that are declared in Fig. 2 respectively refer to the four adjustments of the monitoring activity that are presented in the scenario in Section II. Finally, the last part is devoted at the expression of the strategy of adaptation by associating the triggering events with the adaptation instructions (those defined in the previous clauses). The syntax, naturally built around the two keywords `when` and `apply`, is faithful to the ECA policy style.

V. Compiler Prototype

The definition of the AMSDL language does not impose any specific implementation. As a proof of concept we developed a compiler, which is able to first parse an AMSDL source file, then to generate both ECA rules and Java code for the policy based management engine Ponder 2 [8]. The AMSDL compiler architecture is built on three main modules: a lexical parser, a syntactical parser and a code generator. The two parsers are generic (respectively implemented thanks to *lex* and *yacc*) while the generator can be specialized for a specific implementation of the interface of the adaptable monitoring service. The code generator of the prototype is dedicated to the use of a Ponder2 ECA engine. Two files are generated. The first one contains the `PonderTalk` rules that represent the adaptive monitoring policy derived from the AMSDL strategy. The second file is a Java class devoted to the implementation of the Action clause of an ECA rule. This class contains the equivalent of the `Initial` and `Adaptation` clauses of the AMSDL program. These expressions are Java RMI invocations of the methods defined in the adaptable monitoring service interface. We use the Java/WBEM implementation of this RMI interface, which is detailed in [9].

We tested several use cases on the prototype. In particular, the scenario presented in Section II was experimented, from the compilation step to the real achievement of the monitoring adaptations at runtime. Most importantly, with AMSDL, no knowledge of PonderTalk or Java is required, and consequently we were able to reduce the use case's total number of lines of code up to 70%.

VI. Conclusion et Perspectives

The already realized efforts to provide programmers with DSL for adaptive monitoring often end in solutions very dependent on monitoring mechanisms, protocols, information models or monitored system. AMSDL is a generic and declarative language, the originality of which lies in the emphasis of the control plane of a monitoring aiming to be adaptive. The version of AMSDL presented in this paper can easily be extended by integrating other basic monitoring mechanism such as sampling. Besides, the modularity of the proposed architecture allows to reuse most of the AMSDL compiler as a basis for code generators targeting other *de facto* standard monitoring platforms.

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 Conf., March 2008 IEEE pp.1,18, 1-8

[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 12), Nice

[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/]

[9] 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. 2011