FEEDBACKFLOW-An Adaptive Workflow Generator for Systems Management

Artur Andrzejak[1], Ulf Hermann[1], Akhil Sahai[2]
[1]Zuse-Institue Berlin (ZIB), Computer Science Research, Takustr.7, 14195 Berlin, Germany
[2] HP Laboratories, Palo-Alto, CA 94034, USA
{andrzejak, hermann}@zib.de , akhil.sahai@hp.com

Abstract

FeedbackFlow is a framework that implements a general closed control loop of planning – execution - result validation – re-planning, and generates workflows. In this article, we evaluate the design of this framework on scenarios taken from resource construction in Utility Computing environments, and discuss the challenges we have encountered.

1. Introduction

A large share of system management tasks can be broken down into small units, which can be described as *atomic actions*. Examples of such actions include installation or update of a software package from an image, including a new directory in the PATH environment variable, adding entries to the Windows registry, starting a standardized script, or rebooting a system. Understanding which preconditions and post-conditions are necessary for such an atomic action to succeed and which actions have to be taken in which order to achieve a desired goal (e.g. installation of a multi-tier application) currently requires human guidance. This analysis consumes a significant fraction of the time of a human operator and is error-prone (more than 50% of system failures are caused by human operators [1]). Furthermore, when atomic actions are chained the results of each step must be verified manually, consuming more time, and a failure frequently requires the repetition of the analysis.

On the other hand, the pool of atomic actions, if properly parameterized, is mostly similar on the thousands of systems. This implies that the sequences of steps needed to reach a desired system state configuration are also alike. We believe that there is significant potential for automating system management tasks by exploiting these similarities. The idea is to create a pool of atomic and composite action descriptions common to many systems, which would save time of analyzing their dependencies and allow automatic composition of such building blocks. Our larger goal is to automatically extract specifications of atomic actions from sets of management logs.

Assuming a sufficient pool of atomic action specifications (preconditions and effects) derived either manually or by means of machine learning, it is possible to automatically map action task graphs using planning algorithms. In this paper we focus on this approach to automation, and enhance it with the feature of automatic graph correction in response to partial failures.

2. Approach and results

Based on these assumptions, we have built a prototypical framework featuring automatic planning of the action task graph, and the re-planning of this graph in case of partial failures. FeedbackFlow attempts to relieve the system operator from the need to analyze which actions are necessary to achieve a desired state, in which order they must be taken, and whether the execution of these actions was successful.

Specifically, this is done by using AI planning techniques [5] to obtain a task graph of necessary actions derived from a given set of (parameterized) atomic actions with corresponding preconditions and effects (additional input is the system state and the target statement). Such a task graph is subsequently represented as a workflow. During the execution of this workflow, feedback on success or failure of each atomic action is collected in order to update the system state and possibly repeat the execution with an adapted plan. The implementation uses the external components MBP [2], a PDDL [4]-conform planner, and Triana [3], a workflow specification and execution framework.

2. 2. Architecture

FeedbackFlow attempts to automate creation and execution of workflows for system management in an adaptive way. As input it accepts a declarative specification of a collection of available atomic actions, a description of the current system state of the managed system, and a declarative specification of a target system state. Based on this input, FeedbackFlow

Proceedings of the Second International Conference on Autonomic Computing (ICAC'05)
0-7695-2276-9/05 $ 20.00 **IEEE**
IEEE COMPUTER SOCIETY

generates a workflow of the (parameterized) atomic actions, starts a workflow execution engine (which enforces the execution of each atomic action), and compiles the execution results into an updated system state. If the updated system state indicates that the system has not been able to reach the target system state, a new execution cycle is initiated, with a new workflow based on the updated state information. This control loop is executed until the target system state is reached.

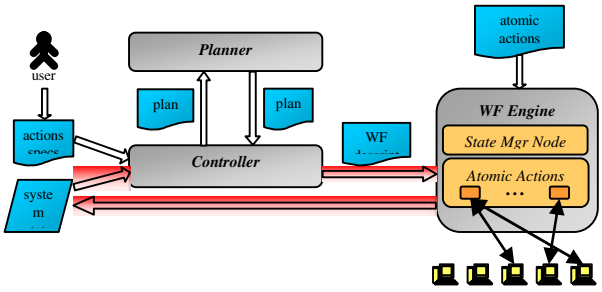

Figure 1: The architecture

2.3. Application to complex resource construction

In order to validate our system we have used examples from the domain of resource construction in Utility Computing environments. A typical task was to assemble a multi-tier system, where each tier required several servers of the same type and running the same application. Different applications were available for different operating systems or sets of them. Further constraints involved a limit on the total cost of the system (cost was modeled as a function of resources).

Our approach worked well for small examples, but we encountered a critical complexity limit, depending on the number of used ``objects'' (resources) and the number of ``exists'' operators. Beyond this limit, the MBP software took prohibitively long to compute a plan (we cut off a computation on a 1 GHz Pentium IV machine if it took longer than 4 hours) and consumed a large amount of memory. Further issues arose in modeling more complex conditions, such as an upper bound on the sum of costs. This required intricate expressions in PDDL, which makes this approach too complex for non-specialists. Our experience with deploying FeedbackFlow showed that the most time-intensive aspect of the system is the specification of the actions, i.e. identifying and encoding their preconditions and effects. Frequently not all relevant characteristics can be captured in the first attempt, which makes debugging necessary. Although our approach facilitates the specification process by using a declarative description language, this stage is still a bottleneck in automating tasks. We believe that methods for automatic deduction of the action specifications (such as Inductive Logic Programming) from collected real-world examples could bring some relief to this problem.

2.4. Design alternatives

As a precursor to FeedbackFlow we attempted to enhance BPEL4WS with declarative target descriptions and support for automatic (re-)planning. We abandoned this approach due to problems with the currently available IBM Alphaworks implementation that supports limited BPEL4WS and the limited variable manipulation power that BPEL4WS 1.1 provides.

3. Conclusions

We have described a prototypical system for adaptive generation and execution of workflows in the domain of systems management. The kernel of *FeedbackFlow* is a closed control loop comprised of state-aware workflow planning, workflow execution, generation of an updated system state, and re-iteration of this process until the desired system state is reached. Using a standardized and generic language such as PDDL allows us to exploit different external planners without changing the system.

We are currently evaluating scalable PDDL compliant planners.

4. Acknowledgment

This research work is carried out in part under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

4. References

[1] G Candea, et al. Building multi-tier dependability. IEEE Computer, (11):60-67, November 2004.
[2] ITC-IRST, Italy, MBP: A Model-Based Planner, 2004
[3] I. Taylor et al. Grid-enabling applications using Triana. In Workshop on Grid Applications and Programming Tools, Seattle, 2003.
[4] M. Fox and D.Long, PDDL2.1: An extension to PDDL for temporal planning domains. Journal of Artifical Intelligence Research, 20:61, 2003.
[5] Stanford University. STRIPS-Stanford Research Institute Problem Solver, 1972

IEEE COMPUTER SOCIETY