Eclipse Winery

Contributors to the Eclipse Foundation

Jul 06, 2022

CONTENTS:

Eclipse Winery is a web-based environment to graphically model TOSCA topologies and plans managing these topologies. The environment includes a type and template management component to offer creation and modification of all elements defined in the TOSCA specification. All information is stored in a repository, which allows importing and exporting using the TOSCA packaging format.

Are you tired of maintaining your TOSCA files manually by just using a text editor?

Use Eclipse Winery as an usability layer on top to maintain your TOSCA files (XML or YAML) in a graphical and intuitive user interface. Eclipse Winery provides a graphical web-editor with which you can create and maintain all TOSCA entities. Thereby, Eclipse Winery stores all TOSCA entities in a defined folder structure that fosters the reusability of TOSCA types. Eclipse Winery validates and stores all TOSCA entities in the syntax defined in the standard.

Further, the graph-based representation of TOSCA topologies in Eclipse Winery provides a quick overview of the entire system and offers a communication basis for the cooperation with other parties. It therefore offers a quicker introduction to modeling with TOSCA and provides newcomers with necessary guidelines.

This is the main documentation of Eclipse Winery.

Organizational information is provided at the eclipse.org page.

Demo	Video
Getting Started	*User Guide*
Licence	EPL-2.0 OR Apache-2.0
Maintainer(s)	Eclipse Winery Contributors

CHAPTER

ONE

USER GUIDE

Eclipse Winery is a web-based environment to graphically model *OASIS TOSCA* topologies and plans managing these topologies. It is an Eclipse project and thus support is available through its project page. Winery is also part of the OpenTOSCA ecosystem where more information is available at opentosca.org. For more information on TOSCA see our *TOSCA information page*.

1.1 Getting Started

1.1.1 Launching with Docker

Note: It is recommended that your host or virtual machine has at least 2GB of memory.

Open a command prompt and execute the following command:

```
docker run -it -p 8080:8080 \
  -e PUBLIC_HOSTNAME=localhost \
  -e WINERY_FEATURE_RADON=true \
  -e WINERY_REPOSITORY_PROVIDER=yaml \
  -e WINERY_REPOSITORY_URL=https://github.com/radon-h2020/radon-particles \
  opentosca/radon-gmt
```

Launch a browser: http://localhost:8080.

Note: To start Eclipse Winery based on an TOSCA XML repository layout, use the following command:

```
docker run -it -p 8080:8080 \
  -e PUBLIC_HOSTNAME=localhost \
  -e WINERY_REPOSITORY_URL=https://github.com/OpenTOSCA/tosca-definitions-public \
  opentosca/winery
```

Note: Make sure you regularly pull the latest images:

```
docker pull opentosca/radon-gmt:latest
# or
docker pull opentosca/winery:latest
```

Use a custom TOSCA model repository

Problem: You want to use an existing TOSCA model repository that you have cloned, e.g., to add new or adapt existing TOSCA types and blueprints in this Git repository.

Please follow the next instructions to mount an existing TOSCA model repository into the Eclipse Winery container. This is useful if you want to save your modeling changes onto your Docker host machine.

Clone or create git repository on your local filesystem, e.g., by cloning https://github.com/radon-h2020/radon-particles.

Open a command prompt and execute the following command:

> **Warning:** Replace `<path_on_your_host>` with the respective dirctory path on your host system.

```
docker run -it -p 8080:8080 \
  -e PUBLIC_HOSTNAME=localhost \
  -e WINERY_FEATURE_RADON=true \
  -e WINERY_REPOSITORY_PROVIDER=yaml \
  -v <path_on_your_host>:/var/repository \
  -u `id -u` \
  opentosca/radon-gmt
```

Launch a browser: http://localhost:8080.

Any change (create service template, modify or create node types) will be reflected on your host machine. You are now able to commit your changes and push them to your own Git remote (e.g., using `git push` from a command-prompt).

Note: To start Eclipse Winery based on an TOSCA XML repository layout, use the following command:

```
docker run -it -p 8080:8080 \
  -e PUBLIC_HOSTNAME=localhost \
  -v <path_on_your_host>:/var/repository \
  opentosca/winery
```

Model and version your applications and company-specific types only

Problem: You want to model applications based on actively maintained TOSCA type repositories but you want to version/save only your own application blueprints and company-specific types inside your (private) Git repository (GitHub, GitLab).

You are able to start Eclipse Winery in a so-called "multi-repository" setup where you can add several TOSCA type repositories that you can use for your application models. However, with this setup, Eclipse Winery creates a specific "workspace" that only contains your company-specific types and application blueprints (separated by namespace). You can then mount the created "workspace" to save your modeling results to your own Git remote.

Open a command prompt and execute the following command:

> **Warning:** Replace `<path_on_your_host>` with a respective dirctory path on your host system.

```
docker run -it -p 8080:8080 \
  -e PUBLIC_HOSTNAME=localhost \
  -e WINERY_FEATURE_RADON=true \
  -e WINERY_REPOSITORY_PROVIDER=yaml \
  -e WINERY_DEPENDENT_REPOSITORIES="[ { \"name\" : \"RADON Particles\", \"url\" : \
↪"https://github.com/radon-h2020/radon-particles.git\", \"branch\" : \"master\" } ]"␣
↪\
  -v <path_on_your_host>:/var/repository \
  -u `id -u` \
  opentosca/radon-gmt
```

Your created TOSCA service templates or company-specific TOSCA node types will be stored on your host machine. You are now able to commit your changes and push them to your own Git remote (e.g., using `git push` from a command-prompt).

1.1.2 Launching with Docker Compose

Note: It is recommended that your host or virtual machine has at least 2GB of memory.

Install Docker and Docker Compose.

Clone the repository:

```
git clone https://github.com/eclipse/winery
cd winery/deploy/compose
```

[Optional] Adapt the Docker Compose configuration to your needs, e.g., to mount a local TOSCA model repository.

Start Winery:

```
docker-compose up
```

Launch a browser: http://localhost:8080.

1.2 Modeling with Winery

Launch a browser and navigate to http://localhost:8080.

1.2.1 Modeling an Application

Eclipse Winery starts in the *Service Template* view. In this view, users can create new TOSCA service template or maintain existing ones.

To create a new TOSCA service template click on *Add new*. In the "Add new" pop up you can specify your template's name, enable/disable versioning, and specify the namespace to be used. For example, you may choose a namespace like `com.example.blueprints` to logically group your TOSCA service templates.

Warning: Do not use spaces in your service template name. Use _ or - to separate names.

In the *Service Template Detail* view you can add some readme text and assign a respective license. Further, to compose your application open the *Topology Modeler* by `Topology Template > Open Editor`.

Model Node Templates

In the editor, you can drag and drop existing TOSCA node types to the canvas to define a new TOSCA node template. You can select a modeled node to modify its display name and additional data using the right pane.

You can change properties or add artifacts by enabling the *Properties* or *Artifacts* view in the header bar.

Define Relations Between Node Templates

Relationships in TOSCA (according to TOSCA YAML 1.3) are modeled using matching *Requirements* and *Capabilities* (please refer to the standard to get more detailed information or checkout the *Notes on TOSCA* page).

In the *Topology Modeler*, you can enable the *Requirements & Capabilities* view in the header bar. Then, open the *Requirements* of the source node and the *Capabilities* of the target node. Finally, drag a respective relationship type (e.g., `HostedOn`) from the requirement (e.g., `host`) to a matching capability (e.g., `host`).

1.2.2 Export CSAR

The TOSCA exchange format is a Cloud Service Archive (CSAR). A CSAR is essentially a ZIP file following a certain directory layout and contains all necessary files and template to execute the deployment of the modeled application.

Open the *Service Template* view. Search for your service template and open it. In the *Service Template Detail* view you can click on *Export* either to *Download* the CSAR or to save it to the filesystem (`<repository>/csars` on your host system).

1.3 Node Type Development

Start Eclipse Winery as described in "Use a custom TOSCA model repository" of the Getting Started page. This way, newly created Node Types will be reflected in the filesystem which is mounted into Winery's Docker container.

Before you start, create a new branch:

```
git checkout -b <name>
```

You can push this branch to a Git origin to share your work with others or you could propose a pull-request to the original Git repository.

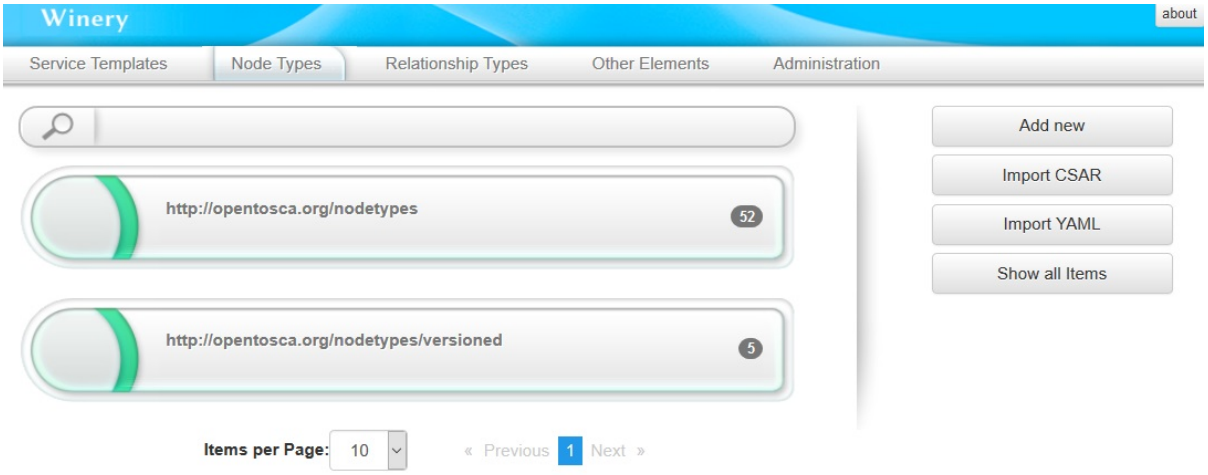
1.4 Modeling based on TOSCA XML (deprecated)

This guide shows an overview of how to model TOSCA node types and service templates using Winery. Before starting this guide, please take a look at *Miscellaneous Notes*.

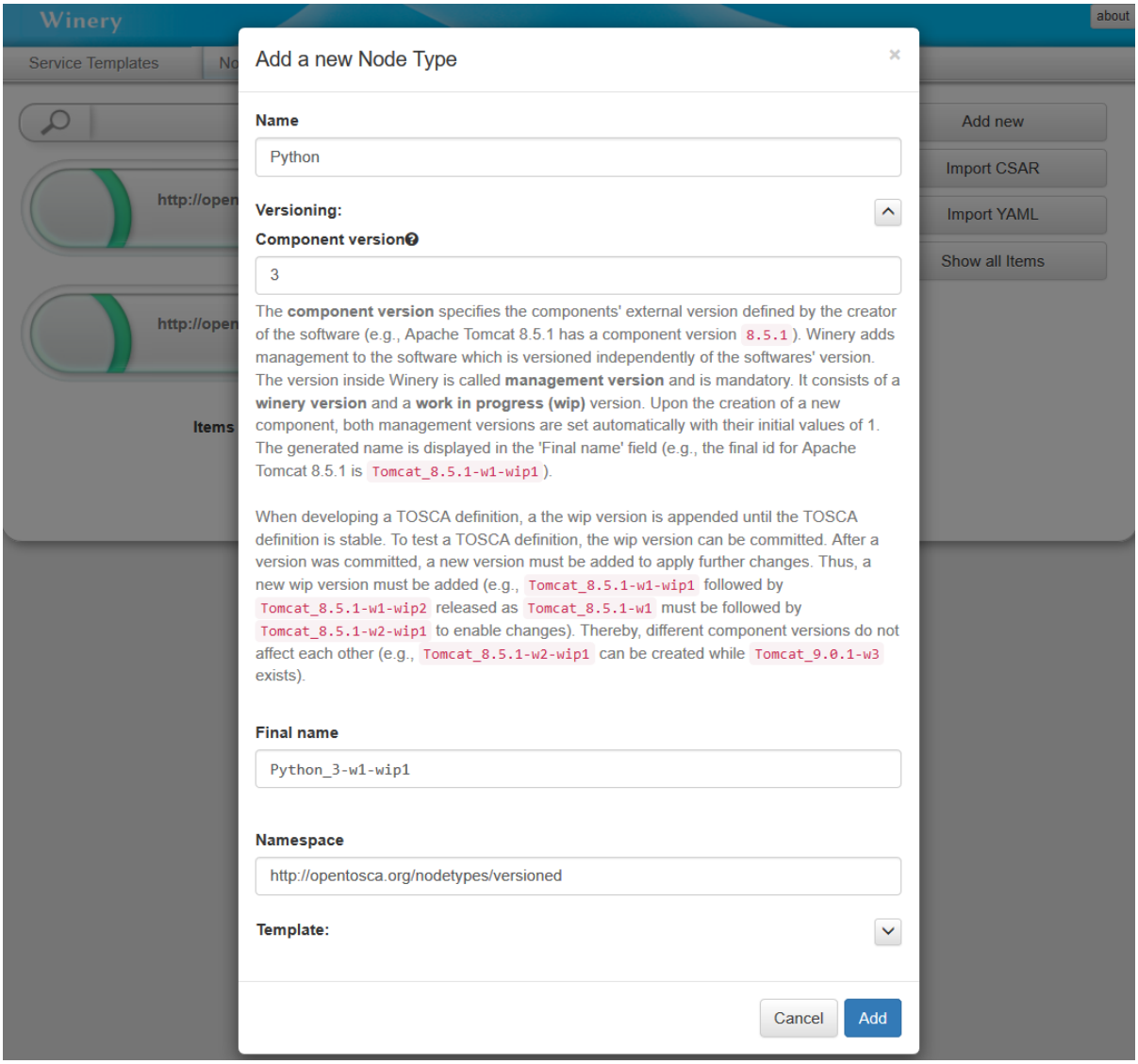
The following shows how to model new node types and how to use them at the modeling of a new service template. In this example, the runtime **Python3** shall be installed on an **Ubuntu 14.04** virtual machine running on an **Openstack** infrastructure. For this, we require three node types. In this example, we model two node types, Python3 and Ubuntu 14.04, and assume that the OpenStack node type was previously modeled.

