Pruning Modes for Deployment Models: From Manual Modeling to Automated Removal of Elements and Their Implications

Miles Stötzner[a], Sandro Speth[b] and Steffen Becker[c]
Institute of Software Engineering, University of Stuttgart, Stuttgart, Germany

Keywords: Pruning, Modes, Deployment Models, Variability Management, TOSCA, EDMM, VDMM.

Abstract: The deployment of modern applications, which consist of multiple components distributed across multiple environments, typically requires a combination of multiple deployment technologies. Besides, applications are deployed in different variants due to different requirements, such as costs and elasticity. Managing deployment variability across multiple heterogeneous deployment technologies is complex and error-prone. Therefore, Variable Deployment Models provide a deployment variability modeling layer independent from the underlying deployment technologies. To ease modeling, elements are pruned, i.e., elements are automatically removed from the deployment due to consistency issues and semantic aspects. However, this might lead to unexpected removal of elements and might mask modeling errors. In this work, we investigate the implications of giving up control when pruning elements and analyze different degrees of pruning. Therefore, we introduce different Pruning Modes, that define which consistency issues and semantic aspects should be considered while pruning elements. We evaluate proposed pruning modes by implementing a prototype, conducting a case study, and experimenting using this prototype.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

1 INTRODUCTION

Managing the deployment of applications is error-prone and complex (Oppenheimer et al., 2003; Oppenheimer, 2003; Brogi et al., 2018). This especially applies to modern applications that are typically distributed across multiple clouds. The deployment of such applications often requires a combination of multiple deployment technologies (Wurster et al., 2021) since these technologies have different areas of application (Bergmayr et al., 2018). Besides, applications must be deployed in different variants due to varying requirements, e.g., costs and elasticity.

Managing such deployment variabilities is complex and error-prone. Therefore, *Variable Deployment Models* (Stötzner et al., 2022, 2023a) provide a method to model deployment variability across heterogeneous deployment technologies. In this method, a modeler, e.g., a software architect or DevOps engineer, creates a Variable Deployment Model of an application, which contains all possible elements, i.e., components, relations, configurations, and component implementations, of any deployment variant of the application. These elements have conditions assigned, which specify the presence of the elements in the deployment. When variability is resolved under a given context, elements whose conditions do not hold are removed, and a deployment model is generated, which can be executed to deploy the application as desired while using multiple deployment technologies. Thereby, consistency checks are conducted to ensure that the generated deployment model complies with its underlying metamodel.

To reduce the repetitive modeling of conditions targeting consistency issues and semantic aspects, elements are pruned, i.e., conditions are automatically generated and assigned to elements (Stötzner et al., 2023c). For example, a condition assigned to a virtual machine states that it should be automatically removed once it no longer hosts any component.

However, automatically removing inconsistent or semantically incorrect elements from the deployment might lead to the unexpected removal of elements and masked modeling errors since consistency checks never complain. For example, an application connects to a database, and a manual modeling error leads to a required but absent database, as shown in Figure 1. The derived deployment model consists of the ap-

[a] https://orcid.org/0000-0003-1538-5516
[b] https://orcid.org/0000-0002-9790-3702
[c] https://orcid.org/0000-0002-4532-1460

Stötzner, M., Speth, S. and Becker, S.
Pruning Modes for Deployment Models: From Manual Modeling to Automated Removal of Elements and Their Implications.
DOI: 10.5220/0012568300003711
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 14th International Conference on Cloud Computing and Services Science (CLOSER 2024)*, pages 149-158
ISBN: 978-989-758-701-6; ISSN: 2184-5042
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

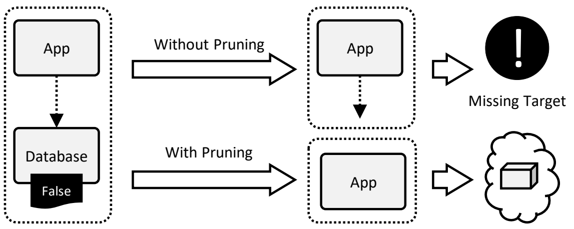

Figure 1: Due to a manual modeling error, the database is absent, which is masked while pruning.

plication and a database connection without a target. Consistency checks detect this modeling error when checking consistency issues and semantic aspects of the underlying metamodel. However, relations without a target are automatically removed when elements are pruned. Hence, the modeling error is masked, and the application is deployed without a database.

In this work, we investigate the implications of giving up control when pruning elements from the deployment, e.g., when pruning overrides manually modeled conditions. Our contributions are as follows:

(i) We define and analyze different Pruning Modes, which use different degrees of pruning elements.

(ii) We implement a prototype and evaluate our modes.

The remainder of this work is structured as follows. In Section 2, we introduce the required fundamentals. We define and discuss Pruning Modes in Section 3. In Section 4, we evaluate them by implementing a prototype and conducting experiments and a case study. Finally, we discuss related work in Section 5 and conclude our work in Section 6.

2 FUNDAMENTALS

In the following, we introduce the required fundamentals considering the automated deployment of applications and managing deployment variability.

2.1 Essential Deployment Models

Deployment technologies, such as Ansible, automate the deployment of applications. Typically, *declarative deployment models* (Endres et al., 2017) are used to describe *what* should be deployed instead of stating *how*. Therefore, deployment technologies automatically derive and execute required deployment tasks.

