A Multi-Platform Event-Driven Controller for Network Experiments

Alina Quereilhac
INRIA
Sophia Antipolis, France

Thierry Turletti
INRIA
Sophia Antipolis, France

Walid Dabbous
INRIA
Sophia-Antipolis, France

ABSTRACT

Network researchers rely on a wide variety of experimentation platforms, ranging from simulators to emulators and live testbeds, to validate new ideas. Many experiment management frameworks have been created to ease up the complexity and time cost of deploying experiments on different platforms. However, providing flexible deployment capabilities for arbitrary platforms remains a challenging problem.

In this work we propose an experiment controller architecture based on event scheduling, designed to enable flexible experiment deployment on diverse platforms. This architecture is capable of handling arbitrary deployment dependencies, both imposed by user requirements and by platform restrictions. It additionally enables flexible resource provisioning, at any point in time during experiment execution, and flexible experiment monitoring events.

Categories and Subject Descriptors

C.2 [**Computer-Communication Networks**]: Miscellaneous

Keywords

Simulation, Emulation, Testbed, Experiment Control

1. INTRODUCTION

Today's network research is aided by a large number of experimentation platforms, used to evaluate improvements in networking protocols and applications. In order to simplify and speed up the evaluation process on complex experimentation environments, including simulators, emulators, testbeds or a mixture of them, a number of experiment management frameworks have been developed to automate deployment. However, most frameworks are built to work on a specific platform or a limited subset of them, and are not easily extensible to arbitrary platforms. Orchestrating an experiment requires to execute actions in the right order so deployment dependencies between resources are respected. This usually implies either that the orchestration engine must have complete knowledge of all possible resources taking part of an experiment, to be able to resolve all possible dependencies, or that the user needs to codify this information himself. The first omniscient orchestration, presents challenges to scale to a large number of different platforms, due to the complexity of resolving deployment dependencies between arbitrary types of resources, and the second requires a high time and effort investment from the user, possibly rendering the use of such automation tool obsolete.

To tackle this problem we propose an event-driven experiment controller architecture, which is easily extensible to support deployment restrictions of new experimentation platforms, and provides the necessary flexibility to enable users to express specific scenario requirements. We do not aim to target at a particular type of experiment scenario, and both measurement and system implementation projects should be compatible with this architecture.

The proposed architecture splits dependency evaluation among different entities, and implements event re-scheduling as a way to properly handle dependency ordering. The users are able to express application orchestration dependencies as part of the experiment definition. Furthermore, it does not constraint resources provisioning to the beginning of the experiment, and enables to trigger monitoring actions upon user-specified conditions.

Existing management frameworks, such as OMF [6] and Emulab [5], already incorporate the use of events into their experiment orchestration engines. However most solutions are not extensible to arbitrary platforms, or restrict resource provisioning to a static stage at the beginning of the experiment.

2. SYSTEM ARCHITECTURE

As shown in Figure 1, the proposed architecture is composed of three major entities: the Experiment Controller (EC), the Aggregate Controllers (AC)s, and the Resources. Resources are abstractions representing virtual or physical components on a specific platform (e.g PlanetLab [2] nodes, ns-3 simulator [1] wireless interfaces, Netns [3] applications, etc). They follow a well defined life cycle through states transitions, such as, *Registered* → *Allocated* → *Started* → *Stopped* → *Release*. State transitions, and configuration changes on Resources are triggered by independent *Actions* (e.g. *Allocate*, *Start*, *Stop*, etc), forwarded by the controlling AC. When the AC invokes an *Action* on a Resource,

CoNEXT Student'12, December 10, 2012, Nice, France.

the latter performs the corresponding operation on the actual platform component it represents.

An AC groups Resources from a single platform. An experiment can involve more than one AC to manage resources from different platforms. All AC entities expose a same API to interact with the EC, but can have different implementations to reflect different platform logics. By creating a new AC implementation, and the associated Resource implementations, it is possible to support arbitrary experimentation platforms, by coding the appropriate deployment dependencies into the AC.

Upon receiving an *Action* message from the EC, the AC evaluates the *Action's* feasibility according to the internal platform logic (e.g. An interface might not be allocated before the associated node is allocated). If the evaluation succeeds, the action is forwarded to the target Resource, which tries to execute the action, and returns a result status. The AC replies to the EC with either *Fail*, meaning that the action failed to be performed, *Retry* meaning that the action might be possible later on, or *Success* meaning that the action was successful.