1.4.1 Creating a new Node Type

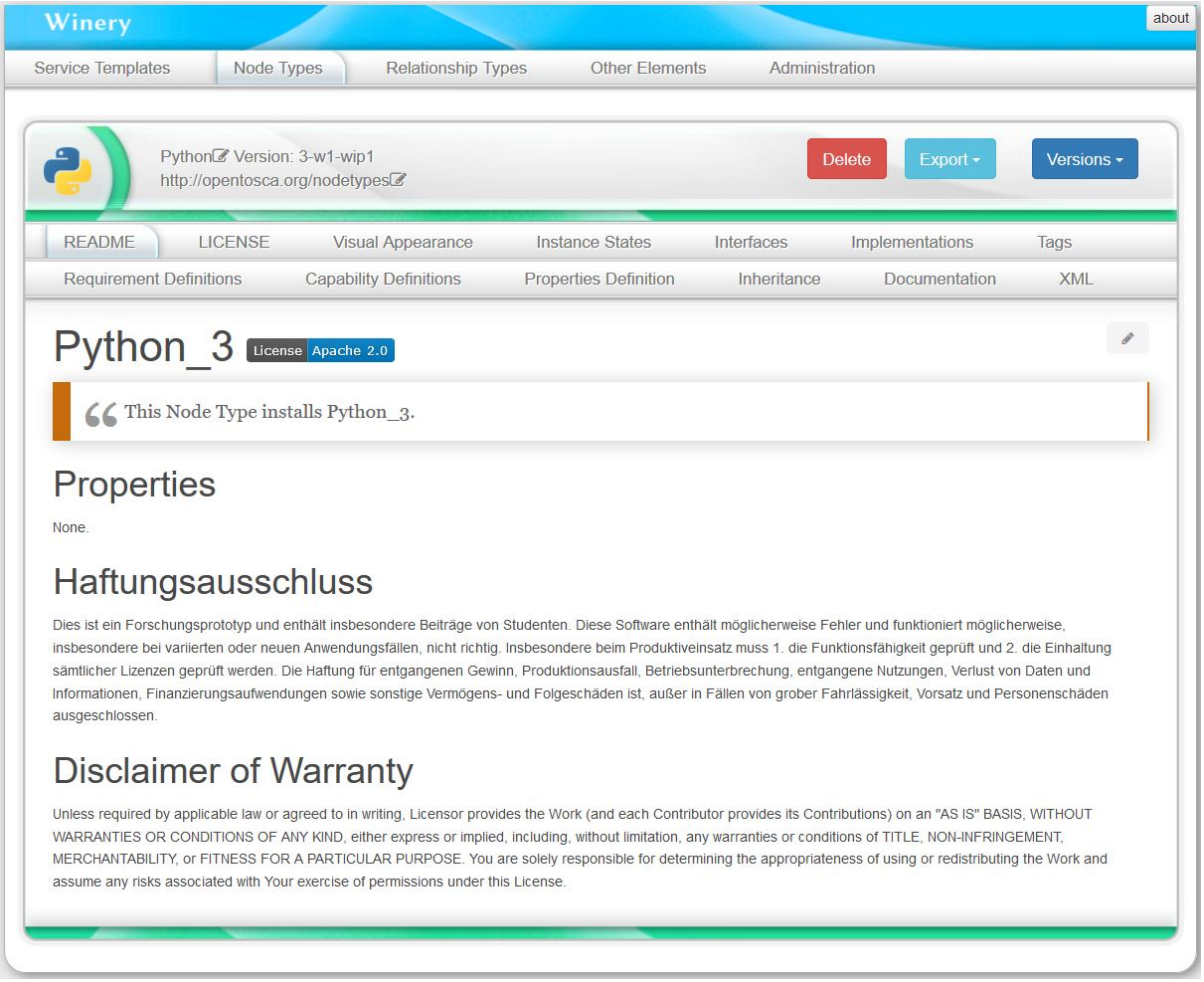
By selecting the tab *Node Types*, a list of available node types is shown. To create a new node type, press the button *Add new*.



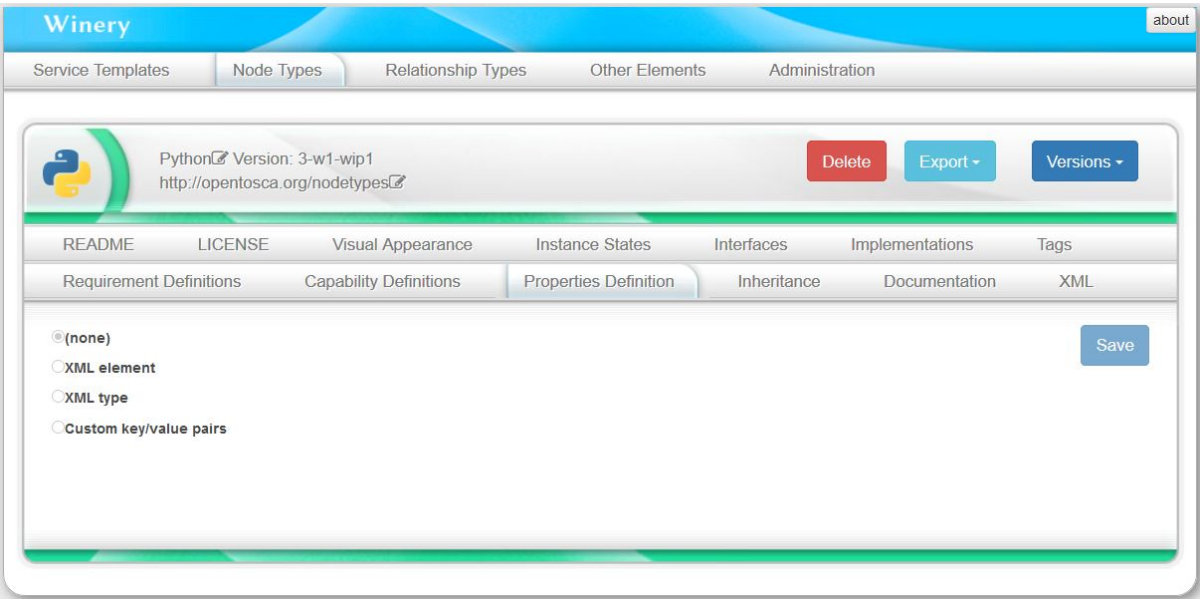
This will open a dialog in which the *Name*, *Component version*, and *Namespace* of the new node type can be configured.



Once the node type is created, it can be further configured through different tabs of its detailed view.



For example, to add properties to the node type, select the tab *Properties Definition*.

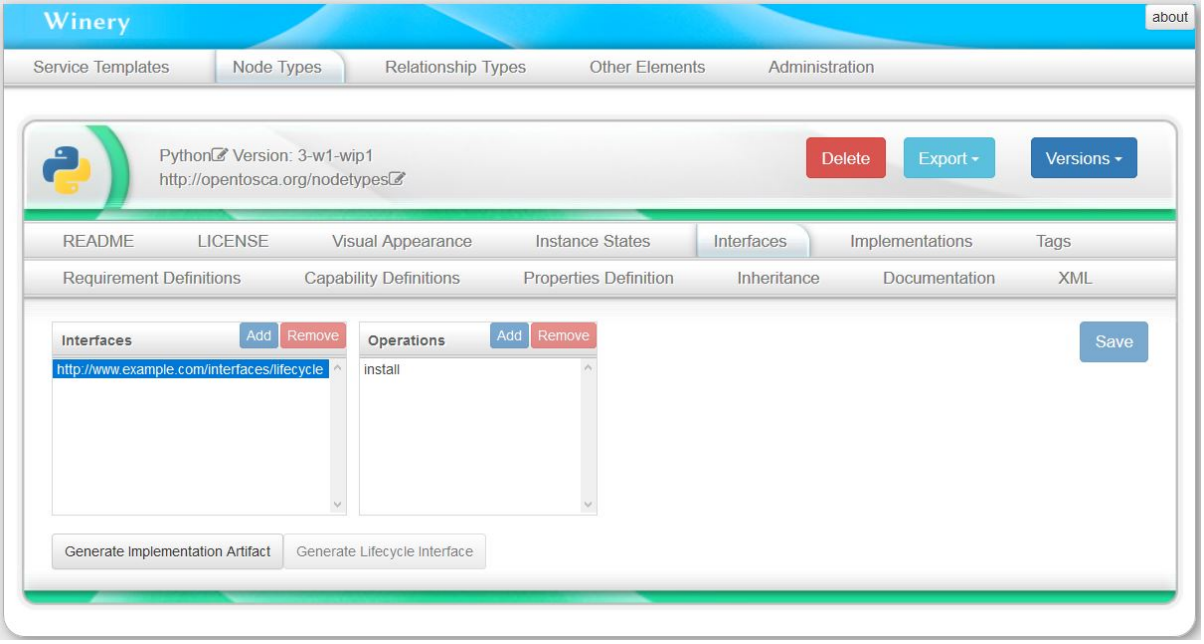


In this example, the Python3 node type does not require any properties.

1.4.2 Modeling the Node Type Interface

To specify what the *Python3* node type should do, we define an *interface* and the *operations* provided by this interface. An interface containing lifecycle operations (install, configure, start, stop, uninstall) can be automatically generated, however, any arbitrary interface can be created.

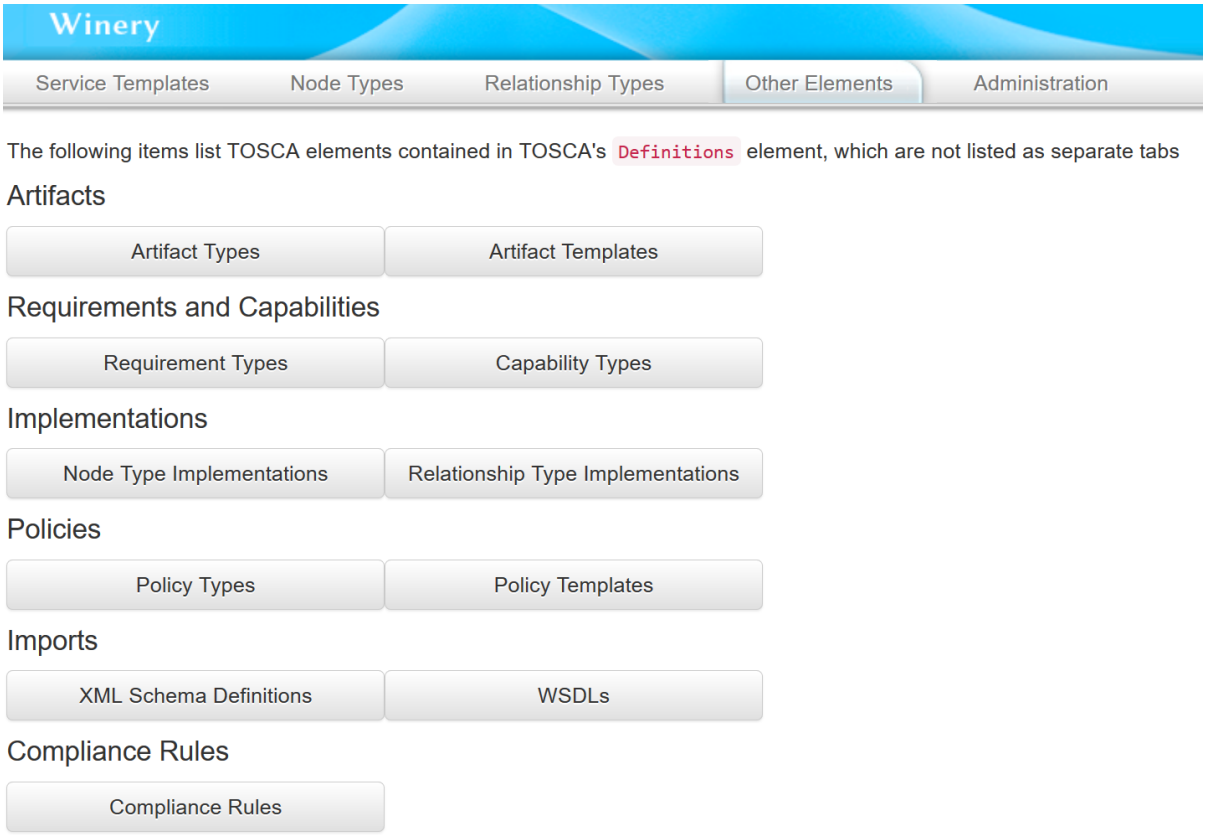
To generate a lifecycle interface, press *Generate Lifecycle Interface* and *Save*. For the node type Python3, we only use the operation *install*.