Deploying modern applications typically requires the combination of multiple deployment technologies (Wurster et al., 2021). For example, Terraform provisions a virtual machine and Ansible installs web

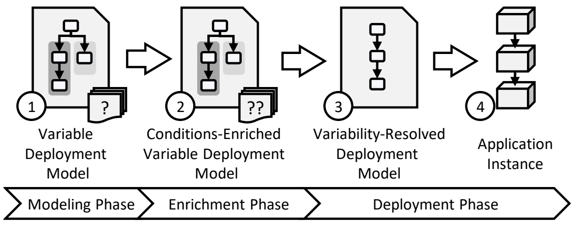
1
?
Variable Deployment Model
2
??
Conditions-Enriched Variable Deployment Model
3
Variability-Resolved Deployment Model
4
Application Instance
Modeling Phase
Enrichment Phase
Deployment Phase

Figure 2: Overview of the method for pruning elements (figure based on Stötzner et al. (2023c)).

servers or databases on this virtual machine. Moreover, migrating between deployment technologies is challenging. Therefore, we can use the *Essential Deployment Metamodel (EDMM)* (Wurster et al., 2019) to model *Essential Deployment Models*, which can be (i) mapped to the most popular deployment technologies (Wurster et al., 2019) and (ii) executed while using multiple deployment technologies in combination (Wurster et al., 2021).

2.2 Variable Deployment Models

An integrated deployment variability management layer is required to manage deployment variability across combined deployment technologies. Therefore, we introduced the Variable Deployment Modeling Method (Stötzner et al., 2022, 2023a), which generates Essential Deployment Models once deployment variability is resolved. To support modeling, we introduced the pruning of elements, i.e., the automated removal of elements due to consistency issues and semantic aspects considering the underlying metamodel (Stötzner et al., 2023c). The resulting method is shown in Figure 2.

In the modeling phase, the modeler (i) creates a Variable Deployment Model that contains all elements of the deployment variants of an application with conditions assigned specifying their presence in the deployment. The modeler is not required to model conditions targeting consistency issues and semantic aspects of the underlying EDMM since they are automatically addressed in the subsequent enrichment phase. In the enrichment phase, the software component *Condition Enricher* (ii) generates conditions targeting consistency issues and semantic aspects and assigns them to the elements. For example, a condition assigned to a virtual machine states that it should be automatically removed once it no longer hosts any component. In the deployment phase, the operator passes *variability inputs* to the software component *Variability Resolver*, which (iii) automatically resolves the variability and generates a *Variability-Resolved Deployment Model*: The Variability Resolver evaluates all conditions, removes elements whose conditions do not hold, and conducts consistency checks to ensure that, e.g., every relation has a source and a target. The generated model (iv) is then executed to deploy the desired variant.

2.3 Variable Deployment Metamodel

To model deployment variability, the *Variable Deployment Metamodel (VDMM)* (Stötzner et al., 2023c) extends EDMM by conditional elements. An EDMM model represents the deployment of an application as a topology graph:

- *Components* represent application components, e.g., virtual machines and web applications.
- *Relations* represent relationships between components, e.g., hosting relations.
- *Properties* represent component and relation configurations, e.g., ports and credentials.
- *Deployment artifacts* represent component implementations, e.g., Node.js files and binaries.

VDMM extends EDMM as follows:

- *Conditional elements* are elements, i.e., components, relations, properties, and deployment artifacts, that can have conditions assigned specifying their presence in the deployment.
- *Variability inputs* represent the context under which the variability is resolved, e.g., a requirement for elasticity.
- *Variability conditions* are conditions over variability inputs specifying the presence of an element.
- *Manual conditions* are variability conditions that are manually modeled and assigned to elements, e.g., a condition checking if elasticity is required according to given variability inputs.
- *Pruning Conditions* are variability conditions that are automatically generated and assigned.

2.4 Pruning Conditions

The Condition Enricher automatically generates and assigns pruning conditions to elements to reduce the number of repetitive manual conditions addressing consistency issues and semantic aspects.

Pruning conditions targeting consistency issues considering EDMM cause the automated removal of any inconsistent element from the deployment. We refer to these pruning conditions as *consistency pruning conditions* and to their generation as *consistency pruning*. For example, for relations, a condition is generated, which *checks if the relation source and the target are present*. Also, a condition is generated for

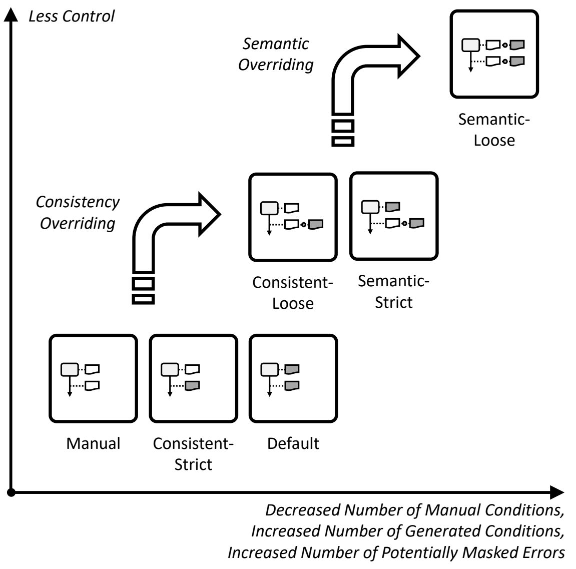

Figure 3: Overview of the Pruning Modes.

properties and deployment artifacts, which *checks if the container is present*.

