SmartFrog and Data Centre Automation

Patrick Goldsack, Paul Murray, Andrew Farrell, Peter Toft
HP Laboratories, Bristol
HPL-2008-35
April 17, 2008*

Automation, Next Generation Data Centre, Declarative Modelling, Workflow, Orchestration

If we are to facilitate service provision in next generation data centres then we need to tackle a number of challenges which lie at the heart of the automation problem for these data centres, including: scale, reliability, security, and service heterogeneity.

In this position paper, we consider the requirements for a solution to these challenges which entail a shift in philosophy from imperative to declarative models for data centres. An important aspect of such a shift is the replacement of workflow as a mechanism for automation. In its place, we propose a declarative approach based on modelling individual components of a system together with their possible configuration states. Dependencies may be specified between components which guard how components may change configuration states. We determine the actions that may be performed to dynamically achieve target states for the system from these models. We have built an experimental system around these concepts and describe this approach in outline.

Internal Accession Date Only Approved for External Publication

To be presented at the Rise and Rise of the Declarative Datacentre (R2D2), Microsoft/HP Joint Workshop, May 2008, Cambridge, United Kingdom.

SmartFrog and Data Centre Automation

Patrick Goldsack, Paul Murray, Andrew Farrell, Peter Toft
HP Laboratories, Bristol, UK
patrick.goldsack@hp.com, pmurray@hp.com, andrew.farrell@hp.com, peter.toft@hp.com

Abstract

If we are to facilitate service provision in next generation data centres then we need to tackle a number of challenges which lie at the heart of the automation problem for these data centres, including: scale, reliability, security, and service heterogeneity.

In this position paper, we consider the requirements for a solution to these challenges which entail a shift in philosophy from imperative to declarative models for data centres. An important aspect of such a shift is the replacement of workflow as a mechanism for automation. In its place, we propose a declarative approach based on modelling individual components of a system together with their possible configuration states. Dependencies may be specified between components which guard how components may change configuration states. We determine the actions that may be performed to dynamically achieve target states for the system from these models. We have built an experimental system around these concepts and describe this approach in outline.

Keywords Automation, Next Generation Data Centre, Declarative Modelling, Workflow, Orchestration

1. Background

If we are to facilitate service provision in next generation data centres then we need to tackle a number of challenges which lie at the heart of the automation problem for these data centres, including: scale, reliability, security, and service heterogeneity.

We need to consider how to build and manage very large data centres, capable of running very large numbers of heterogeneous services that are designed and implemented by different customers with appropriate levels of security, reliability and flexibility. This heterogeneity both in terms of the nature of the service and the range of ownership, makes the task far more complex than running sets of relatively homogenous services or where ownership is vertically integrated from hardware through to end service.

A principal benefit of next generation data centres (NGDCs) is the democratization of services. This would allow small to medium enterprises or even individuals to deliver services to the world in the same way as the web has democratized the delivery of information. For such an ambition, we need to find ways to allow these individuals to develop and run systems securely within well-connected infrastructure without the need to own significant capital assets. This requires that service providers support infrastructure on which these services may be delivered. An early entrant into the space of democratized service platforms is Amazon with its EC2 (Amazon EC2) and S3 (Amazon S3) offerings. However, although these infrastructural services show the way to a certain degree, and illustrate the possibilities for the future, they do not yet go far enough towards providing the levels of security, reliably or flexibility required for solid service delivery.

We enumerate the following challenges for NGDCs.

