Apex: An Engine for Dynamic Adaptive Policy Execution

Liam Fallon, Sven van der Meer and John Keeney
Network Management Lab, Ericsson, Athlone, Co. Westmeath, Ireland
{liam.fallon sven.van.der.meer john.keeney}@ericsson.com

Abstract—The advent of “Soft Networking”, where networks are composed of virtual nodes and links, promises to dramatically ease the definition and deployment of networks whilst allowing network applications that are limited only by the imagination of the developers of those applications. In such a dynamic environment, the Autonomic Management pattern supervised by policies has been recognised as holding more promise for management of Soft Networks than traditional techniques. We have proposed Dynamic Adaptive Policies as an approach to give classic policies the dynamicity and flexibility to manage such networks and whatever applications are running on them. In this paper, we describe our ongoing work on Apex, an engine that executes and administers Dynamic Adaptive Policies in a scalable and distributed manner.

Index Terms—Automation, Policy, Adaptive Policy, Control Loop, Adaptive Automation, Multi-domain

I. INTRODUCTION

It is a truism to say that in modern telecommunications networks, the only thing that doesn’t change is that the pace of change is continually increasing. Changes such as seamless interaction between multiple technologies in Radio, Core and Fixed Access networks, software defined networks and network virtualization (soft networking), networks on networks and application-defined networks, and the blurring of the lines between telecommunication and data communication networks are well established trends that are generally accepted by practitioners in the field as continuing and accelerating.

The management pattern of Autonomic management, long advocated for managing complex and heterogeneous networks, is increasingly being recognised as a practicable approach for managing such modern networks, with the autonomic loop usually supervised by policies [1][2][3][4]. Whether autonomic loops are implemented at a low level in the network in and between Network Elements (NEs) [1][4] or at higher levels in the network in a control loop [5][6][3], policies may be used to control the execution of the autonomic system.

The current decision selection and decision making processes in network control and network management are by-and-large static: they cannot adapt to changing context (such as new parameters or environmental factors), cannot adapt to shifts in the underlying infrastructure (changes in traffic mix or networking domain behaviour), and sometimes cannot even cope with the speed of normal changes to network topology.

We have proposed Dynamically Adaptive Policies [7], which add adaptiveness to classic policies. Adaptive policies can dynamically adapt their behaviour at runtime in response to changes in internal or external state or context. For example, an adaptive policy may dynamically adapt the actions it applies to a network to keep a given KPI value stable as the load or traffic mix of that network changes. Behaviour boundaries are set using models that specify domain-specific syntax and semantics, catalogues that describe specific policy tasks, and algorithms for realizing specific tasks in different ways.

This short paper describes our ongoing work on Apex, an engine for execution of dynamic adaptive policies. Apex executes adaptive policies and manages their execution life cycle. Apex bounds the adaptive behaviour of policies by actively managing the context in which policies are running across all Apex run-time instances. As we recognise that any system managing adaptive policies must itself exhibit a high degree of adaptability, Apex is architected to be distributed, to be embedded or run independently, and to scale horizontally.

In this paper, related work is discussed in §II. In §III, we describe the Apex engine, in §IV we present some preliminary performance results we have observed in our lab, and in §V we summarise the work undertaken to date and describe the next steps we plan to take in our work.

II. BACKGROUND AND RELATED WORK

The lack of adaptiveness in decision making systems in network management is well known. Current systems usually rely on precompiled knowledge and rules and often require “human-in-the-loop” back-end analysis to support changes that require extensive system, deployment and technology knowledge, increasing both CAPEX and OPEX [8]. As changes occur and the scale of management tasks increase, existing systems inevitably become static as they get more complicated.

Fleck [9] identifies the lack of dynamic feedback to business or system goals in the event-condition-action loop of current policy based management tools. He makes case for a dynamic, adaptive policy system that i) supports “on-the-fly” reconfigurations, ii) is tied to and can modify the goals of the system, iii) incorporates support for context to guide reconfiguration of policies, and iv) incorporates feed-forward and feed-back loops to assure stability of policy-based decision making processes.

One of the few solutions [10] is used in the TMF Zoom project [11]. This solution has the drawback that it promotes context as a selector of a policy. Used in a control loop, this quickly results in unpredictable behaviour, since (rapidly)
changing context results in different policies being selected without overall control over the decision making behaviour.

III. THE APEX ENGINE

This section describes the requirements on the Apex engine and the engine’s architecture. Context management in and the run time behaviour of the engine is also explained.

A. Requirements and Attributes

The purpose of the Apex engine is to run Adaptive Policies [7]. The three key functional requirements from [7] are a) policies must be enabled to make decisions at runtime rather than simply selecting decisions described at design time; b) policies must be empowered to use additional contextual information not contained in the policy trigger in the decision process; and c) policies must have the capability to be adapted or self-adapt [12] at runtime to control the highly dynamic networking domains to which they will be applied.

To be useful, an engine running adaptive policies must itself be highly adaptive. It must allow distribution by thread, execution process, host (physical/virtual), and location so that engine instances are available to run policies in a coordinated manner in small or massive installations. The engine must be deployable as a component embedded in a system or application or run independently. The engine must be scalable horizontally so that as policy execution load increases, a system can be expanded simply by adding more hosts.