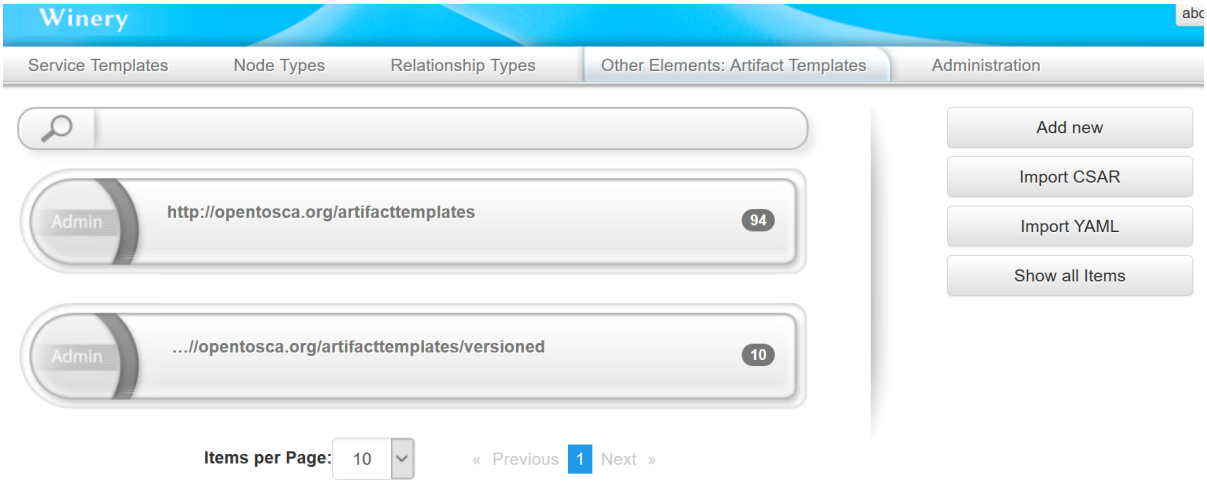


1.4.3 Modeling an Artifact Template for a Node Type Operation

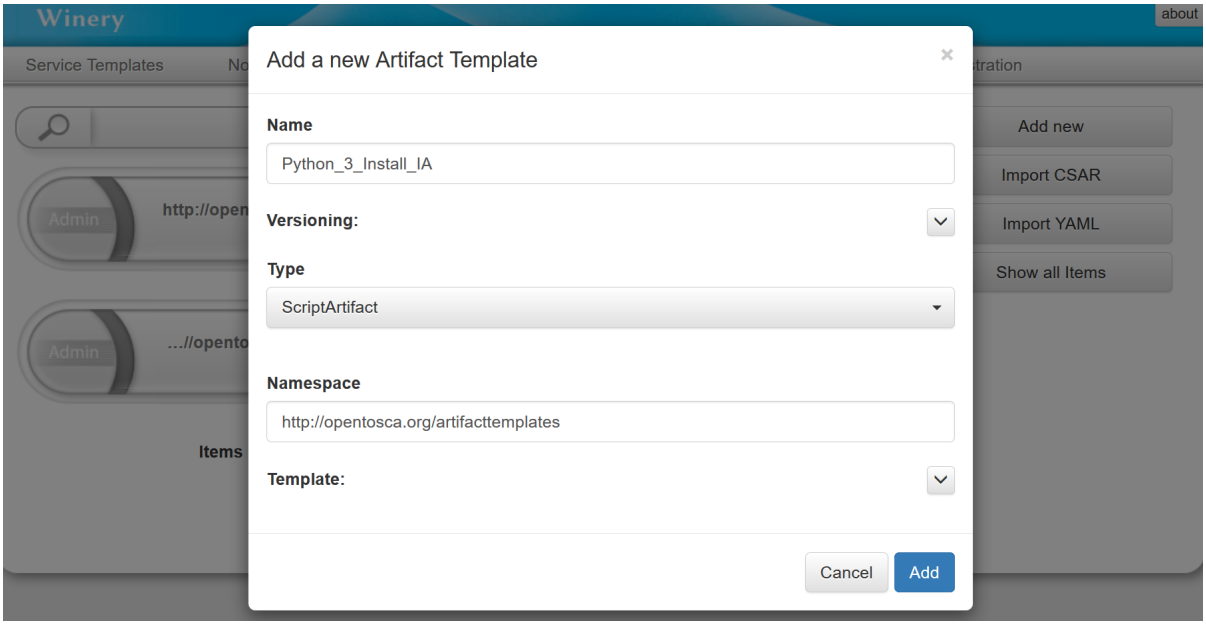
Once the operations of a node type are defined, artifacts (e.g., shell scripts, .war files) implementing these operations need to be modeled. In this example, we have a *shell script* to install Python3 on Ubuntu, which we model as an artifact template.

To create an artifact template, select the tab *Other Elements*, under the category *Artifacts* select the option *Artifact Templates*, and press the button *Add new*.

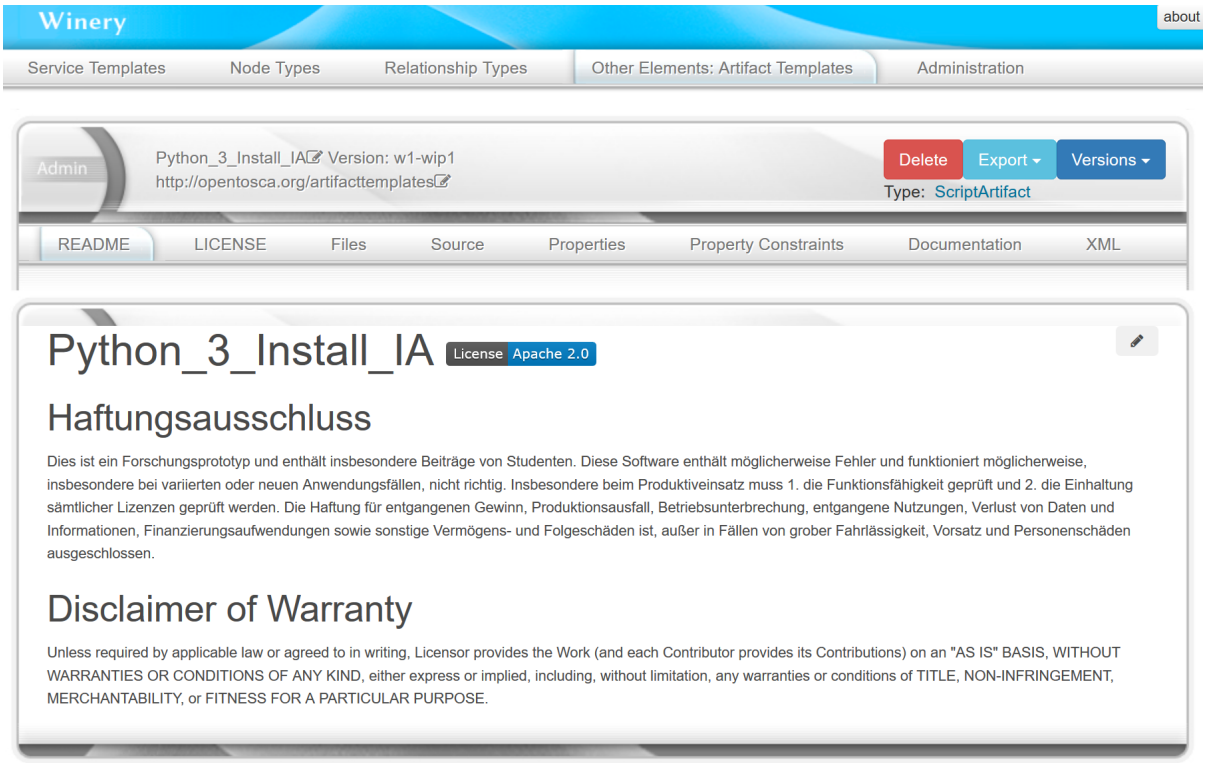




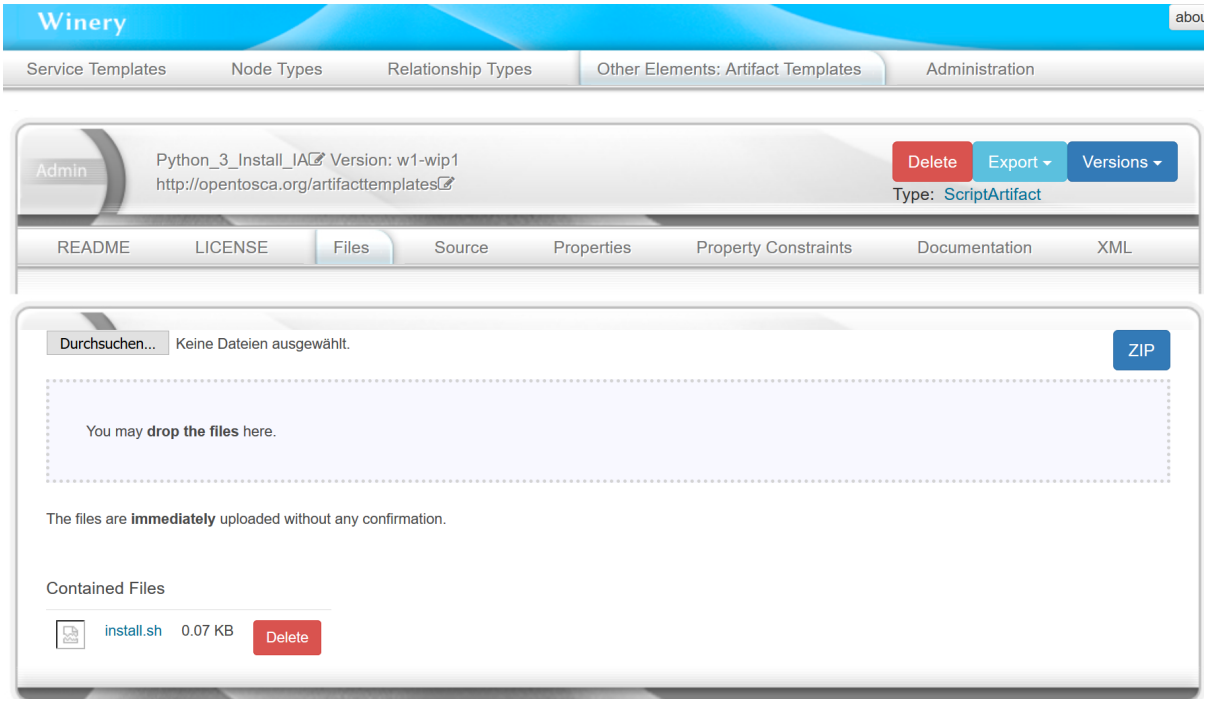
This will open a dialog in which the *Name*, *Versioning*, *Type*, and *Namespace* of the artifact template can be configured. Assuming that some artifact types were previously modeled, choose the type *ScriptArtifact*.



Once the artifact template is created, it can be further configured through different tabs of its detailed view.