- Scale: simple back-of-the-envelope calculations show that it is perfectly reasonable to envisage data centres that run 10^5 different service instances on roughly an order of magnitude more virtual machines. Systems of this scale have many implications for the architecture of management systems: failure becomes common place, transactions across the system become impossible, global views or optimizations become problematical and centralized decision making becomes intractable.
- Heterogeneity: since the services will be of an arbitrary nature, decided on by the service owners and not the operators of the data centre, simplifications of aspects such as resource management that come from homogeneity or single ownership cannot be assumed. The data centre management system cannot have accurate knowledge of the services as these are not pre-defined and may change on the fly.
- Security: since in most cases the service code cannot be trusted by the data centre, nor by other services running on the same infrastructure, strong boundaries will need to be put in place between the services and the infrastructure. However, there will still be the need for the services to be managed (including autonomically) and to interact with the core system to achieve end goals. Effectively, therefore, a complete system is defined by the potential interactions of 10^5 individual management entities with limited visibility of the other entities and their states.
- Reliability: With failure being common-place, both of physical components and software, reacting to failure and taking prompt corrective action is necessary within the management system. Furthermore, it is impossible to build systems of this scale without bugs. Consequently self-repair is vital, as is the handling of the failure across the many independent management systems which may be competing to 'fix' the failures within their own sphere of influence.
- Churn: systems of that scale will always suffer churn including the coming and going of services, service reconfigurations due to changing circumstances and requirements, temporary rescaling of a service due to load, short-term overload and congestion, and so on. All of these are occurring asynchronously with respect to each other and to changes in the underlying infrastructure (failure and repair of components, the addition of new and the retiring of old equipment, etc).

These properties make the design of management systems particularly challenging. In this work, we are principally concerned with the orchestration of management actuators, which effect low-level or primitive actions on system components.

2. The Imperative Data Centre

Large-scale management systems traditionally depend on *workflow* execution engines that are triggered in response to an administrator

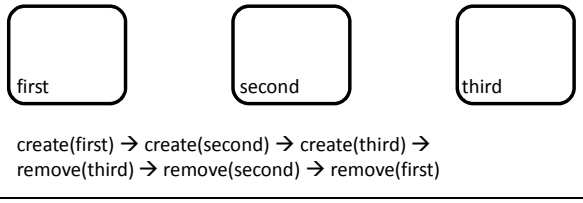

Figure 1. Management Workflow for Simple Scenario

request or, in the case of autonomic systems, some set of internal events. These actions range from provisioning one or more servers or storage volumes, through to the deployment and configuration of software packages. In many cases the actions will range over a number of devices, via interaction with a number of low-level actuators, with complex ordering requirements to ensure that system properties are not violated.

A workflow, according to (The Workflow Management Coalition February 1999), is "[t]he automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules". The sense of this definition is very much an imperative or procedural one: work passes from one agent to the next for actioning. Thus, a workflow entails an ordering of actions to be carried out by agents – a recipe for getting a task done.

It is useful to make a further distinction between localised and distributed workflow. Traditionally, with its root in automating production lines and latterly being used for the automation of office procedures, workflow has been principally characterised by a single controller co-ordinating the passing of work between agents. This is a highly localised view of workflow.

In more recent times, there has been a tendency to associate the term workflow with a collection of distributed procedures, where each procedure is the logic associated with effecting the workflow within any one of the participating entities – an end-point perspective. This distributed view of workflow is grounded within the Remote Procedure Call (RPC) paradigm, where communication between agents is prescriptively synchronised (Alonso et al. 2004). Web Services Composition Languages such as `WS-BPEL` (OASIS) and `WS-CDL` (WS-CDL W3C Working Group) have been defined with the purpose of respectively capturing localised and distributed aspects of this RPC view of distributed workflow.

Unfortunately, distributed workflow-based control of dynamic systems is substantially flawed. Many of these flaws are rooted in the fundamental characteristic of workflow-based systems that they are concerned with prescribing orderings or recipes of actions to be carried out on/by a number of agents or system components. The alternative, as we shall describe, would be to describe the system components themselves as well as relationships between them and have the recipes of actions organically emerge from the declarative models of components. This promotes a much more loosely-coupled approach to federation of distributed system components.

A very simple example is the following. Let's say that there is a system of three managed entities, on which we wish to orchestrate management actions. Each managed entity may be 'created' and subsequently 'removed'. The initial state for each entity is that it is neither created nor removed. Consider also that the second may not be created until the first has been created. Similarly, the third may not be created until the second has been created. Conversely, the second may not be removed until the third has been removed. Similarly, the first may not be removed until the second has been removed. This scenario is captured in Figure 1. If we were to capture the behaviour described (there may be other possible behaviours not described) as a workflow, it would be as a single six-action sequence of actions for creating and removing the managed entities, as depicted in the figure.