B. Architecture

Fig. 1 shows an Apex engine executing adaptive policies in an application. The engine holds a pool of adaptive policy instances which are triggered by events from a domain-specific triggering system. A trigger event contains contextual information (event parameters), thus each instance of a trigger event provides different yet well known and understood information. An adaptive policy then goes through a well-defined state machine (see §III-D) that issues action events to a separate domain-specific actioning system, potentially with additional context information supplied by the adaptive policy.

Fig. 1 also shows how contextual information influences adaptive policies as they run. Business goals (such as give priority to VIP customers) and Domain goals (such as keep Cell Load below 70%) define the boundaries within which an adaptive policy must operate. Goal targets encoded as contextual information can be changed at runtime thus introducing further adaptability in the policies. The context labelled “Other Context” on Fig. 1 represents any other type of information (such as the current Cell Load) that can be retrieved or queried by an executing policy. Such information can (quite often will) impact and influence the decisions the policy will make for a given trigger.

The Apex engine adapts by using incoming trigger event context as well as its environmental context to select which task is executed in each state of its state machine, fulfilling requirement a). The important aspects of the problem domain are modelled as context and an adaptive policy self-adapts its execution as that context changes, fulfilling requirement b). The engine also allows tasks within policies, the logic that selects tasks within policies, and policies themselves to be modified at run time, fulfilling requirement c).

The Apex engine actively performs conflict identification and mitigation. Every data item that an adaptive policy uses (reads and/or writes) is modelled with metadata as context. This allows conflicts to be identified and eliminated at three states in a policy’s life cycle. Firstly, potential conflicts between the policy and other policies in the policy repository can be flagged by the policy design environment to the policy designer at design time by analysing the metadata of the policies. Secondly, the policy deployment system can use policy metadata definitions to check that policies being deployed do not conflict with each other or with already deployed policies. Thirdly, defined events used and the defined context read or written by policies during execution are monitored and analysed to ensure consistency across the set of policies running in and across distributed Apex engines.

Adaptive policies are defined in a Policy Specification. A policy design environment is used to specify i) each state of adaptive policy execution in the policy, ii) the algorithm used to select which task to execute in each adaptive policy execution state, and iii) the logic of tasks that may be executed in each state. A policy deployment component is used to plan and deploy changes to policies running in Apex engines.

C. Context

The Apex engine uses context metadata from the policy specification of each policy it is running to understand the
structure of the context those policies are using and manipulating. Four types of context are considered in Apex (Fig. 2):

- Global Context $C_g$ is available across all policy instances running in a given system.
- Policy Context $C_p$ is available and modifiable only across instances of a particular policy.
- Incoming context $C_i$, event parameters carried into a policy by an event.
- Outgoing context $C_o$, event parameters carried from a policy by an event.

Context metadata describes the structure, constraints, and relationships that context may have. Global context has a single metadata definition. Each policy specification has a separate individual policy context metadata definition; $C_{px}$ for each of $s$ policies, where $x \in \{1, 2, ..., s\}$. A separate individual context metadata definition exists for each incoming event as $C_{ix}$ for each of $q$ incoming events, where $x \in \{1, 2, ..., q\}$ and for each outgoing event as $C_{ox}$ for each of $r$ outgoing events, where $x \in \{1, 2, ..., r\}$. Metadata may be defined using XML, UML or semantically using an ontology language such as OWL. It is made available to all Apex engine instances using a Context Knowledge Base, which is distributed across all Apex engine instances and may be maintained as a semantic model, as runtime objects or in a database. Individual context models are held for $C_g$, $C_p$, $C_i$, and $C_o$.

Context may be collected from many sources in many forms. For example, topology information from an underlying management system or information in RDF format from a weather or news service may be collected and stored as Global Context $C_g$ or Policy Context $C_p$. Business or Domain Goals can be considered as global and policy context, and are therefore handled identically to all other context. The policy logic in adaptive policies is written to ensure that the system converges towards the business or domain goals specified in $C_g$ and $C_p$. For example, an adaptive policy may read the traffic mix or load on a network from $C_g$ and dynamically change the actions it applies to the network to cause the network to converge towards a target KPI value in $C_p$.

### D. Apex at Run Time

The state machine of an Adaptive Policy instance running in an Apex engine is shown in Fig. 3. There are four sequential active MEDA states (Match, Establish, Decide, Act) each of which is adaptive. Each state may examine $C_g$, $C_p$, and $C_i$ to determine which policy task logic should be executed, selected from a set of policy logic tasks available for each state (Fig. 4).

The task logic and the logic to select the appropriate task for a given context can all be changed dynamically. Therefore, by extension, the policy itself has dynamically adaptable behaviour. The task logic that is executed in each active state is selected dynamically by the policy based on the context in which the policy instance is running so these states and by extension the policy itself has dynamically adaptive behaviour.

