Structural Change Distilling of Ansible Roles

Ruben Opdebeeck, Ahmed Zerouali, Camilo Velázquez-Rodríguez, Coen De Roover
Software Languages Lab, Vrije Universiteit Brussel, Belgium
{ropdebee,azeroual,cavelazq,cderoove}@vub.be

I. Presentation Abstract

A growing number of cloud-native applications are deployed and managed using Infrastructure as Code (IaC), a practice involving domain-specific languages to automatically and reliably create and manage complex digital infrastructures. One of the activities in deploying such infrastructures is configuration management of the infrastructure's machines. This may include installing and configuring software dependencies such as MySQL, Apache, nginx, *etc*. There exist numerous IaC languages that enable this activity, such as Ansible, one of the most popular in use today.

Ansible provides a concept called *roles*, which are reusable collections of configuration tasks and variables. The Ansible Galaxy ecosystem offers a catalogue of more than 25.000 roles, which can be imported by an infrastructure developer to fulfil a goal in their infrastructure deployment. The primary part of a role are its *tasks*, which are a series of steps that are executed on an infrastructure machine to make it reach the desired state, and its *variables*, which can be used to parametrise the behaviour of the tasks. For example, a role that installs and configures nginx may include tasks to install the nginx package on various OS families, tasks to set up the nginx configuration, and variables to customise the configuration values. As such, roles can be considered similar to libraries.

Since roles essentially consist of code, just like libraries in general-purpose programming languages, their code evolves. It is often necessary to extract the changes that were made to this code after the fact, *e.g.*, for change pattern mining, incremental static analysis, *etc*. For general-purpose programming languages, the practice of *change distilling* is well-understood. Tools such as GumTree [1], ChangeDistiller [2], and ChangeNodes [3] can be used to extract changes from tree representations of source code. However, these tools work on generic representations of code, such as generic abstract syntax trees, and offer few opportunities to employ domain-specific knowledge, which may be beneficial for Ansible roles. Moreover, these tools can only extract generic changes, such as the addition or removal of a node, an edit of a node's value, or the relocation of a subtree. Classifying the changes in terms of which elements were changed, is left to the user of the tool.

In this presentation abstract, we will present a domain-specific change distilling algorithm for Ansible roles, which operates on a structural representation of roles. This structural representation consists of the main elements that make up a role, *e.g.*, its tasks, variables, keywords defined on tasks, blocks of tasks, *etc*. As such, it can disregard syntactical refactorings that have no effect on the semantics of the role. The algorithm extracts changes belonging to one of 41 different change types, created orthogonally from combinations of one of the four generic change kinds described above, and the element that has been changed. Examples include the addition of a task, an edit to the value of a variable, the removal of a block of tasks, *etc*.

The algorithm is implemented as a pair-wise depth-first traversal of two structural models for two versions of a role. In general, any subtree that is present in the first version, but not in the second, is extracted as the addition of an element, and vice versa for removal. During backtracking, the sets of additions and removals of children are compared to find relocations of subtrees to another position in the tree. However, when a subtree is slightly edited, it is undesirable to represent this as a pair of an addition and removal, and thus we employ fuzzy comparisons by means of similarity scores. This also allows the algorithm to distil a relocation of an edited element.

The domain specificity stems from the similarity functions, which are customised for many different element types. For example, the similarity of two tasks is defined in terms of the weighted average of the similarities of its keywords. We assign larger weights to keywords that are deemed more important for the task's semantics, such as the action it performs, its arguments, and keywords that have an effect on control flow. Keywords that have little to no semantic effect, such as a task's tags, are assigned smaller weights. As such, two tasks with the same action but different tags are a stronger match than two tasks with the same tags but different actions.

We have previously applied our algorithm in an empirical study of the versioning practices in the Ansible Galaxy ecosystem [4]. In future work, we plan to use the change distiller to predict defect-inducing commits. We are also planning to extend the structural representation to support Ansible playbooks, as well as other IaC languages.

References

[1] J.-R. Falleri, F. Morandat, X. Blanc, M. Martinez, and M. Montperrus, "Fine-grained and accurate source code differencing," in *Proc. 29th ACM/IEEE Int. Conf. on Automated Software Engineering (ASE14)*, 2014.

[2] B. Fluri, M. Wuersch, M. PInzger, and H. Gall, "Change distilling: Tree differencing for fine-grained source code change extraction," *IEEE Trans. on Soft. Eng.*, vol. 33, no. 11, pp. 725–743, Nov. 2007.

[3] R. Stevens and C. De Roover, "Extracting executable transformations from distilled code changes," in *Proc. 24th Int. Conf. on Software Analysis, Evolution and Reengineering (SANER17)*, 2017.

[4] R. Opdebeeck, A. Zerouali, C. Velázquez-Rodríguez, and C. De Roover, "Does infrastructure as code adhere to semantic versioning? an analysis of ansible role evolution," in *Proc. 20th IEEE Int. Working Conf. on Source Code Analysis and Manipulation (SCAM 2020)*, 2020.