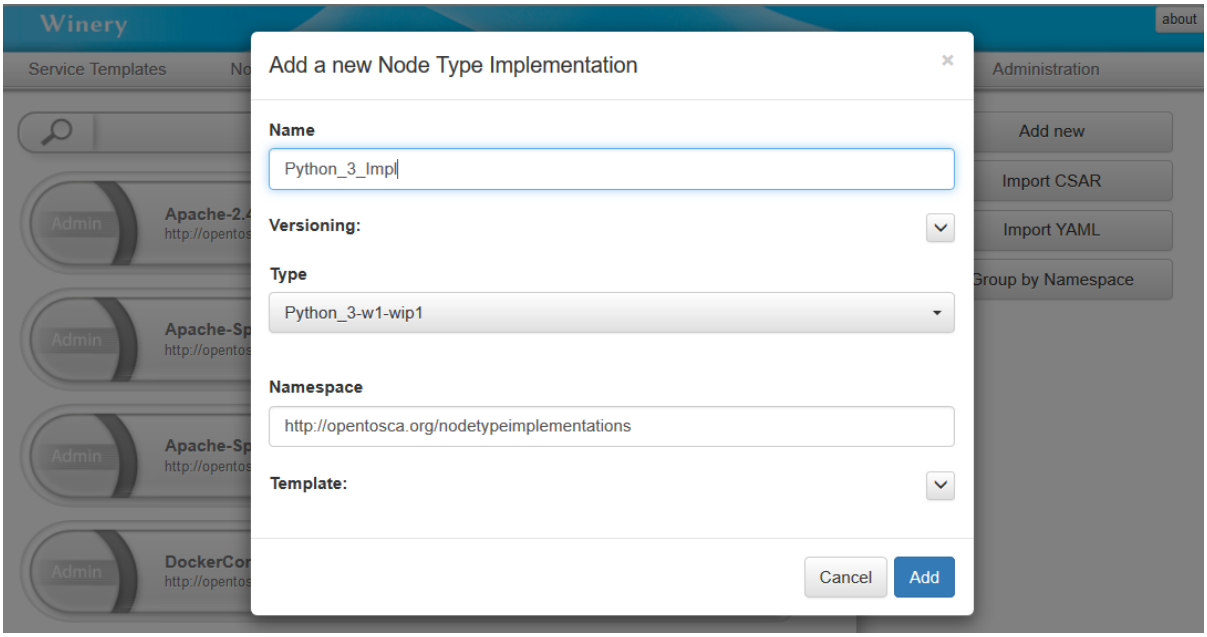


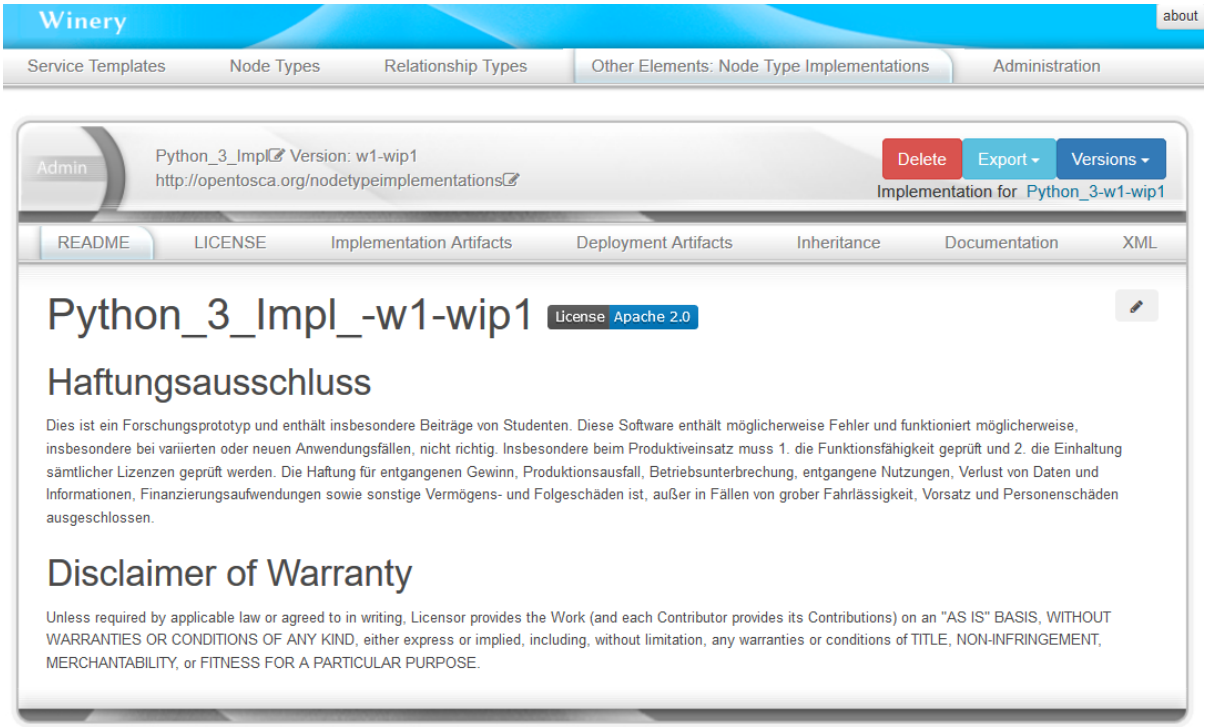
Finally, to load th install script to the artifact template, select the tab *Files*, and drop the file into the drop zone.



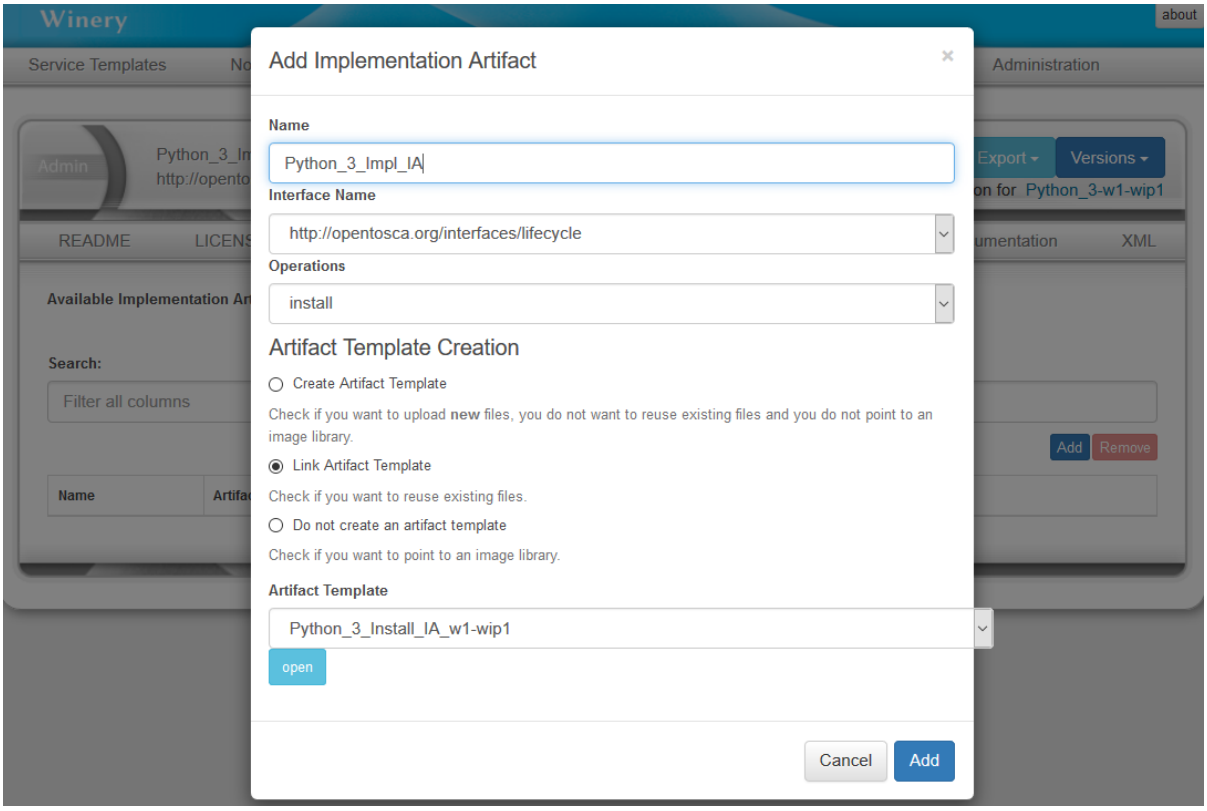
1.4.4 Modeling the Node Type Implementation

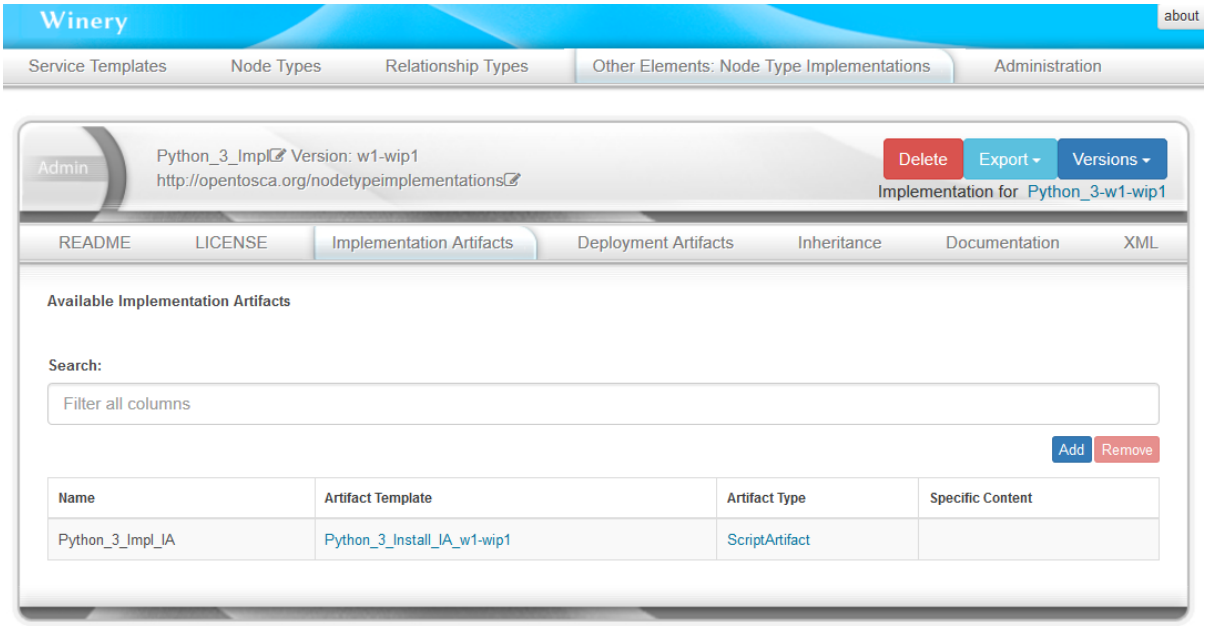
To create a node type implementation, select the tab *Other Elements*, under the the category *Implementations* select the option *Node Type Implementations*, and press the button *Add new*. This will open a dialog in which the *Name*, the corresponding node *type*, and *Namespace* of the node type implementation can be configured. By type, select the node type we created before.





To link the created artifact template to this node type implementation, select the tab *Implementation Artifacts* and press the button *Add*. In the shown dialog, choose the option *Link Artifact Template*, then select the artifact template that was previously created.





1.4.5 Modeling the Ubuntu Node Type

The modeling of the Ubuntu node type is similar to the modeling of the Python3 node type.

Winery

about

Service Templates | Node Types | Relationship Types | Other Elements | Administration

Ubuntu Version: 14.04-w1-wip1
http://opentosca.org/nodetypes

Delete | Export | Versions

README | LICENSE | Visual Appearance | Instance States | Interfaces | Implementations | Tags

Requirement Definitions | Capability Definitions | Properties Definition | Inheritance | Documentation | XML

Ubuntu-14.04

License Apache 2.0

> This node type corresponds to an Ubuntu-14.04 virtual machine.