Workflow-based management systems typically suffer from the following issues:

- Workflows are hard to analyze, manipulate, and reason about. In just capturing recipes of actions, the state of components is only implied. It's hard to see which action belongs to which component, and it is hard to reason about the states of individual components.
- Capturing the full range of procedures for managing a system of components may require a large number of workflows, which may or may not be aggregated into one or more super-workflows. The overhead in maintaining such workflows is huge because of the nature of what you are representing, namely recipes/procedures for management actions.
- In a system the size of a large data centre there may be hundreds of workflows active at any time: provisioning systems, deploying software, and so on. Each of these is competing for resources and interfering over required interactions with the entities they are attempting to control.
- Workflows are very fragile to changes in the underlying assumptions about a system, such as when a new technology is introduced. Such changes generally force either refactoring or outright abandonment of existing workflows.
- Workflows can be very long-lived - each action taking from seconds to weeks (as would be the case if the steps involve the aspects such as the ordering of new equipment). It is essential to be able to assess the current state of the workflow, but as state is only captured in the steps of a workflow, it is hard to understand the current overall state.
- Error handling usually dominates the structure of a workflow: even the failure recovery can fail, often leaving the system in an unknown state. The fundamental issue is that workflows specify idealized or default behaviour, given their roots in specifying the operation of manufacturing production lines. The original workflows were never meant to be subject to exceptional behaviour (van der Aalst and Weske 2005; Casati et al. 1999). If ever there was something that is more ill-suited to the modelling/specification of dynamic data centres it is workflow! Error handling has been put in place as an afterthought in workflow systems, and it is not particularly effective. It is hard to model behaviour, hard to understand behaviour and hard to know whether you have it right.
- Workflow composition is hard: one can either assume independence thus allowing parallel execution, or impose a specific sequence. It is hard to generate more subtle compositions and interleaving, since it is impossible to reason about the dependencies between the steps of the workflows. After all, how can you effectively compose workflows when you are composing recipes? Fundamentally you need to reason over state.

These, and other problems, lead us to consider an alternative approach to the design of the orchestration of actuators - one which provides a more robust framework for the development of large systems.

3. The Declarative Data Centre

We need to find an approach that is more robust for managing these systems – one which promotes a much more loosely-coupled approach to federation of system components. Such an approach is essential for resolving issues of scale, reliability and fault-tolerance, composition and service heterogeneity in next generation data cen-

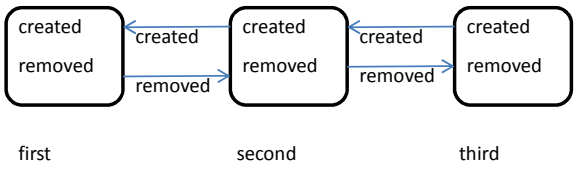

Figure 2. Management Orchestration for Simple Scenario, using Declarative Modelling Approach

tres. Fundamentally, this means moving away from the RPC-based model for distributed workflows, where the operations of components is rigidly and prescriptively synchronised, to a model where components and their states are explicitly modelled and individual components may decide what to do next based on the *states* of other components. Decision making is kept local, rather than it being prescribed on a global/distributed level.

By explicating components' states and grounding decision making on relationships between states, the know-how for managing systems is captured declaratively. This leads to models of systems which are more intuitive and more easily maintained. From these declarative models, we may deduce a number of possible workflows for carrying out particular management tasks. In this sense, the meaning of such declarative models is mutiplicious in entailing a number of allowable procedures or recipes. Whereas any one workflow would be singular in meaning, in prescribing a single procedure. A declarative model for orchestrating management actions on systems is typically more intuitive and simple than an imperative one.

Returning to the previous example, we would capture the given scenario simply as three components whose modelled state is characterised as containing attribute/value pairs for 'created' and 'removed'. We would specify dependencies between components, as captured by the arrows in Figure 2. These dependencies enforce the ordering prescribed in the example narrative. However, any management workflow, such as the one presented in the previous section, is simply an entailment of this model. There is no absolute prescription regarding the behaviour for individual components, and in this sense their management is much more loosely-coupled. That is, we model what *could* be done and not what *should* be done.