At startup, a policy instance enters an Inactive state. When its policy context is initialized, it enters the Match state, and waits for an incoming Trigger Event. On reception of an event, it recognises the incoming event and its context and transits to the Establish state. In the Establish state, it determines the situation that generated the incoming event and transits to the Decide state. In that state, it resolves the correct course of action to take and enters the Act state. In the Act state, it determines the best way to realise that course of action and issues an appropriate Action Event. It then transits back to the Match state and awaits another triggering event.

If the Adaptive Policy instance is inactivated or if a serious error occurs in any of the four active states or if the logic of the Adaptive Policy instance is to be updated, the Adaptive Policy transits to the Inactive state. It can then be re-initialised and can resume handling events in the Match state or can exit.

The diagram in Fig. 4 shows how an Adaptive Policy instance processes a Trigger Event through its four active states and issues an Action Event. The manner of execution in each of the four states is identical. In each state, execution commences on reception of an incoming event with $C_i$ and completes by issuing an outgoing event with $C_o$. What distinguishes the states is the Task Selection Logic and Tasks specified for each of the four MEDA states.

An adaptive policy is adaptive because the manner of execution in each state as shown in Fig. 4 is adaptive. In each state, the Adaptive Policy instance executes Task Selection Logic to select a specific task from a set of available tasks. Once a task has been selected, the Adaptive Policy Instance executes the selected task.

Given an incoming event $q$ and an Adaptive Policy instance $s$, the selection function of the Task Selection Logic of an Adaptive Policy state may be expressed as below:

$$Task = f_{select}(C_{iq}, C_{ps}, C_g)$$ (1)

The selection function uses the Incoming Context of incoming event $q$, the Policy Context of adaptive policy instance $s$ and Global Context to select the task to execute. Task Selection
logic is specified in a means that allows it to execute in an Apex engine; a set of rules, a list of parameters such as priority, time or location, an adaptive algorithm expressed in a scripting language such as Python, or a compiled program injected into an Adaptive Policy state implementation at run time.

Once a task is selected, its task logic is executed. Given a selected task and its parameters, an incoming event \( q \) and an Adaptive Policy instance \( s \), the function of the task logic of an Adaptive Policy states may be expressed as below:

\[
Event_r[C_{r}] = f_{task}(C_{iq}, C_{ps}, C_{g}) \tag{2}
\]

The task logic uses the Incoming Context \( C_{iq} \) of the incoming event \( q \), the Policy Context \( C_{ps} \) of the adaptive policy instance \( s \), and the Global Context \( C_{g} \) to execute and to generate an outgoing event \( r \) with outgoing context \( C_{r} \). The event or events emitted by the Act state of an Adaptive Policy is the Action Event that is output by the Adaptive Policy Instance. The task logic is specified in a means that allows it to execute on the Apex engine, which may also be a set of rules, a script in a scripting language, or a programmatic object injected into the Adaptive Policy implementation at run time.

The Apex engine scales easily. The goals and context in which policies are executing may be distributed across Apex engine instances running on multiple physical or virtual machines using technologies such as Distributed Hash Tables [13]. Policies and policy tasks are loosely coupled. All communication between policy tasks is via events, and any state shared between policies or policy instances is modelled, traced and controlled. This allows many distributed policy instances to be deployed to handle a large incoming trigger event load.

IV. PRELIMINARY RESULTS

We have conducted preliminary tests on our Apex engine implementation in the Ericsson Network Management lab. The tests used a straightforward test Adaptive Policy and were run on a single HP ProLiant BL460c Gen8 blade with 32 cores and 256GB of memory. A total of 72 adaptive policy instances were spun up in three Apex engines, each of which ran in a Java virtual machine. The tests were designed to maximise the server load and the tests were run for a period of nine hours.

The results in Fig. 5 are typical of the results observed over many test runs. The results show that the engine can execute 140,000 policy transactions per second on a single blade. An 8 blade rack could handle over 1 million policy transactions per second, a level of performance adequate for deployment in a management system and probably adequate to handle real time traffic policies. A single policy takes 140 ms to execute only during times of heaviest load. Execution times of less than 10 milliseconds were observed for policy transactions when the blade CPU usage was less than 70%. Policy execution is very stable, as is CPU and memory usage. These results show that the Apex engine performs well and that policy execution can be scaled across threads and JVMs on a server.

V. CONCLUSIONS AND FUTURE WORK

We have described the Apex engine that runs Adaptive Policies that dynamically adapt to changes in goals, environment, and incoming event context. The use of metadata allows policy instances to be added, modified and removed on a running system. Unlike classic ECA policies, the metadata approach to context enables conflict detection at a per-state of execution level and per task level. Policy state logic (tasks) and policies can be incrementally developed, as policies evolves from simple to more complex implementations as their deployments mature. The approach is highly scalable and distributable; multiple instances of a policy can run and can share the incoming event load of a particular class, with context sharing between policy instances strictly modelled and controlled.

In our current and future work, we are evaluating the engine on multiple servers in a fully distributed environment. We are determining its characteristics and performance with more complex adaptive policies, specifically to assess the run-time behaviour of such policies when fully distributed.

REFERENCES