Pruning conditions targeting semantic aspects considering EDMM cause the automated removal of any semantically incorrect element from the deployment. We refer to these pruning conditions as *semantic pruning conditions* and to their generation as *semantic pruning*. For example, for components with at least one incoming relation, a condition is generated, which *checks if any incoming relation is present*. Also, for components with at least one deployment artifact, a condition is generated, which *checks if any deployment artifact is present*.

3 PRUNING MODES

For analyzing different degrees of pruning, we define *Pruning Modes*, which generate different kinds of conditions depending on the modeler's requirements.

3.1 Overview

In total, we define six Pruning Modes. They range from not generating any conditions and having full control to generating the most possible conditions while giving up control. We define the modes in an incremental order, in which a mode includes its previous mode considering generated conditions. For example, a Pruning Mode, which generates and assigns consistency pruning conditions to elements without manual conditions, is followed by a Pruning Mode, which generates and assigns consistency pruning con-

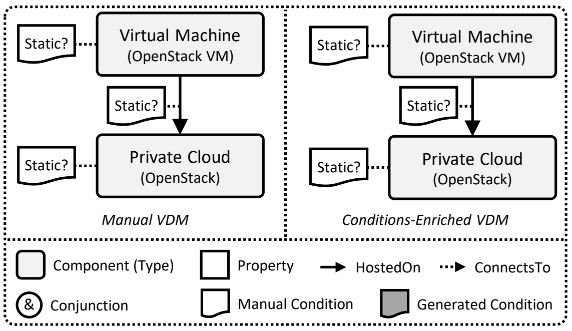

Figure 4: The example of the Manual Pruning Mode. On the left, the Manual Variable Deployment Model (VDM) is given as created by the modeler. On the right, the Conditions-Enriched Variable Deployment Model is given as generated by the Condition Enricher.

ditions not only to elements without manual conditions but also to elements with manual conditions.

A graphical overview of the Pruning Modes is given in Figure 3. Pruning Modes, which are on the right side, require fewer manual conditions to be modeled: They generate more conditions considering the modes that are left of them. The increased number of generated conditions implies a higher risk of potentially masked errors, and the risk of unexpected removal of elements rises. Moreover, the modeler gives up more control: Pruning Modes on the top override more manual conditions than those on the bottom.

The structure of the describing the Pruning Modes is inspired by the structure of component hosting patterns presented by Yussupov et al. (2021). Each Pruning mode contains an example based on the original pruning paper (Stötzner et al., 2023c). The corresponding example models are open-source and available on Zenodo[1] and GitHub[2].

3.2 Manual Pruning Mode

Problem: *How to model deployment variability while obtaining full control over consistency issues and semantic aspects?*

Solution: The Condition Enricher does not generate and assign any pruning condition and, therefore, directly returns the given Variable Deployment Model without any modifications. Therefore, only manual conditions are considered by the Variability Resolver.

Example: On the left of Figure 4, the deployment of a virtual machine is given. This virtual machine is only required for a static deployment, i.e., if no elasticity is required. Hosted components are not shown for the sake of brevity.

[1]https://doi.org/10.5281/zenodo.10363694
[2]https://github.com/opentosca/opentosca-vintner

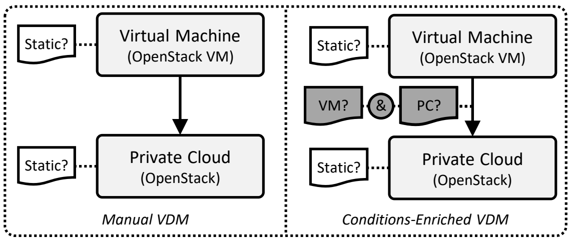
Static?
Virtual Machine (OpenStack VM)
Static?
Private Cloud (OpenStack)
Manual VDM
Static?
Virtual Machine (OpenStack VM)
VM?
&
PC?
Static?
Private Cloud (OpenStack)
Conditions-Enriched VDM

Figure 5: The example of the Consistent-Strict Pruning Mode. "VM" is short for "Virtual Machine" and "PC" for "Private Cloud".

When applying the Manual Pruning Mode, no conditions are generated by the Condition Enricher. Thus, all conditions must be modeled by hand, e.g., the virtual machine has a condition assigned checking if a static deployment is required. Hence, the manually modeled Variable Deployment Model matches the Conditions-Enriched Variable Deployment Model, as shown on the right of Figure 4.

Result: Since no conditions are generated, all conditions must be modeled by hand. Thus, the modeler must consider consistency issues and semantic aspects when creating the Variable Deployment Model. Also, no modeling errors are masked since no conditions are generated.

3.3 Consistent-Strict Pruning Mode

Problem: *How to model deployment variability without considering consistency issues for elements without manual conditions while obtaining full control over them?*

Solution: Before variability is resolved, the Condition Enricher iterates over every element of the given Variable Deployment Model. Thereby, the Condition Enricher generates and assigns consistency pruning conditions to elements that have no manual conditions assigned. Semantic pruning conditions are not generated. Also, elements that have manual conditions assigned are not touched.

Example: On the left of Figure 5, the deployment of a virtual machine is given. This virtual machine is only required for a static deployment. Hosted components are not shown for the sake of brevity.