We have been experimenting with an approach which accommodates our view and which is based on a number of simple concepts:

- Each entity within the system is self-managed, working on local visibility of the global state - its local model which represents aspects such as desires and current actualities drive the automation of the entity. These entities include the service domains (representing the services and their owners), as well as major components in the core system such as physical nodes or essential services such as DNS.
- We define distributed interaction through 'model exchange' between entities. Protocols are provided that allow the exchange of model data between these entities with well-understood semantics regarding consistency and timeliness, such as Anubis (Paul Murray 2005). Based around a discovery and group membership protocol, we can provide a highly recoverable and reliable layer that is the core of the system: models that get out of synchrony for some reason will get resynchronized within some 'reasonable' time. Failure is recovered by automatically repopulating the models and driving the local system again to the required state.

The architecture is designed so that inconsistencies between local views will eventually be eliminated (if nothing else changes) and so the overall data centre will tend towards a correct and consistent state. The architecture is also such that any interim inconsistencies do not cause any significant or lasting problems, and in the specific case of security properties, we ensure that at no time is security jeopardised.

The approach does mean that we cannot be certain that the entire data centre is configured correctly for the sum of all requirements from the services that run upon it. However this is impossible in any case with a system of such size, as stability is in practice never reached. Furthermore, optimization is hard to achieve across the extent of the data centre with fully decentralised action and models. However, the truth is that this is also a vain expectation when it comes to such large and dynamic systems.

The advantages of having an explicit declarative statement of the desired state at each entity and the degree to which these desires have been achieved, and a clear semantics so far as the consistency of this information within the extended distributed system is concerned, have been significant. It has made building and debugging our experimental systems much easier than might have been expected. We have far greater separation of concerns, better isolation of specific problems, and a more straight-forward compositional approach to adding new features.

4. SmartFrog and Live-State Dependency Models

Our approach to orchestrating management actions for next generation data centres is based on defining declarative models of managed entities/components and their states, and defining dependencies between the components that prescribe how they may change state. Put simply, the dependencies are requirements of one part of state on another: for example 'service running' requires 'operating system installed'. Thus an assertion that the system desires the service to be running will result in the operating system being installed, and only then will it be deemed possible to install and start the service to achieve the desired state.

The system (Andrew Farrell, Paul Murray and Patrick Goldsack) we have built to support this style of automation consists of three major elements: a comprehensive modelling environment based on the concepts contained in the SmartFrog (Automated Infrastructure Laboratory, HP Labs) notation; a way of defining relationships and dependencies over the models; and a runtime environment based on the SmartFrog runtime which provides a means by which the models may be animated.

We consider a managed entity to consist of the sum of a large number of 'fine-grained' state models each representing some minimal aspect of the entity, for example the operating system on a specific node, or a software package, or a volume, and so on. The modelling can be described at whatever level is required. Actions are attached to these partial state models - actions that know how to create, terminate or modify the 'real-life' equivalent of that aspect in response to changes in the model. These models are known as live-state models: as the model changes the actions ensure that the state of the live system follows, and vice-versa.

We define dependencies between these state models (or indeed arbitrary groupings of these models - known as composite state models). These dependencies are parameterized by the states themselves and may be active or passive dependant on the current state. Actions for states that need to be achieved will then be evaluated in an order satisfying the dependencies.

Parameterizable templates can be written and instantiated, causing desired states to be defined, and actions triggered to satisfy these desires according to these dependencies. Many simultaneous or interleaved changes to the overall system model simply implies that a greater number of states need to be achieved, and their order determined by a larger set of dependencies.

A representation in the SmartFrog modelling language for the scenario presented in Figure 2 is as follows.

```
#include "org/smartfrog/components.sf"
#include "org/.../services/dependencies/statemodel/components.sf"
#include "org/.../services/dependencies/threadpool/components.sf"

ManagedEntity extends State {
 sfClass "org.smartfrog.services.dependencies.examples.ManagedEntity";
   created false;
   removed false;
   sink false;
}

createdDependency extends Dependency {
   enabled LAZY on:created;
   relevant (! LAZY by:created);
}

removedDependency extends Dependency {
   enabled LAZY on:removed;
   relevant LAZY by:created;
}

sfConfig extends Model {

   first extends ManagedEntity;
   second extends ManagedEntity;
   third extends ManagedEntity;

   firstCreated extends createdDependency {
      on LAZY first;
      by LAZY second;
   }

   secondCreated extends createdDependency {
      on LAZY second;
      by LAZY third;
   }

   thirdRemoved extends removedDependency {
      on LAZY third;
      by LAZY second;
   }

   secondRemoved extends removedDependency {
      on LAZY second;
      by LAZY first;
   }

}
```