Properties

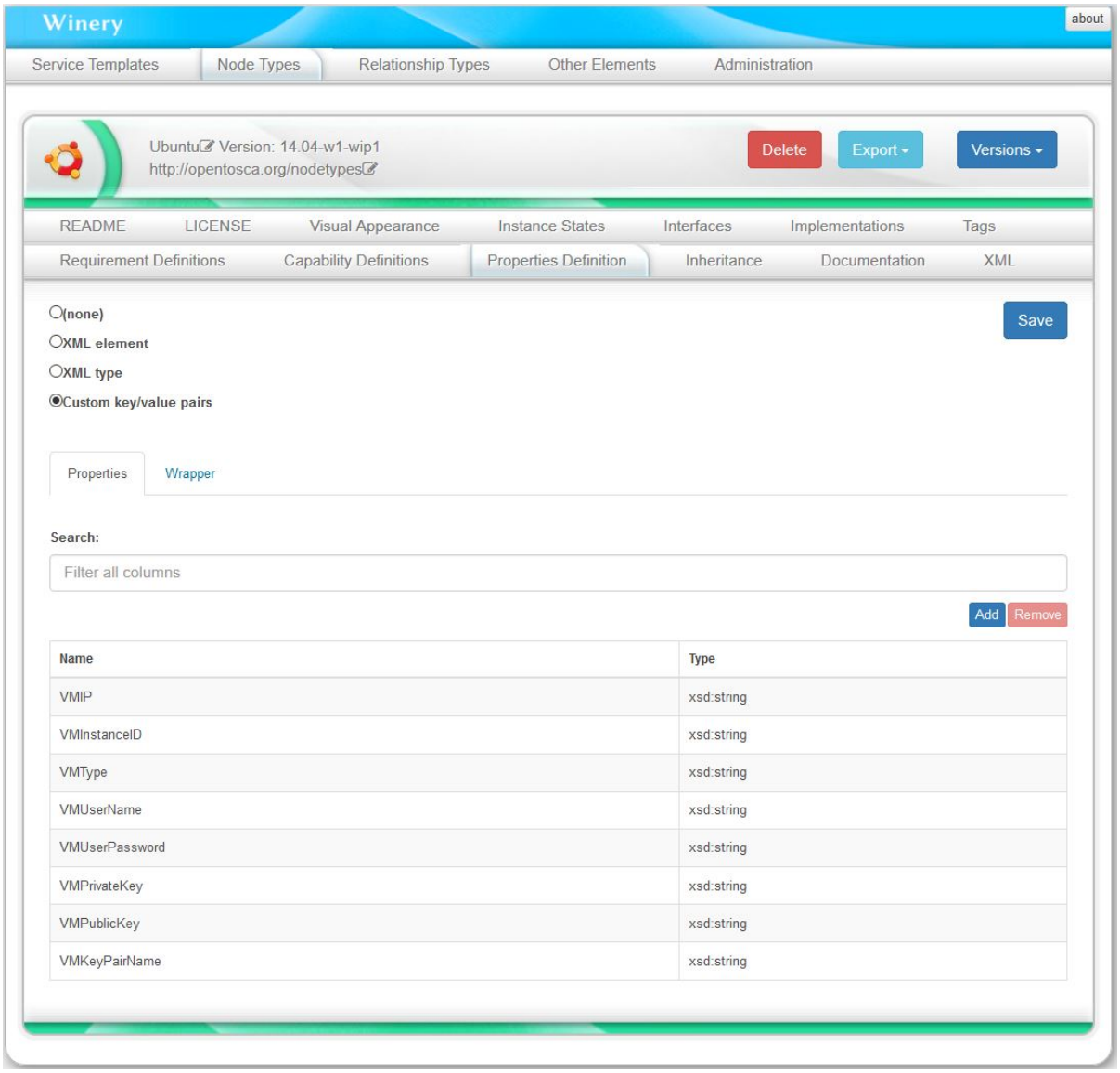
- `VMIP` (optional)
- `VMInstanceID` (optional)
- `VMType`
- `VMUserName`
- `VMUserPassword`
- `VMPrivateKey`
- `VMPublicKey`
- `VMKeyPairName`

Haftungsausschluss

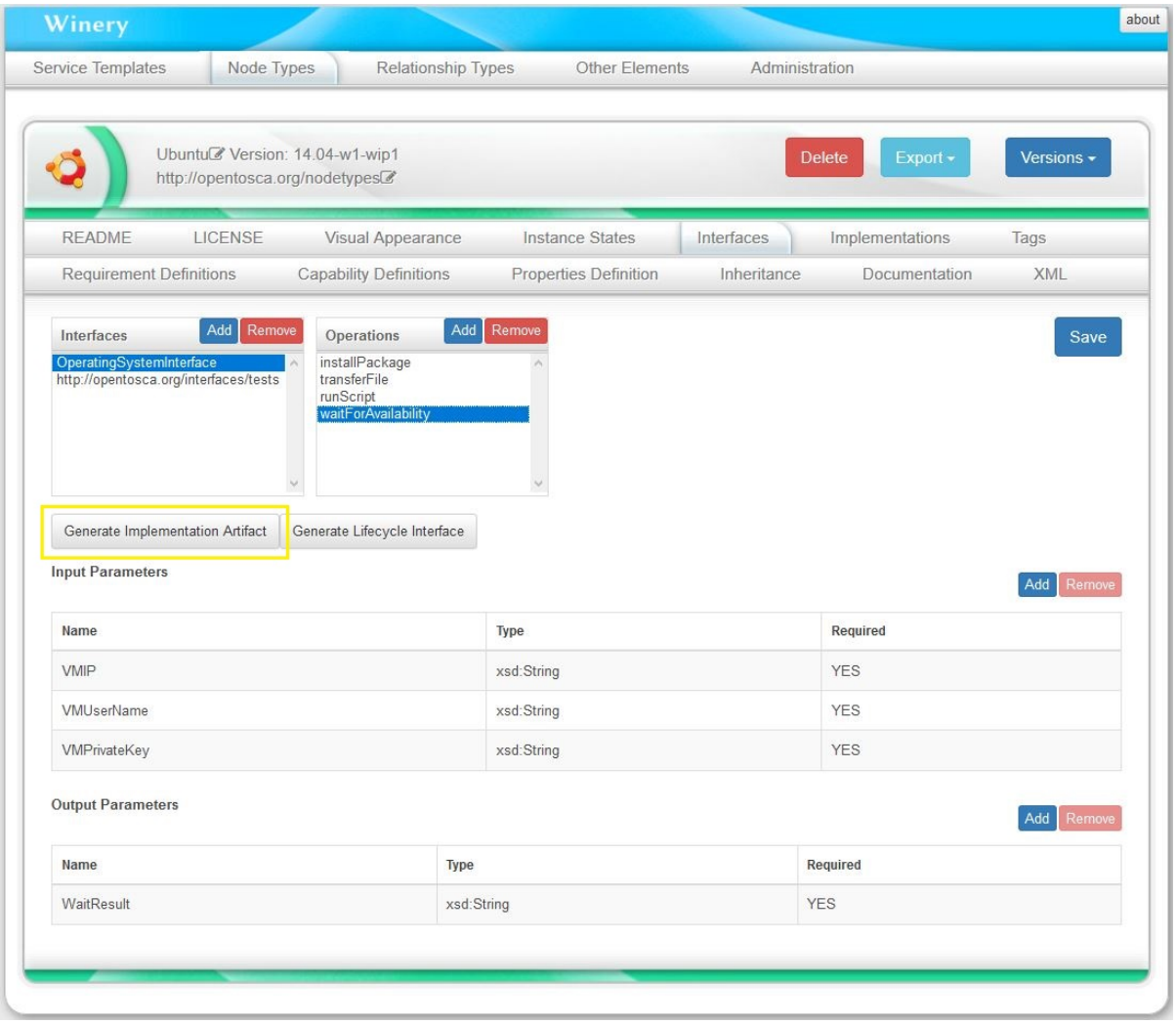
Dies ist ein Forschungsprototyp und enthält insbesondere Beiträge von Studenten. Diese Software enthält möglicherweise Fehler und funktioniert möglicherweise, insbesondere bei variierten oder neuen Anwendungsfällen, nicht richtig. Insbesondere beim Produktiveinsatz muss 1. die Funktionsfähigkeit geprüft und 2. die Einhaltung sämtlicher Lizenzen geprüft werden. Die Haftung für entgangenen Gewinn, Produktionsausfall, Betriebsunterbrechung, entgangene Nutzungen, Verlust von Daten und Informationen, Finanzierungsaufwendungen sowie sonstige Vermögens- und Folgeschäden ist, außer in Fällen von grober Fahrlässigkeit, Vorsatz und Personenschäden ausgeschlossen.

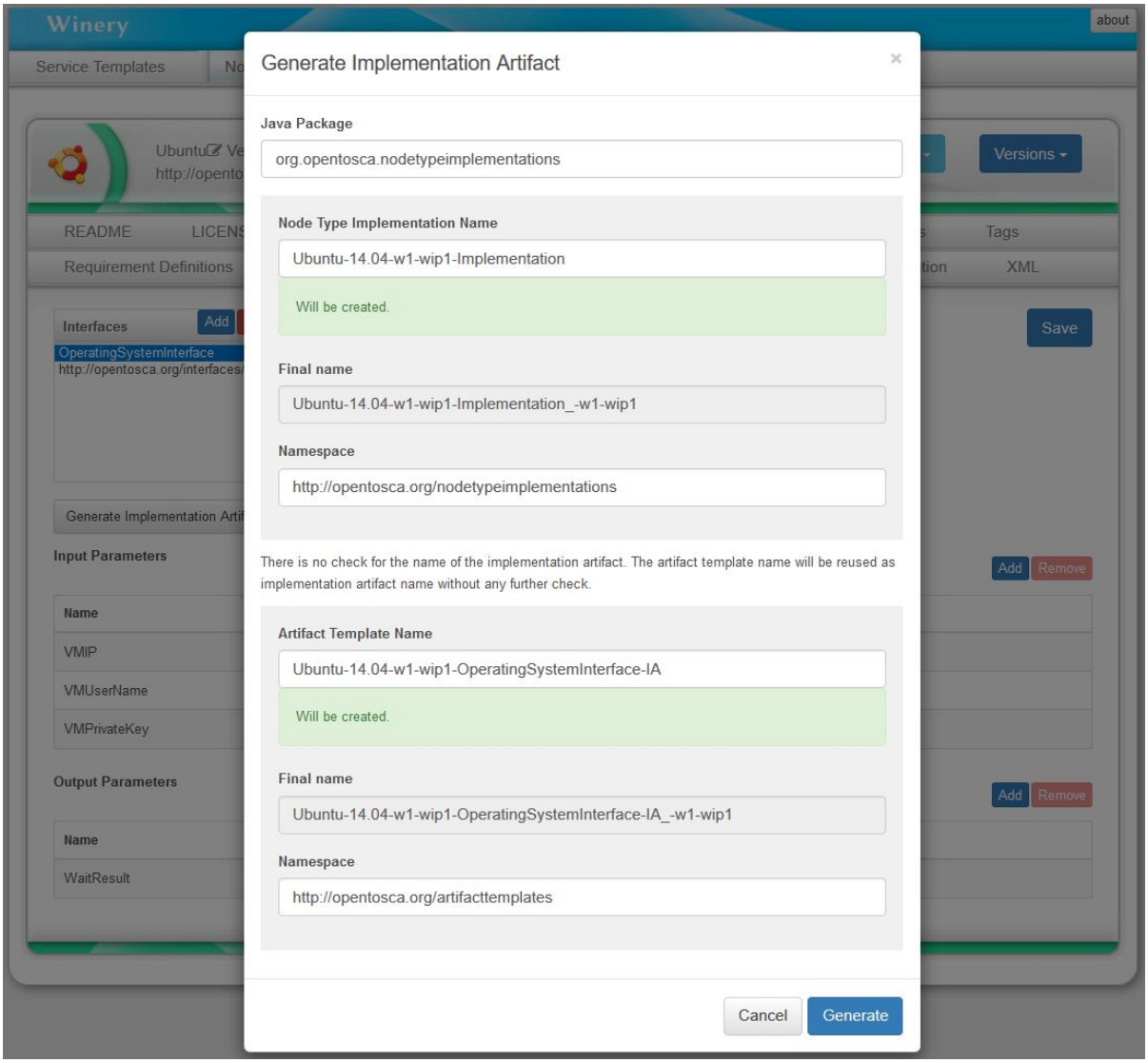
Disclaimer of Warranty

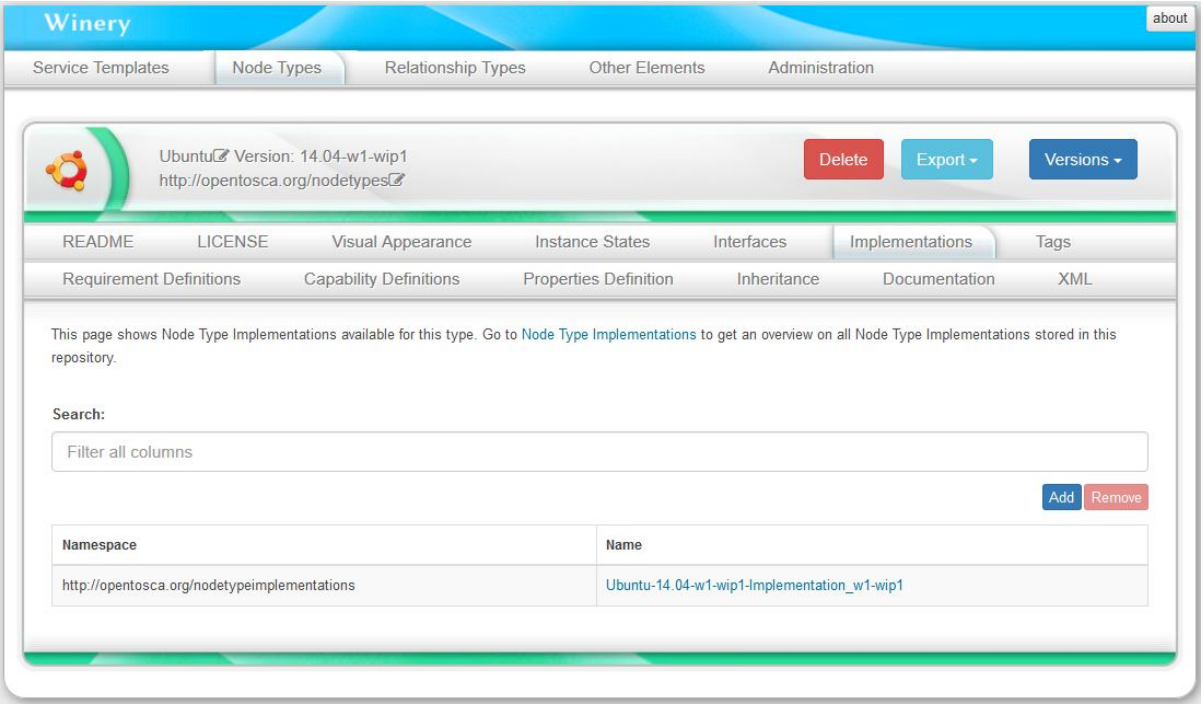
Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