The EC is the global experiment coordinator in charge of orchestration and monitoring. It receives instructions from the user either through an experiment description (ED) file, or by directly invoking the EC API methods. These instructions are then transformed into *Action* messages and sent to the corresponding AC. The challenge is to execute *Actions* in the correct order to respect generic restrictions on Resource state transitions (e.g. Can not start resource before it is allocated) as well as arbitrary platform restrictions, and at the same time empower users to define their own dependencies (e.g. "Execute application X after application Y"), while still enabling resource provisioning to be performed at any point during the experiment. To this end, the EC incorporates an event scheduler which is able to re-schedule events when conditions for execution are not yet met (e.g. user defined condition not satisfied or AC replied *Retry*). User instructions can be expressed as events composed of an execution *Time*, an *Action*, a user defined *Condition*, and a repetition *Period*. The *Condition* can be arbitrary code, and can be formulated using internal EC information such as Resource state and configuration.

With this architecture we aim to make it possible to express events such as, a) user defined deployment dependencies (e.g. "Application A starts after application B"), b) flexible monitoring (e.g. "Every time channel delay > X log value Y") and c) flexible provisioning (e.g. "At time T provision node N"), for arbitrary experimentation platforms.

3. PROTOTYPE AND FUTURE WORK

We are working on a prototype implementation of the described architecture, loosely derived from the pre-existent NEPI [4] experiment management framework. Contrary to this new architecture, where the deployment of a resource can occur at any moment, in NEPI deployment occurs only at the beginning of the experiment. During this deployment stage, resources in NEPI follow a series of steps to synchronously transition from one state to another. The deployment stage ends when all resources are in state 'Started'. This constrained deployment scheme does not allow the user to specify arbitrary deployment dependencies, nor to provision resources at later points during the experiment execution.

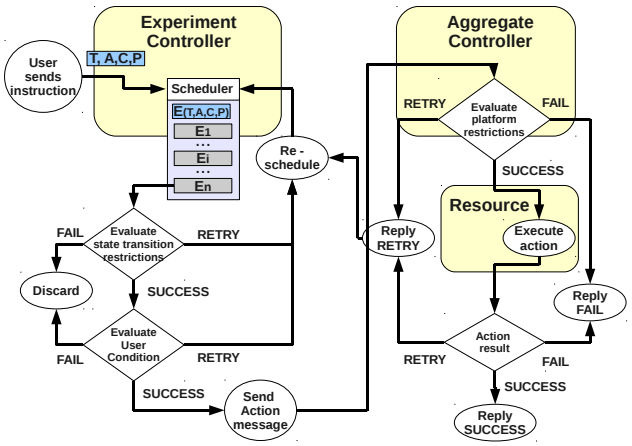

Figure 1: Event re-scheduling logic.

Like NEPI, the event-driven controller prototype implementation is provided as a Python library, and for the moment only supports conducting experiments on the Netns [3] emulator platform. A Netns AC, and Resources to represent Nodes, Ethernet interfaces, Switches and Applications, among others, are already implemented and working under the described architecture.

We plan to extend the prototype to the platforms already supported by NEPI, such as PlanetLab and the ns-3 simulator, and additional ones, such as OMF [6]. Furthermore, we plan to assess the improvements in deployment flexibility, and to evaluate the impact in deployment time, between the prototype implementation and NEPI.

Finally, we envision to formalize the models proposed in this work on a future extended publication.

4. REFERENCES

[1] The ns-3 network simulator. http://www.nsnam.org/.

[2] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: an overlay testbed for broad-coverage services. 33:3–12, July 2003.

[3] M. Ferrari. Netns: End of studies report ubinet master. http://hal.inria.fr/inria-00601848/fr/.

[4] C. Freire, A. Quereilhac, T. Turletti, and W. Dabbous. Automated deployment and customization of routing overlays on planetlab. TRIDENTCOM '12, 2012.

[5] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau. Large-scale virtualization in the emulab network testbed. ATC'08, pages 113–128, Berkeley, CA, USA, 2008. USENIX Association.

[6] T. Rakotoarivelo, M. Ott, G. Jourjon, and I. Seskar. Omf: a control and management framework for networking testbeds. *Operating Systems Review*, pages 54–59, 2009.