Using Adaptation Plans to Control the Behavior of Models@Runtime

Maksym Lushpenko, Nicolas Ferry, Hui Song, Franck Chauvel, Arnor Solberg

SINTEF

Models@Runtime 2015
Related to future challenges from Betty

- Higher level adaptation
The models@runtime pattern

• **Synchronization engine implicitly define an adaptation plan**
  – Typically, this plan is derived from the analysis of the difference between the desired (new) and the current state of the system and a set of fixed rules (producing operations going from current to new)

• **Problem:** Need for customization of the adaptation plan.
Motivation and contribution

• There are often alternative ways to enact these adaptations
  – Can significantly affect the effectiveness performance and the quality of service.
  – To avoid adaptations that are not possible for specific platforms/applications

• Contribution
  – a domain specific modelling language for the specification of adaptation plans and,
  – a runtime environment to manage the enactment of such adaptation plans.
Example with CloudMF

• Two main components:
  – A **modelling environment** with a tool-supported domain-specific modelling language (**CloudML**) to model the provisioning and deployment of **multi-cloud** systems
  – A **models@run-time environment** for enacting the provisioning, deployment and adaptation of these systems.
  – Open source
    • www.cloudml.org
Example, Deploying Storm

- AWS EC2 offers ubuntu
- Nimbus
- Zookeeper
- Supervisor
- Relationship
- Requires
- Offers
Proposed approach

1. Create/update runtime model
   - Declarative definition of the desired system's state

2. Generate adaptation plan from the runtime model

3. Execute adaptation plan

4. Reconfigure adaptation plan if needed
   - Definition of the adaptation's behavior
Specifying Adaptation Plans
Adaptation plan DSL
Manipulating Adaptation Plans

• Internal DSL

```java
Activity deploymentPlan = ActivityBuilder.getActivity();
ActivityInitialNode start = ActivityBuilder.controlStart();
Action provision = ActivityBuilder.actionNode("Provision", VM);
ActivityFinalNode stop = ActivityBuilder.controlStop();
Fork fork = ActivityBuilder.forkNode(false);
ActivityBuilder.connect(fork, provision, true);
```

• Runtime Visualization
Runtime environment

Reasoning engine
- Adaptation plan reasoner

- Target Model
- Adaptation Plan Generator
- Adaptation Plan
- Plan Validator
- Plan Execution Engine
- Adaptation Action Executor

Diff

Current Model

Running System

Technology for a better society
Demo – run time view of initial deployment
Demo – run time view of Adaptation
Discussion

• Plan generator: Generate from the result of the diff an adaptation plan. For now, domain (CloudML) dependent, and plan reasoner is application specific (e.g., Storm specific).
  - At some point it needs to be, but probably room for generalisations/automation

• Plan validator: Checks the correctness of the adaptation plan. domain independent.

• Execution engine: Workflow engine that navigates through the plan, exploits reflection of component states. Domain independent
  - Dynamic intervention of running adaptation plans?
    • Can make sense in the cloud domain, where adaptations can last long (in some cases hours)
Conclusion

• A language to build adaptation plans
• A runtime environment integrated within a models@runtime engine to enact such adaptation plans

• Future work
  – Generalizing and applying our approach to other domains
  – Apply similar approach to larger part of the MAPE-K loop, providing frameworks for higher level adaptations.
Thank you!

Contact:
Arnor.solberg@sintef.no
nicolas.ferry@sintef.no