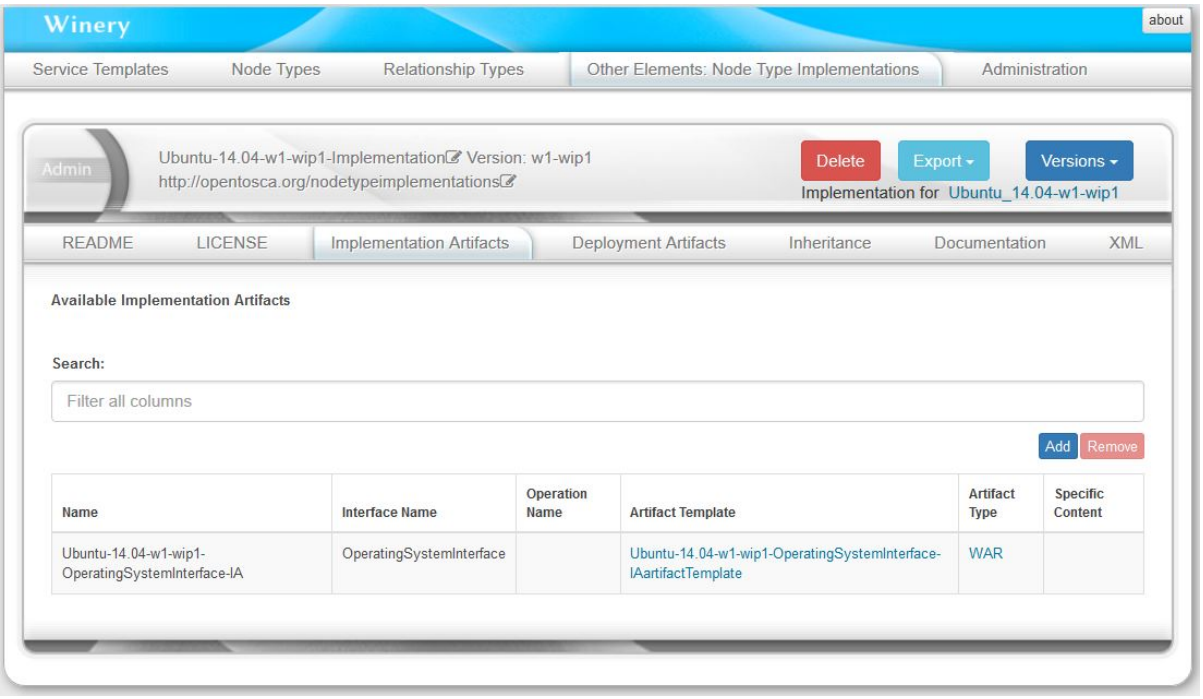


However, the artifact templates for the Ubuntu 14.04 are *.war files* instead of *shell scripts*. In this case, after defining the *interfaces* and *operations* of the Ubuntu node type, we can automatically generate a stub java maven project to build a *.war file* for a defined interface. For this, press *Generate Implementation Artifact*. The node type implementation will be automatically generated as well.


Winery
about
Service Templates
Node Types
Relationship Types
Other Elements
Administration
Ubuntu Version: 14.04-w1-wip1
http://opentosca.org/nodetypes
Delete
Export
Versions
README
LICENSE
Visual Appearance
Instance States
Interfaces
Implementations
Tags
Requirement Definitions
Capability Definitions
Properties Definition
Inheritance
Documentation
XML
Interfaces
Add
Remove
OperatingSystemInterface
http://opentosca.org/interfaces/tests
Operations
Add
Remove
installPackage
transferFile
runScript
waitForAvailability
Save
Generate Implementation Artifact
Generate Lifecycle Interface
Input Parameters
Add
Remove
Name
Type
Required
VMIP
xsd:String
YES
VMUserName
xsd:String
YES
VMPrivateKey
xsd:String
YES
Output Parameters
Add
Remove
Name
Type
Required
WaitResult
xsd:String
YES


Winery
about
Service Templates
Generate Implementation Artifact
Java Package
org.opentosca.nodetypeimplementations
Node Type Implementation Name
Ubuntu-14.04-w1-wip1-Implementation
Will be created.
Final name
Ubuntu-14.04-w1-wip1-Implementation_-w1-wip1
Namespace
http://opentosca.org/nodetypeimplementations
There is no check for the name of the implementation artifact. The artifact template name will be reused as implementation artifact name without any further check.
Artifact Template Name
Ubuntu-14.04-w1-wip1-OperatingSystemInterface-IA
Will be created.
Final name
Ubuntu-14.04-w1-wip1-OperatingSystemInterface-IA_-w1-wip1
Namespace
http://opentosca.org/artifacttemplates
Cancel
Generate
Versions
Tags
XML
Save
Interfaces
Add
OperatingSystemInterface
Input Parameters
Name
VMIP
VMUserName
VMPrivateKey
Output Parameters
WaitResult
Remove





After editing the generated stub project, we can built it and load the resulting .war file to the artifact template in the tab *Files*.

Winery

about

Service Templates | Node Types | Relationship Types | Other Elements: Artifact Templates | Administration

Admin

Ubuntu-14.04-w1-wip1-OperatingSystemInterface-IAartifactTemplate
http://opentosca.org/nodetypeimplementations

Delete | Export | Versions

Type: WAR

README | LICENSE | Files | Source | Properties | Property Constraints | Documentation | XML

Durchsuchen... Keine Dateien ausgewählt.

ZIP

You may **drop the files** here.

The files are **immediately** uploaded without any confirmation.

Already included Files:

Add New | Reload from Server | Delete | Rename | Copy all to Files | Save

src/main/webapp/WEB-INF

src/main/java/org/opentosca/nodetypeimplementations

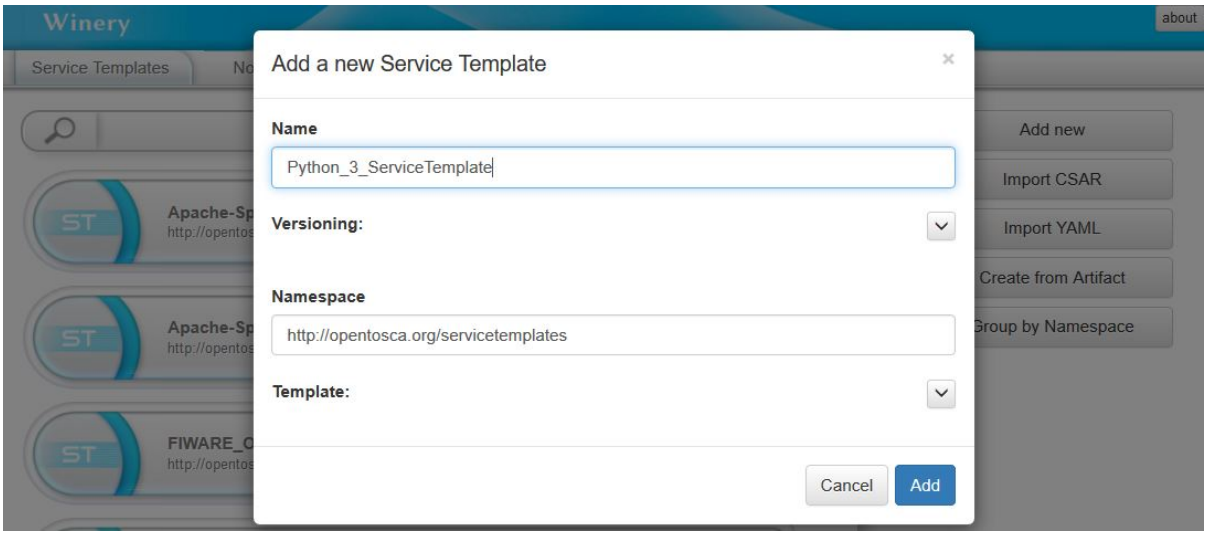
AbstractIAService.java	2.65 KB
org_opentosca_nodetypes_Ubuntu_14_04-w1-wip1__OperatingSystemInterface.java	4.74 KB
README.md	3.61 KB
pom.xml	11.74 KB

org_opentosca_nodetypes_Ubuntu_14_04-w1-wip1__OperatingSystemInterface.java

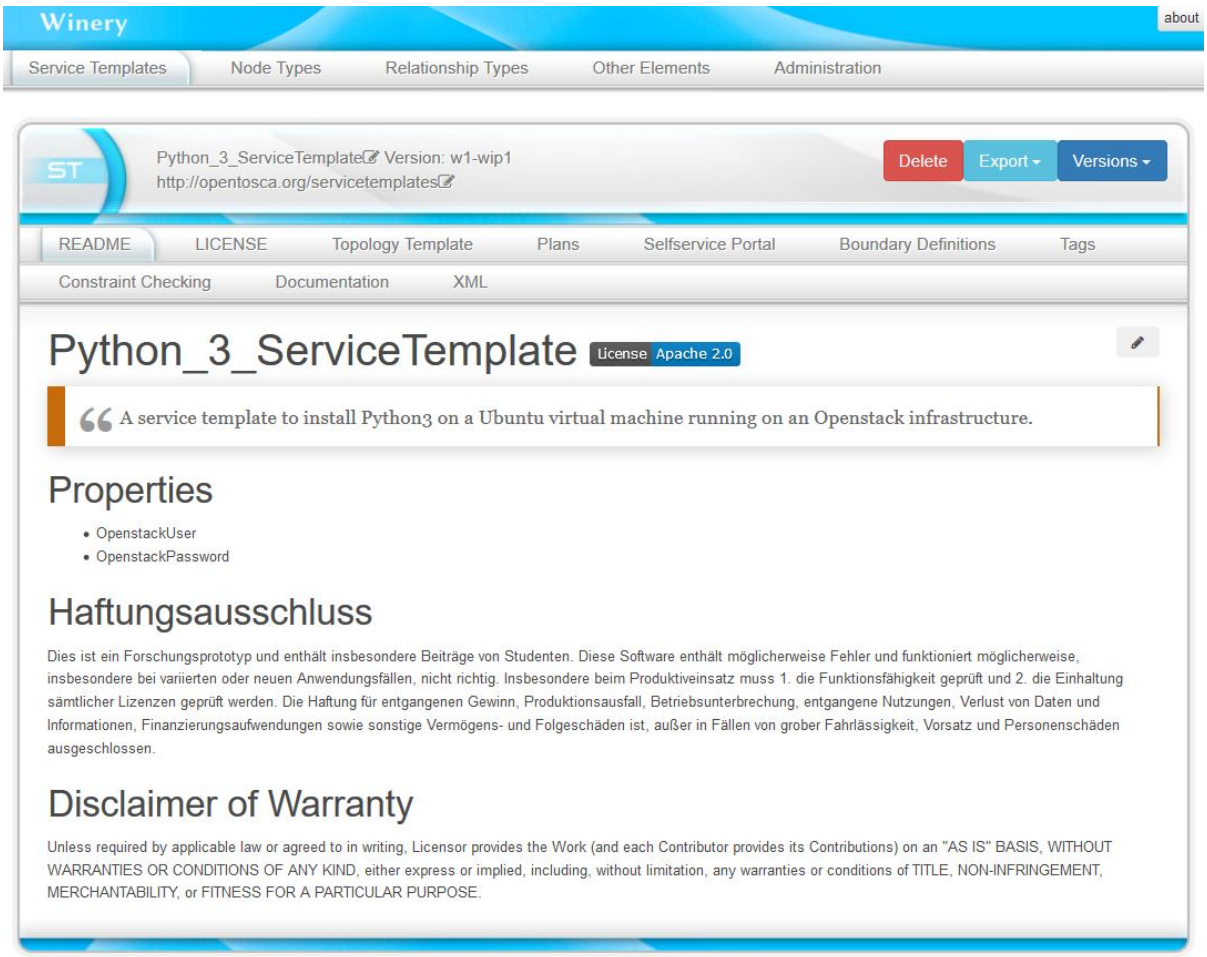
```
1  package org.opentosca.nodetypeimplementations;
2
3  import java.util.HashMap;
4
5  import javax.jws.Oneway;
6  import javax.jws.WebMethod;
7  import javax.jws.WebParam;
8  import javax.jws.WebService;
9  import javax.jws.soap.SOAPBinding;
10 import javax.xml.bind.annotation.XmlElement;
11
12 @WebService
13 public class org_opentosca_nodetypes_Ubuntu_14_04-w1-wip1__OperatingSystemInterf
14
15     @WebMethod
16     @SOAPBinding
17     @Oneway
18     public void installPackage(
19         @WebParam(name="VMIP", targetNamespace="http://nodetypeimplementations.o
20         @WebParam(name="VMUserName", targetNamespace="http://nodetypeimplementat
21         @WebParam(name="VMPrivateKey", targetNamespace="http://nodetypeimplement
22         @WebParam(name="PackageNames", targetNamespace="http://nodetypeimplement
23     ) {
24         // This HashMap holds the return parameters of this operation.
25         final HashMap<String,String> returnParameters = new HashMap<String, Stri
26
27         // TODO: Implement your operation here.
28
29
30
```