When applying the Consistent-Strict Pruning Mode, consistency pruning conditions are generated and assigned to elements without manual conditions. For example, the conditions checking for the presence of the relation source and relation target are generated and assigned to the hosting relation of the virtual machine and OpenStack. However, it is required to model the remaining conditions by hand, e.g., the

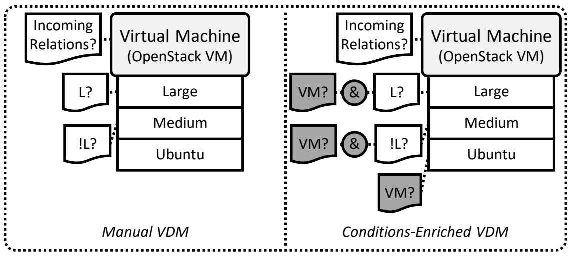

Figure 6: The example of the Consistent-Loose Pruning Mode. “L” is short for “Large”.

condition assigned to the virtual machine checking if a static deployment is required.

Result: Since consistency pruning conditions are generated, the modeler must not model them. Inconsistent elements are automatically removed. Also, since these conditions are only assigned to elements without manual conditions, the modeler obtains full control. However, modeling errors are masked since inconsistent elements are automatically removed.

3.4 Consistent-Loose Pruning Mode

Problem: *How to model deployment variability without considering consistency issues?*

Solution: Before variability is resolved, the Condition Enricher iterates over every element of the given Variable Deployment Model. Thereby, the Condition Enricher generates and assigns consistency pruning conditions to every element, including elements with manual conditions. Semantic pruning conditions are not generated.

Example: In Figure 6, the deployment of a virtual machine is given. The virtual machine is configured to use Ubuntu and either a medium or a large tier. If the virtual machine does not host anything, it should be removed from the deployment.

When applying the Consistent-Loose Pruning Mode, only the conditions checking for the tier and any hosted component must be modeled, as shown on the left of Figure 6. Consistency pruning conditions, e.g., the conditions checking for the presence of the virtual machine at every property, are automatically generated and assigned to elements without and with manual conditions, as shown on the right of Figure 6.

Result: Since consistency pruning conditions are generated, the modeler must not model them. Hence, inconsistent elements are automatically removed, which masks modeling errors. Besides, since pruning conditions are assigned to every element, the modeler loses control and cannot override this behavior.

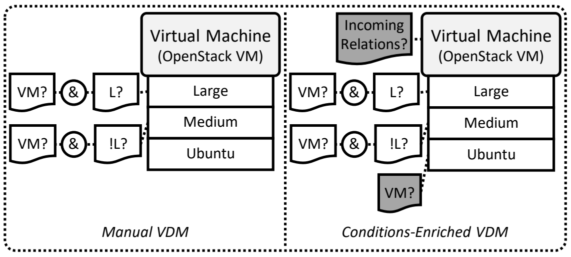

Figure 7: The example of the Default Pruning Mode.

3.5 Default Pruning Mode

Problem: *How to model deployment variability without considering consistency issues and semantic aspects for elements without manual conditions while obtaining full control over them?*

Solution: Before variability is resolved, the Condition Enricher iterates over every element of the given Variable Deployment Model. Thereby, the Condition Enricher generates and assigns consistency and semantic pruning conditions to the elements, which have no manual conditions assigned. Elements that have manual conditions assigned are not touched.

Example: In Figure 7, the deployment of a virtual machine is given. The virtual machine is configured to use Ubuntu and either a medium or a large tier. If the virtual machine does not host anything, it should be removed from the deployment.

When applying the Default Pruning Mode, consistency and semantic pruning conditions are only generated and assigned to elements without conditions. Thus, the conditions checking for the presence of the container at the tier properties must be modeled, as shown on the left of Figure 6. Meanwhile, the condition checking for incoming relations is generated and assigned to the virtual machine, and a condition checking for the presence of the virtual machine to the image property, as shown on the right of Figure 6.

Result: Since consistency and semantic pruning conditions are generated, the modeler must not model them. As a result, inconsistent elements are automatically removed, which also masks modeling errors. However, this only applies to elements without manual conditions. Thus, the modeler can override this behavior and obtain control.

3.6 Semantic-Strict Pruning Mode

Problem: *How to model deployment variability without considering (i) consistency issues for any elements and (ii) semantic aspects for elements without manual conditions while obtaining control over them?*

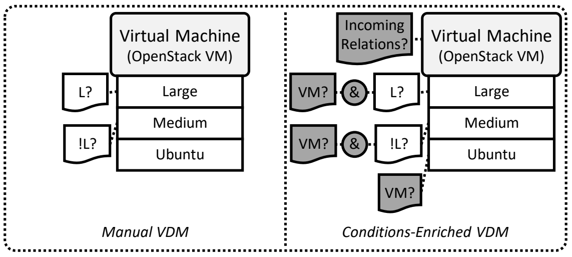

Figure 8: The example of the Semantic-Strict Pruning Mode.

Solution: Before variability is resolved, the Condition Enricher iterates over every element of the given Variable Deployment Model. Thereby, the Condition Enricher generates and assigns consistency pruning conditions to every element and semantic pruning conditions to elements without manual conditions.

Example: In Figure 8, the deployment of a virtual machine is given. The virtual machine is configured to use Ubuntu and either a medium or a large tier. If the virtual machine does not host anything, it should be removed from the deployment.