In this SmartFrog model, we define three instances of a 'ManagedEntity': first, second, and third. We also define the four dependencies previously described between these managed entities. There is also an implementation (in Java) of the 'ManagedEntity' component, which may be found in (Andrew Farrell, Paul Murray and Patrick Goldsack). At the current, formative stage of our work the state transitions that a component may make are hard-coded. This aspect should be captured by the orchestration model, and we are maturing our approach to make this possible. This is essential as the model author should have control over it, and it is appropriate that we are able to reason over the transitions that a component may make.

We are currently characterising the semantics of our orchestration approach mathematically, in order to provide a robust account of its features. We are also considering how we may provide verification and simulation support for orchestration models. An example issue that we would typically seek to verify is an *absence* of locking, i.e. deadlock and livelock. Under normal circumstances, we would seek to avoid deploying models which have a capacity to exhibit deadlock. However, it is not necessarily inappropriate that models exhibit livelock, as long as the model author is aware of the ways in which a model may be subject to such locking. Indeed, it is consistent with the very nature of our approach that components may loop through states an unbounded number of times.

From a mathematical perspective, our modelling approach is very simple. This is an advantage in itself in being simple for a model author to understand. It is not inconceivable for workflow authoring that a model author may not fully understand the complexities and subtleties of the modelling approach. Worse still, workflow models are often deployed without any checks made with respect to their structural and behavioural integrity (van der Aalst 2004). A real benefit of our work will lie in the tools that will be developed to assist a model author to understand the true nature of what they are authoring.

5. Current Status

The system outlined is currently a prototype, developed in HP Labs based on the open source SmartFrog system. We have used it in a number of experiments, including in the context of managing very large data centres. We are sufficiently encouraged by the initial experiments to start exploring how best to enhance the current pragmatic programming model with a more sound theoretical basis.

References

Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services, ISBN: 3540440089*. Springer, 2004.

Amazon EC2. Amazon Elastic Compute Cloud. `http://aws.amazon.com/ec2`.

Amazon S3. Amazon Simple Storage Service. `http://aws.amazon.com/s3`.

Andrew Farrell, Paul Murray and Patrick Goldsack. Guide to Orchestration in SmartFrog. `http://smartfrog.svn.sourceforge.net/viewvc/*checkout*/smartfrog/trunk/core/smartfrog/docs/SFOrchestration.pdf`.

Automated Infrastructure Laboratory, HP Labs. SmartFrog: Smart Framework for Object Groups. `http://www.smartfrog.org`.

Fabio Casati, Stefano Ceri, Stefano Paraboschi, and Guiseppe Pozzi. Specification and Implementation of Exceptions in Workflow Management Systems. *ACM Trans. Database Syst.*, 24(3):405–451, 1999. ISSN 0362-5915.

OASIS. Web Services Business Process Execution Language Version 2.0, OASIS Standard, 11th April 2007, at: `http://www.oasis-open.org/apps/org/workgroup/wsbpel`.

Paul Murray. A Distributed State Monitoring Service for Adaptive Application Management. In *2005 International Conference on Dependable Systems and Networks (DSN 2005), 28 June - 1 July 2005, Yokohama, Japan*, pages 200–205, 2005.

The Workflow Management Coalition. Workflow Management Coalition Terminology & Glossary. Document Number: WFMC-TC-1011. Document Status: Issue 3.0. February 1999.

W.M.P. van der Aalst. Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Management, BPM Center Report BPM-04-03. Technical report, BPMcenter.org, 2004.

W.M.P. van der Aalst and Mathias Weske. Case Handling: a New Paradigm for Business Process Support. *Data Knowledge Engineering*, 53(2):129–162, 2005. ISSN 0169-023X.

WS-CDL W3C Working Group. Web Services Choreography Description Language Version 1.0 W3C Working Draft 17 December 2004, at: `http://www.w3.org/TR/ws-cdl-10`.