1.4.6 Creating the Service Template

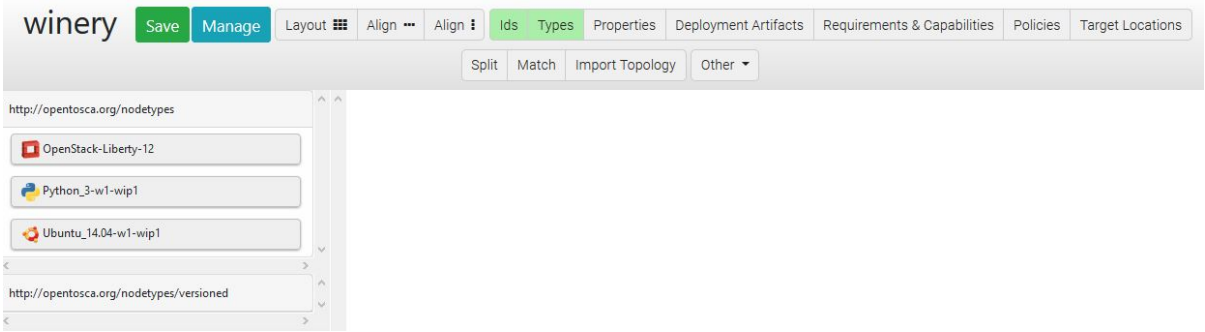
To finally model the service template, at the tab *Services Templates*, press *Add new*.



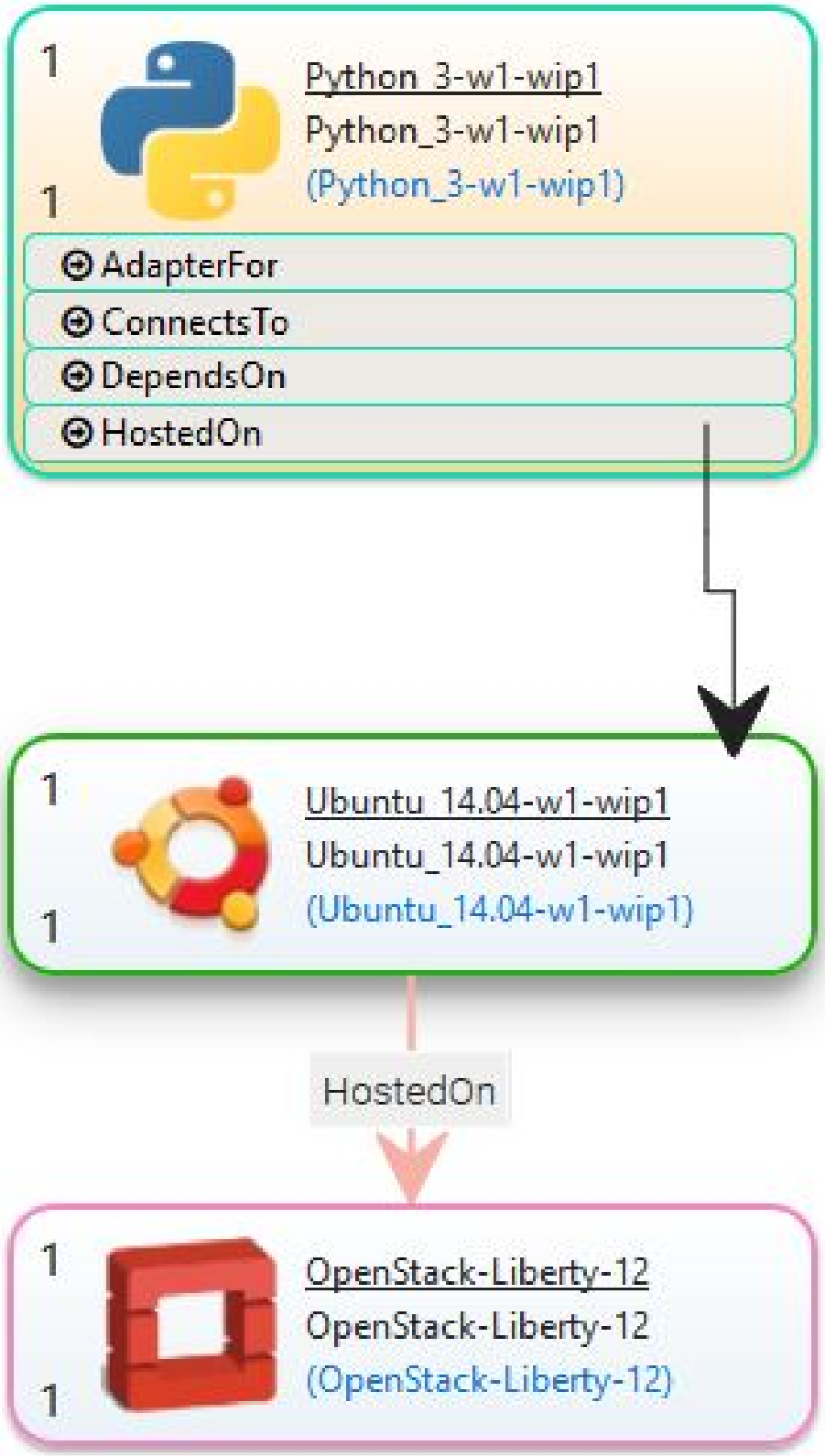
Go to tab *Topology Template* and press the button *Open Editor*.



In the editor, the *Palette* on the left shows the available node types, which can be drag and dropped in the modeling area.

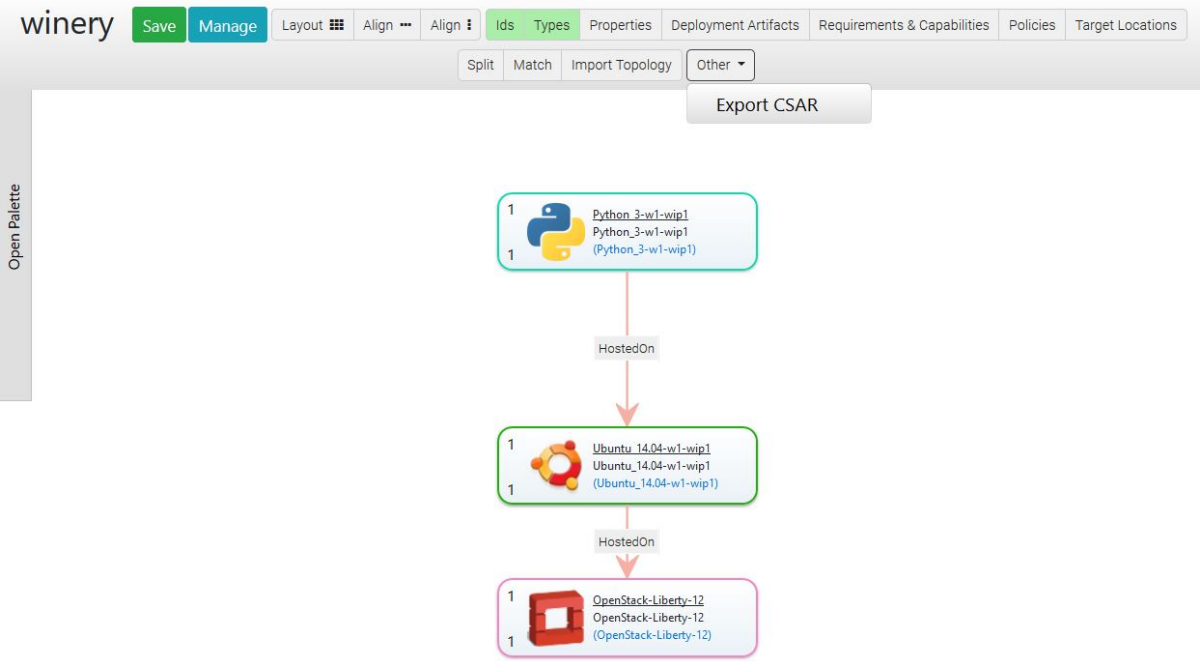


To model the relationship that the Python3 runtime is hosted on the Ubuntu virtual machine, click at the Python3 node template. This will show a list of possible relationship types (previously modeled). Click in the option *HostedOn* and pull the shown arrow to the Ubuntu node template area, in order to connect these node templates.


1
Python_3-w1-wip1
Python_3-w1-wip1
(Python_3-w1-wip1)
1
AdapterFor
ConnectsTo
DependsOn
HostedOn
1
Ubuntu_14.04-w1-wip1
Ubuntu_14.04-w1-wip1
(Ubuntu_14.04-w1-wip1)
1
HostedOn
1
OpenStack-Liberty-12
OpenStack-Liberty-12
(OpenStack-Liberty-12)
1

1.4.7 Exporting a Service Template Package

To export the Service Template as a CSAR package, press *Other*, then *Export CSAR*.



1.4.8 Miscellaneous Notes

Properties of a Template can be either full XML or key/value based. If key/value based, a wrapper XML element is required. Since QNames have to be unique, Winery proposes as namespace the namespace of the template appended by `propertiesdefinition/winery`. The name of the wrapper element is `properties`.

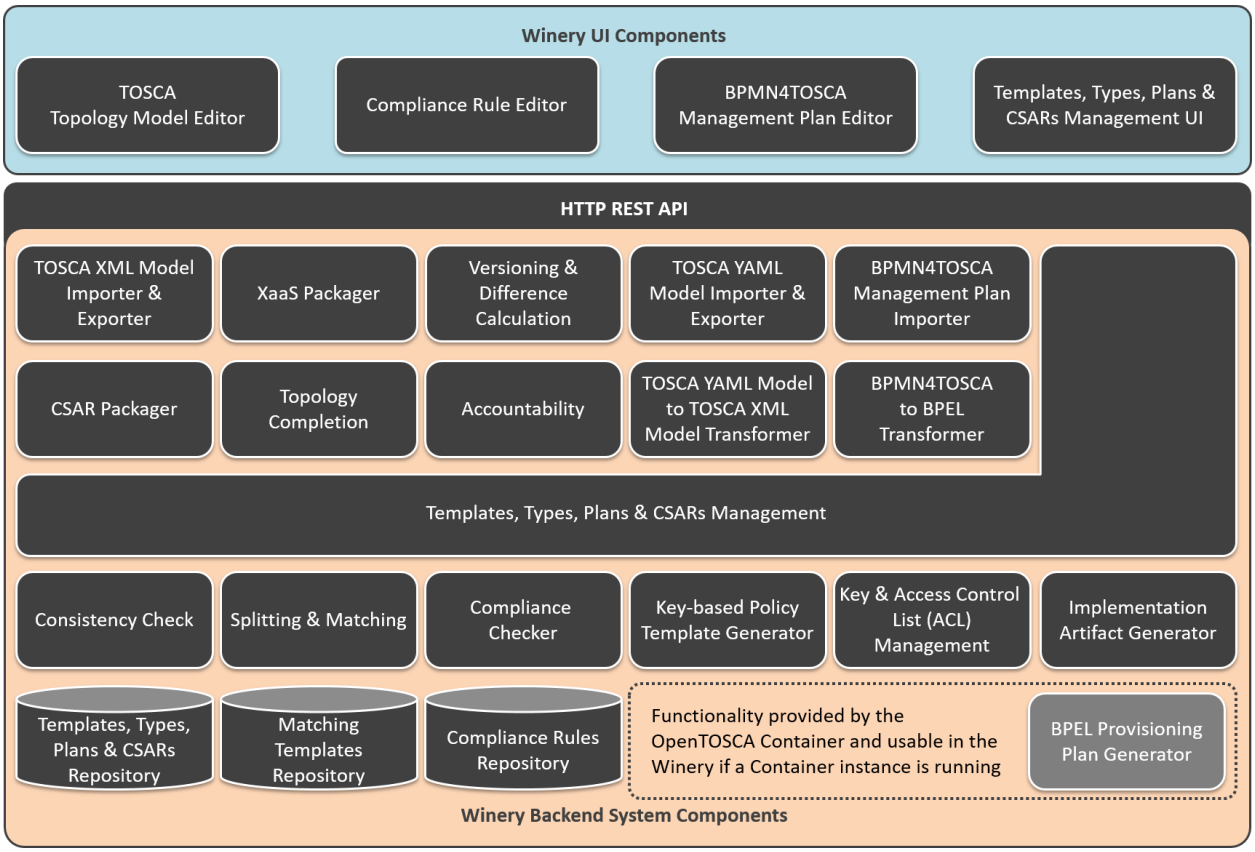
Note: Implementation hint: This is implemented in `PropertiesDefinitionComponent.onCustomKeyValuePairSelected` (TS) and `org.eclipse.winery.model.tosca.TEntityType.getWinerysPropertiesDefinition` (Java).