When applying the Semantic-Strict Pruning Mode, only the conditions checking for the expected static workload must be manually assigned, as given on the left of Figure 8. The remaining conditions are automatically generated, as given on the right of Figure 8. For example, the semantic condition checking for incoming relations is generated and assigned to the virtual machine. Also, the condition checking for the presence of the virtual machine is generated and assigned to each property to ensure consistency.

Result: Since consistency and semantic pruning conditions are generated, the modeler must not model them. As a result, inconsistent and semantically incorrect elements are automatically removed, which also masks modeling errors. However, semantic pruning conditions are not assigned to elements with manual conditions. Thus, the modeler can override this behavior and obtain control.

3.7 Semantic-Loose Pruning Mode

Problem: *How to model deployment variability without considering consistency issues and semantic aspects?*

Solution: Before variability is resolved, the Condition Enricher iterates over every element of the given Variable Deployment Model. Thereby, the Condition Enricher generates and assigns consistency and semantic pruning conditions to every element. This is described in more detail in the original pruning publication (Stötzner et al., 2023c).

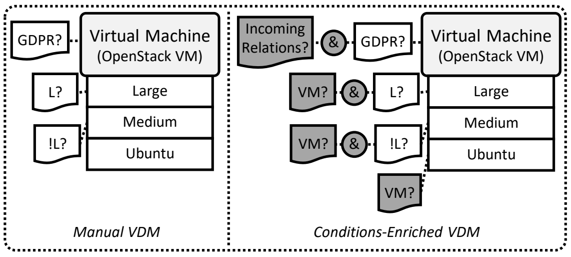

Figure 9: The example of the Semantic-Loose Pruning Mode. “GDPR” is short for “General Data Protection Regluation”.

Example: In Figure 6, the deployment of a virtual machine is given, which is only required when the General Data Protection Regulation (GDPR) holds. The virtual machine is configured to use Ubuntu and either a medium or a large tier. If the virtual machine does not host anything, it should be absent.

When applying the Semantic-Loose Pruning Mode, only the conditions checking for GDPR and the expected static workload must be manually assigned, as given on the left of Figure 9. The remaining conditions are automatically generated, as given on the right of Figure 9. For example, the semantic condition checking for incoming relations is generated and assigned to the virtual machine. Also, the condition checking for the presence of the virtual machine is generated and assigned to each property to ensure consistency.

Result: Since consistency and semantic pruning conditions are generated, the modeler must not model them. As a result, inconsistent and semantically incorrect elements are automatically removed. Thus, the number of conditions that must be manually modeled is significantly reduced. However, this also masks modeling errors. Also, since pruning conditions are assigned to every element, the modeler loses control and cannot override this behavior.

4 EVALUATION

To evaluate our Pruning Modes, we implement a prototype. We use our prototype for experiments, in which we model the examples of each Pruning Mode and ensure that they are correctly resolved. Moreover, we conduct a case study using our prototype, in which we model and analyze the motivating scenario of the original pruning paper (Stötzner et al., 2023c) using different Pruning Modes. The prototype, the models, and the step-by-step guide are open-source and publicly available on Zenodo[1] and GitHub[2].

4.1 Prototype

Our prototype is based on the open-source TOSCA preprocessing and management layer OpenTOSCA Vintner (Stötzner et al., 2022). TOSCA (OASIS, 2020) is an open standard for deploying and managing cloud applications in a vendor-neutral and technology-independent manner. To conform with EDMM during our evaluation, we only use features from TOSCA Light (Wurster et al., 2020), an EDMM-conform subset of TOSCA. OpenTOSCA Vintner supports Variability4TOSCA (Stötzner et al., 2022, 2023c,a,b), which extends TOSCA by conditional elements and their pruning, and thus, implements VDMM. OpenTOSCA Vintner instructs TOSCA orchestrators to execute generated TOSCA models.

We extend Variability4TOSCA and OpenTOSCA Vintner to support Pruning Modes. Therefore, we extend Variability4TOSCA by the possibility of modeling the Pruning Mode and OpenTOSCA Vintner to generate and assign pruning conditions as specified by the modeled Pruning Mode.

4.2 Experiments

We conduct experiments to show the technical feasibility of the examples of our modes: We enrich a Variability4TOSCA model for each Pruning Mode example using OpenTOSCA Vintner and ensure that the generated Variability4TOSCA model is as expected. For each Pruning Mode, we proceed as follows.

1. Manually model the Manual Variability4TOSCA model of the Pruning Mode example.
2. Manually model the Conditions-Enriched Variability4TOSCA model of the mode example.
3. Enrich the Manual Variability4TOSCA model using OpenTOSCA Vintner, i.e., generate the Conditions-Enriched Variability4TOSCA model.
4. Assert that the manually modeled Conditions-Enriched Variability4TOSCA model matches the generated Conditions-Enriched Variability4TOSCA model.

4.3 Case Study

We further conduct a case study to evaluate our Pruning Modes. In contrast to the experiments, we focus not on the enriching phase but on the end-to-end variability resolving with subsequent deployment.

For this case study, we use the webshop application that served as the motivating scenario for the original pruning paper (Stötzner et al., 2023c). We model the application using different Pruning Modes,

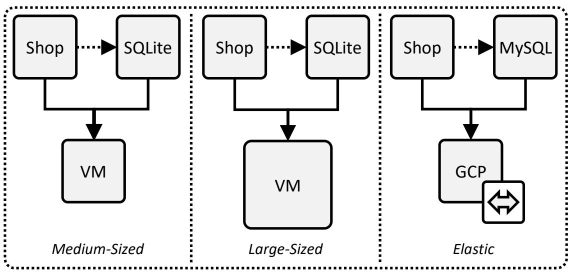

Figure 10: The different deployment variants of the case study (based on Stötzner et al. (2023c)).

analyze the different models, and deploy the application. Our case study shows the technical feasibility of our Pruning Modes and that our extensions to the prototype are sufficient to implement Pruning Modes.

4.3.1 Scenario

In our case study, we manage the deployment variability of a webshop application, which can be either deployed medium-sized on-premise, large-sized on-premise, or elastic in the cloud. The webshop consists of a shop component, which connects to a database. For simplicity, we chose this scenario, which already considers various deployment variabilities. Other scenarios might include multiple clouds.

In the static on-premise deployment variants, the shop component and an SQLite database are both deployed on the same virtual machine on a local OpenStack (OS) instance, as shown on the left and in the middle of Figure 10. To serve different static workloads, the virtual machine can be either of medium or large size. Thus, this deployment is only suitable if the workload is static and known upfront.

However, to serve unpredictable workloads, elasticity is required. Therefore, in the elastic cloud deployment variant, the webshop application is deployed on the public cloud Google Cloud Platform (GCP), as shown on the right of Figure 10. The shop component is deployed with auto-scaling enabled on GCP AppEngine, and the database is deployed on GCP CloudSQL with high availability enabled. Also, the shop component is configured to use the MySQL database dialect since a MySQL database is used in this variant.

4.3.2 Modeling

We are interested in the number of manual conditions and the number of generated conditions when using different Pruning Modes. Therefore, we manually model the Variability4TOSCA model of the webshop application for each Pruning Mode. To ensure that variability is correctly resolved, we gener-

Table 1: The different numbers of manual and generated conditions when modeling the case study using different modes.

Pruning Mode	Manual Conditions	Generated Conditions
Manual	52	0
Consistent-Strict	19	47
Consistent-Loose	17	61
Default	12	54
Semantic-Strict	10	68
Semantic-Loose	10	68

ate all possible TOSCA models and assert that they are as expected by comparing them to already existing TOSCA models of the original pruning paper (Stötzner et al., 2023c). The Variability4TOSCA model of the Consistent-Loose Pruning Mode is already available as part of the original pruning paper.

4.3.3 Analysis

In the previous step, we modeled the Variability4TOSCA models and ensured they all generated the same TOSCA models. An overview of the models and their conditions is given in Table 1. When modeling the webshop application using the Manual Pruning Mode, no conditions are generated. Thus, 52 manual conditions are required. For example, the virtual machine has a condition assigned to check if a static deployment is required. Moreover, since the models are executable, they also have properties, e.g., credentials for GCP or OpenStack.

When using the Consistent-Strict Pruning Mode, the number of manual conditions significantly declines from 52 to 19. This is mainly due to conditional elements with many properties, e.g., the DBMS hosted on GCP CloudSQL. The modeler must no longer manually ensure that these properties are only present when their container is present. In total, 47 pruning conditions are generated.

We can observe the monotonic order of the Pruning Modes in Table 1. The number of manual conditions reduces further when using the Default Pruning Mode, Semantic-Strict Pruning Mode, and Semantic-Loose Pruning Mode. This order is not strictly monotonic, as we can see in our case study, since the Semantic-Strict and Semantic Loose Pruning Mode both generate the same Conditions-Enriched Variability4TOSCA model.

The Default Pruning Mode is a special case that can not be directly placed into order. It generates more conditions than the Consistent-Strict Pruning Mode and fewer conditions than the Semantic-Strict Pruning Mode. However, no general assumption exists on how it behaves considering the Consistent-Loose Pruning Mode. In our case study, it requires fewer manual conditions than the Consistent-Loose Pruning Mode but also generates fewer conditions. The reason is that the Default Pruning Mode does not generate all the conditions that the Consistent-Loose Pruning Mode generates. However, the Semantic-Strict Pruning Mode does generate all the conditions that the Default Pruning Mode generates.

It is also noteworthy that the numbers of manual and generated conditions do not add up to the same value. This is because the Condition Enricher might generate redundant conditions, e.g., a condition at the deployment artifact of the shop component checking for the presence of the shop component, which, however, is always present.

Considering generalization, the numbers in Table 1 are specific to the webshop application. However, in general, our analysis of the modes holds since the Condition Enricher generates pruning conditions based on rules considering the underlying EDMM and, therefore, are generic (Stötzner et al., 2023c).

4.3.4 Deployment

The Variability4TOSCA models of our case study can be executed to deploy the desired variant of the webshop. Since they all generate the same TOSCA models, we can choose any of them. We already described the elastic deployment of this webshop on GCP using the Semantic-Loose Pruning Mode in the original pruning paper (Stötzner et al., 2023c). We complement this work and deploy the static variant using the Consistent-Loose Pruning Mode. Since we now take the operator role, the chosen mode does not affect us.

First, we import the Variability4TOSCA model into Vintner. Then, we specify that a medium-sized static deployment is required. Vintner then generates a TOSCA model and executes it using the TOSCA orchestrator xOpera. As a result, the webshop is deployed on OpenStack as desired.

5 RELATED WORK

In the following, we discuss related works in the domain of variability management with a focus on structural models. To summarize, they propose different concepts to manage variability, addressing their specific problems. However, none of them propose different modes of resolving variability to address different requirements of the modeler, considering aspects such as loss of control and masked errors.

Product line engineering is a concept for managing the variability of software (Pohl et al., 2005; Pohl and Metzger, 2018) that is also used for structural models (Groher and Voelter, 2007; Voelter and Groher, 2007). In this concept, a customer makes a feature configuration, which is then used as input for a generator to derive a customer-tailored product by removing elements from the product whose conditions do not hold. Overall, the Variable Deployment Modeling Method uses product line engineering concepts, e.g., assigning conditions to elements, and, therefore, provides a product line for Essential Deployment Models while focusing on modeling reusable assets.

There exists a lot of research in the domain of using product line engineering and UML (Ziadi et al., 2004; Clauß and Jena, 2001; Junior et al., 2010; Korherr and List, 2007; Dobrica and Niemelä, 2008, 2007; Sun et al., 2010). Typically, UML stereotypes are used to model variability in UML models while UML OCL constraints are used to restrict further allowed variants. Such UML OCL constraints can be used to define pruning conditions. However, these works do not propose different modes of resolving variability to address different requirements.

Czarnecki and Antkiewicz (2005) propose an approach for managing the variability of models based on manual and default conditions. They do not discuss different modes of resolving variability but propose post-processing the model to patch or simplify it after evaluating conditions, e.g., to close the flow between two UML activities. In contrast, pruning pre-processes models before conditions are evaluated and assigns additional conditions to elements. Węsowski (2004) also discuss post-processing derived models, e.g., to remove unused state machines of a state chart. Such post-processing methods for deployment models (Harzenetter et al., 2020; Soldani et al., 2022; Knape, 2015; Soldani et al., 2015; Hirmer et al., 2014) can be combined with our method.

6 CONCLUSION

Automatically removing any inconsistent or semantically incorrect element from the deployment might lead to the unexpected removal of elements and might mask modeling errors. To understand the trade-off between manual modeling and pruning, we defined Pruning Modes. These modes systematically describe different degrees of pruning and the implications of giving up control to increase the number of generated conditions. Hence, we provide guidelines for practitioners on when to use which degree of pruning.

With an increased number of generated conditions, the risk of unexpected removal of elements and, therefore, masked modeling errors rises. Depending on the requirements or experience of the modeler, different Pruning Modes are appropriate. In our opinion, the risk of unexpected removal of elements due to consistency issues is low. Therefore, we generally recommend using the Consistent-Loose Pruning Mode instead of the Manual Pruning Mode due to the expected high reduction of manual conditions. However, if correctly used, the Semantic-Loose Pruning Mode is the most effective Pruning Mode.

In future work, we plan to develop a testing framework in which a modeler can define test cases to validate that a Variable Deployment Model is resolved as expected. We also plan a user study to evaluate the cognitive load when using different Pruning Modes.

ACKNOWLEDGEMENTS

This publication was partially funded by the German Federal Ministry for Economic Affairs and Climate Action (BMWK) as part of the Software-Defined Car (SofDCar) project (19S21002).

REFERENCES

Bergmayr, A., Breitenbücher, U., Ferry, N., Rossini, A., Solberg, A., Wimmer, M., Kappel, G., and Leymann, F. (2018). A Systematic Review of Cloud Modeling Languages. *ACM Computing Surveys (CSUR)*, 51(1):1–38.

Brogi, A., Canciani, A., and Soldani, J. (2018). Fault-aware management protocols for multi-component applications. *Journal of Systems and Software*, 139:189–210.

Clauß, M. and Jena, I. (2001). Modeling variability with UML. In *GCSE 2001 Young Researchers Workshop*. Springer.

Czarnecki, K. and Antkiewicz, M. (2005). Mapping Features to Models: A Template Approach Based on Superimposed Variants. In *Generative Programming and Component Engineering*, pages 422–437, Berlin, Heidelberg. Springer.

Dobrica, L. and Niemelä, E. (2007). Modeling Variability in the Software Product Line Architecture of Distributed Services. In *Proceedings of the 2007 International Conference on Software Engineering Research & Practice, SERP*, pages 269–275. CSREA Press.

Dobrica, L. and Niemelä, E. (2008). A UML-Based Variability Specification For Product Line Architecture Views. In *Proceedings of the Third International Conference on Software and Data Technologies*. SciTePress.

Endres, C., Breitenbücher, U., Falkenthal, M., Kopp, O., Leymann, F., and Wettinger, J. (2017). Declarative vs. Imperative: Two Modeling Patterns for the Automated Deployment of Applications. In *Proceedings of the 9th International Conference on Pervasive Patterns and*

Applications (PATTERNS 2017), pages 22–27. Xpert Publishing Services.

Groher, I. and Voelter, M. (2007). Expressing Feature-Based Variability in Structural Models. In *Workshop on Managing Variability for Software Product Lines*.

Harzenetter, L., Breitenbücher, U., Falkenthal, M., Guth, J., and Leymann, F. (2020). Pattern-based Deployment Models Revisited: Automated Pattern-driven Deployment Configuration. In *Proceedings of the Twelfth International Conference on Pervasive Patterns and Applications (PATTERNS 2020)*, pages 40–49. Xpert Publishing Services.

Hirmer, P., Breitenbücher, U., Binz, T., and Leymann, F. (2014). Automatic Topology Completion of TOSCA-based Cloud Applications. In *Proceedings des CloudCycle14 Workshops auf der 44. Jahrestagung der Gesellschaft für Informatik e.V. (GI)*, volume 232 of *LNI*, pages 247–258, Bonn. Gesellschaft für Informatik e.V. (GI).

Junior, E. A. O., de Souza Gimenes, I. M., and Maldonado, J. C. (2010). Systematic Management of Variability in UML-based Software Product Lines. *J. Univers. Comput. Sci.*, 16(17):2374–2393.

Knape, S. (2015). Dynamic Automated Selection and Deployment of Software Components within a Heterogeneous Multi-Platform Environment. Master's thesis, Utrecht University.

Korherr, B. and List, B. (2007). A UML 2 Profile for Variability Models and their Dependency to Business Processes. In *18th International Workshop on Database and Expert Systems Applications (DEXA 2007)*, pages 829–834.

OASIS (2020). *TOSCA Simple Profile in YAML Version 1.3*. Organization for the Advancement of Structured Information Standards (OASIS).

Oppenheimer, D. (2003). The importance of understanding distributed system configuration. In *Proceedings of the 2003 Conference on Human Factors in Computer Systems workshop*.

Oppenheimer, D., Ganapathi, A., and Patterson, D. A. (2003). Why do internet services fail, and what can be done about it? In *4th Usenix Symposium on Internet Technologies and Systems (USITS 03)*.

Pohl, K., Böckle, G., and van der Linden, F. (2005). *Software Product Line Engineering*. Springer Berlin Heidelberg.

Pohl, K. and Metzger, A. (2018). *Software Product Lines*, pages 185–201. Springer International Publishing, Cham.

Soldani, J., Binz, T., Breitenbücher, U., Leymann, F., and Brogi, A. (2015). ToscaMart: A method for adapting and reusing cloud applications. *Journal of Systems and Software*, 113:395–406.

Soldani, J., Breitenbücher, U., Brogi, A., Frioli, L., Leymann, F., and Wurster, M. (2022). Tailoring Technology-Agnostic Deployment Models to Production-Ready Deployment Technologies. In *Cloud Computing and Services Science*. Springer.

Stötzner, M., Becker, S., Breitenbücher, U., Kálmán, K., and Leymann, F. (2022). Modeling Different Deployment Variants of a Composite Application in a Single Declarative Deployment Model. *Algorithms*, 15(10):1–25.

Stötzner, M., Breitenbücher, U., Pesl, R. D., and Becker, S. (2023a). Managing the Variability of Component Implementations and Their Deployment Configurations Across Heterogeneous Deployment Technologies. In *Cooperative Information Systems*, pages 61–78, Cham. Springer Nature Switzerland.

Stötzner, M., Breitenbücher, U., Pesl, R. D., and Becker, S. (2023b). Using Variability4TOSCA and OpenTOSCA Vintner for Holistically Managing Deployment Variability. In *Proceedings of the Demonstration Track at International Conference on Cooperative Information Systems 2023*, volume 3552 of *CEUR Workshop Proceedings*, pages 36–40. CEUR-WS.org.

Stötzner, M., Klinaku, F., Pesl, R. D., and Becker, S. (2023c). Enhancing Deployment Variability Management by Pruning Elements in Deployment Models. In *Proceedings of the 16th International Conference on Utility and Cloud Computing (UCC 2023)*. ACM.

Sun, C., Rossing, R., Sinnema, M., Bulanov, P., and Aiello, M. (2010). Modeling and managing the variability of Web service-based systems. *Journal of Systems and Software*, 83(3):502–516.

Voelter, M. and Groher, I. (2007). Product Line Implementation using Aspect-Oriented and Model-Driven Software Development. In *11th International Software Product Line Conference (SPLC 2007)*, pages 233–242. IEEE.

Węsowski, A. (2004). Automatic Generation of Program Families by Model Restrictions. In *Software Product Lines*, pages 73–89. Springer.

Wurster, M., Breitenbücher, U., Brogi, A., Diez, F., Leymann, F., Soldani, J., and Wild, K. (2021). Automating the Deployment of Distributed Applications by Combining Multiple Deployment Technologies. In *Proceedings of the 11th International Conference on Cloud Computing and Services Science*. SciTePress.

Wurster, M., Breitenbücher, U., Falkenthal, M., Krieger, C., Leymann, F., Saatkamp, K., and Soldani, J. (2019). The Essential Deployment Metamodel: A Systematic Review of Deployment Automation Technologies. *SICS Software-Intensive Cyber-Physical Systems*, 35:63–75.

Wurster, M., Breitenbücher, U., Harzenetter, L., Leymann, F., Soldani, J., and Yussupov, V. (2020). TOSCA Light: Bridging the Gap between the TOSCA Specification and Production-ready Deployment Technologies. In *Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER 2020)*, pages 216–226. SciTePress.

Yussupov, V., Soldani, J., Breitenbücher, U., Brogi, A., and Leymann, F. (2021). From Serverful to Serverless: A Spectrum of Patterns for Hosting Application Components. In *Proceedings of the 11th International Conference on Cloud Computing and Services Science (CLOSER 2021)*, pages 268–279. SciTePress.

Ziadi, T., Hélouët, L., and Jézéquel, J.-M. (2004). Towards a UML Profile for Software Product Lines. In *Software Product-Family Engineering*. Springer.