1.4.9 Uniqueness of QNames

Intentionally, a QName should be unique within the repository. We did not follow this assumption, but only require that QNames are unique within a type. That means, the repository allows `{http://www.example.org}id` for both a service template and a node type. We introduced DefinitionsChildId uniquely identifying a TOSCA element. Future versions might redesign the backend to use a QName as the unique key.

1.5 Component and Feature Overview

1.5.1 Components



The TOSCA modeling tool Winery mainly consists of four parts: (1) the templates, types, plans, and CSARs management, (2) the TOSCA topology model editor, (3) the BPMN4TOSCA management plan editor, and (4) the repository to store templates, types, plans, etc.

For the templates, types, plans, and CSARs management a user interface *Templates, Types, Plans & CSARs Management UI* that enables managing all TOSCA types, templates, and related artifacts is available. This includes node types, relationship types, policy types, artifact types, artifact templates, and artifacts such as virtual machine images. The *Templates, Types, Plans & CSARs Management* backend component provides functionality to access, store, or delete TOSCA elements in the *Templates, Types, Plans & CSARs Repository* which is a file system storing all available TOSCA elements.

The *TOSCA Topology Model Editor* enables the creation of service templates as directed graphs. Service templates consists of instances of node types (node templates) and instances of relationship types (relationship templates). They can be annotated with requirements and capabilities, properties, deployment artifacts, and policies. Modeled service templates can be exported based on the TOSCA XML standard using the *TOSCA XML Model Importer & Exporter* or

as YAML Model using the *TOSCA YAML Model Importer & Exporter*. Because the internal data model of the Winery is based on the XML standard the *TOSCA YAML Model to TOSCA XML Model Transformer* is required to enable the import and export as XMl as well as YAML model. The standard packaging format for service templates and all related TOSCA elements is a Cloud Service Archive (CSAR). The *CSAR Packager* backend component is responsible to package all TOSCA elements in the archive. The archive can be used by a TOSCA runtime for the deployment of the described cloud application.

The *BPMN4TOSCA Management Plan Editor* offers web-based creation of BPMN models with the TOSCA extension BPMN4TOSCA. That means, the editor supports the BPMN elements and structures required by TOSCA plans and not the full set of BPMN. The *BPMN4TOSCA Management Plan Importer* enables to load existing management plans to the Winery. Because not only BPMN but also BPEL is a common modeling language for the automated workflow execution, a *BPMN4TOSCA to BPEL Transformer* component is available to support different modeling standards. In case a running instance of the OpenTOSCA Container is available provisioning plans can be automatically generated by the *BPEL Provisioning Plan Generator*.

In addition to the described basis functionality of the TOSCA modeling tool Winery several advanced functionalities are provided:

- *Consistency Check*: This functionality enables to check whether a service template is valid according to the TOSCA XML specification. This includes the definition of used node types and properties, the QNames, and if License and README files are available. This supports the user to model valid service templates.
- *XaaS Packager*: It enables the deployment of, e.g., a web application by reusing an existing service template and replacing the deployment artifact in the specified node type with the new deployment artifact. The underlying platform or infrastructure services do not have to modeled for each application, predefined templates can be used. More information can be found here.
- *Topology Completion*: The TOSCA Topology Completion of Winery enables the user to model incomplete TOSCA Topology Templates and complete them automatically step-by-step by injecting new node templates to fulfill open requirements. More information can be found here.
- *Splitting & Matching*: The Split & Match function facilitates the redistribution of application components to target locations. For this, the application components can be annotated with target labels to indicate the desired target locations. In the *Matching Templates Repository* platform or infrastructure services can be defined as node templates or complete topology fragments for each target location. Based on the desired split, the node templates of the original service template are split according to the labels and the matching node templates or topology fragments for hosting the application's components in the target location are matched with the corresponding part of the split topology. More information can be found here.
- *Versioning & Difference Calculation*: To support version control of all TOSCA elements, including node types, artifact templates, service templates, and so on, the versioning component enables to add different versions of a TOSCA element and to release them after the development phase. Released elements can not be modified to ensure consistency of specific versions in the ecosystem. In addition, the differences between two versions can be calculated and visualized in the TOSCA Topology Model Editor.
- *Accountability*: In collaborative development of application deployment models in business-critical scenarios (such as data-analysis), accountability is of high importance. Thus, at CSAR export time, Winery enables to store the TOSCA meta file in a blockchain to identify the author of each exported version and whether a contained artifact is changed and by whom. Winery also stores these artifacts versions in a decentralized storage which facilitates comparing them and visualizing the provenance of a specific resource.
- *Compliance Checking (Compliance Rule Editor, Compliance Checker & Compliance Rules Repository)*: The Topology Compliance Checking of Winery enables to describe restrictions, constraints, and requirements for Topology Templates in form of reusable topology-based Compliance Rules. These rules can be modeled using the *Compliance Rule Editor* and stored in the *Compliance Rules Repository*. Each rule consists of an Identifier and a Required Structure. If the defined identifier is contained in a topology, the required structure must be contained as well. Furthermore, the *Compliance Checker* of Winery can be used to ensure that a given Topology Template is compliant to the current set of Compliance Rules. More information can be found here.

- *Key-based Policy Template Generator*: This functionality allows to generate security policy templates based on keys stored in the key manager. Since a key-based security policy represents a key in a decoupled manner, the policy template only contains the details about the key, but not the key itself. Modelers can use this functionality to simplify generation of policy templates which represent respective keys.
- *Key & Access Control List (ACL) Management*: This functionality allows storing and generating symmetric keys and keypairs with self-signed certificates as well as specifying the access rules for keys for specific partner names. It allows modelers to enforce modeled security requirements at CSAR import and export times. However, this is an administrative functionality that potentially can be used for other purposes.
- *Implementation Artifact Generator*: To specify what a node type should do, the user can define an interface and the operations provided by this interface. Once the operations of a node type are defined, artifacts (e.g., shell scripts, .war files) implementing these operations need to be modeled. With the *Implementation Artifact Generator* a stub java maven project to build a .war file for a defined interface is generated automatically.
- *Grouping*: This functionality allows the grouping of node templates in the TOSCA topology model editor. It enables the possibility to model groups within a topology, e.g., to describe that a policy only applies to a certain group of node templates, but not to all node templates of a topology template.

1.5.2 Features

- Splitting - Splitting functionality
- Target Allocation - Select best suited cloud provider for topologies
- Topology Completion - Topology completion overview
- XaaS Packager - Enables reusing modeled topologies as templates for single applications
- Compliance Checking - Enables compliance checking of topology templates based on reusable rules
- Implementation Artifact Generation - Shows how to generate and update an implementation artifact of type `war`
- Version Management - Shows how to update the version of a node template in the topology modeler
- Threat Modeling For NFV - Enables threat modeling capabilities and NFV-based mitigation recommendation
- Pattern-based Deployment and Configuration Models - Describes how PbDCMs can be crated and refined to executable deployment models
- Grouping - Describes the usage of the grouping functionality.

1.6 Winery CLI

The Winery CLI can be used to perform a consistency check for a given repository.

- Linux: `docker run -it -v $(pwd):/root/winery-repository opentosca/winery-cli winery -v`
- Windows: `docker run -it -v ${PWD}:/root/winery-repository opentosca/winery-cli winery -v`

Note: You may replace `$(pwd)` or `${PWD}` with a directory location on your Docker host system.

Currently supported CLI arguments:

```
-h,--help          prints this help
-p,--path <arg>    use given path as repository path
-v,--verbose       be verbose: output the checked elements
```

1.7 Frequently Asked Questions (FAQ)

1.7.1 Q: What is TOSCA?

A: The Topology and Orchestration Specification for Cloud Applications (TOSCA) is an OASIS standard to describe the deployment and management of applications in a portable manner. Based on standard-compliant TOSCA runtimes, such as the OpenTOSCA ecosystem, the deployment and management can be automated. For more details see our *notes on TOSCA*.

1.7.2 Q: What is a CSAR?

A: Cloud Service Archive (CSAR) is a packaging format defined by the TOSCA specification, which enables to bundle modeled TOSCA components in a self-contained manner. Besides the TOSCA elements, the executable artifacts are packed as well. In winery, you can model a service template and export it as a CSAR. This CSAR can be loaded into the OpenTOSCA container in order to deploy your application.

1.7.3 Q: How can I start the OpenTOSCA ecosystem?

A: You can start the ecosystem by simply using Docker Compose or by using installation scripts. Please refer to the OpenTOSCA getting started guide for more details.

1.7.4 Q: Is there an open repository for TOSCA types?

A: Yes! We provide a GitHub repository compatible to Winery, which contains several service templates, node types, etc. To use this repository with a Winery docker container, please refer to the corresponding configuration instructions in the user guide.

1.7.5 Q: Where can I find a quick start guide to model Node Types?

A: You can find a Winery quick start guide about modeling node types in our *user guide*.

1.7.6 Q: How can I export my modeled application as a CSAR?

A: Select the tab *Services Templates*. From the listed service templates, select the one you want to export. In the detailed view, press *Export* and then choose the option *CSAR (XML)*.

1.7.7 Q: My modeled Node Type got the suffix name *wip* what does this mean?

A: This means your node type has a *work in progress* (wip) version. That is, this node type can and might be changed. Once you are done, you can do a release of your node type. In this way, Winery will not allow changes in the (released) node type anymore.

1.7.8 Q: How can I release a Node Type?

A: Select the tab *Node Types*. From the listed node types, select the one you want to release. In the detailed view, press *Versions* and then choose the option *Release management version*.

1.7.9 Q: On Mac OS X, I can neither delete a Node Template nor a Relationship Template.

A: Select the node template (or the relationship template) and press <kbd>fn</kbd> + <kbd>backspace</kbd>.

1.7.10 Q: Where can I get more help?

A: If you need support, contact us at *opentosca@iaas.uni-stuttgart.de*.

1.7.11 Q: How can I contribute to Winery?

A: Please see the contributing guide.

CHAPTER

TWO

DEVELOPER GUIDE

This document provides an index to all development guidelines and background information of Eclipse Winery.

- Recommended Reading
- Modules - Winery's module structure
- Branches - How to branch
- Source Code Headers - Documentation about required source code headers
- REST API - How Winery's REST API works
- Encoding - Information about how encoding is used in Winery
- ID System - Winery's ID System
- Repository Layout - Documents the layout of the repository (stored as plain text files)
- Property Handling
- Configuration and Features
- *TOSCA 1.0 Notes*
- IDE Setup
  - IntelliJ IDEA (recommended): config/IntelliJ IDEA
  - Eclipse: config/Eclipse
- Winery GitHub Workflow
- Setup Winery Toolchain
- Winery and Docker

2.1 Getting Started

- Clone the repository: `git clone https://github.com/eclipse/winery && cd winery`.
- Build Eclipse Winery: `mvn clean install -DskipTests` (skipping the tests for a faster build).
- Setup your IDE:
  - IntelliJ IDEA (recommended): config/IntelliJ IDEA
  - Eclipse: config/Eclipse
- Go to Eclipse Winery Toolchain for further details
- Get familiar with Winery's GitHub workflow

CHAPTER

THREE

NOTES ON TOSCA

The *Topology and Orchestration Specification for Cloud Applications (TOSCA)* is a standard defined by the OASIS organization. It defines a language to model (cloud) applications to automate their provisioning and management. Thereby, TOSCA is vendor and technology independent and aims at defining applications in a portable and interoperable manner.

In general, there are two different flavours built in to TOSCA: (i) declarative modeling and (ii) imperative modeling. While the traditional, declarative way to model an application is in the form of a *Topology Template*, i.e., a graph that describes the application's components and their relations, it also supports imperative workflows that exactly state the tasks and their order in which they have to be processed. However, since we can automatically generate the imperative workflows based on the declarative model, Winery focuses mainly on the creation of the component's types, i.e., *Node Types*, and whole applications, i.e., *Service Templates* that add additional meta-information and wrap a *Topology Template*.

For more details about the standard, go to the specifications as linked below. For more documentation about how to model an application using Winery and the OpenTOSCA ecosystem, see <../user/xml/index.rst>.

3.1 Recommended Readings

1. Portable Cloud Services Using TOSCA. In: IEEE Internet Computing (2012) - Short overview.
2. TOSCA: Portable Automated Deployment and Management of Cloud Applications. In: Advanced Web Services (2014) - Longer overview.
3. TOSCA Simple Profile in YAML Version 1.3 - The simple profile in YAML.

See http://www.opentosca.org/sites/publications.html for a list of publications in the OpenTOSCA ecosystem.

3.2 TOSCA 1.3 YAML

- Official Specification
- `Class Diagram`
- `PlantUML`

3.3 TOSCA 1.0 XML (Deprecated)

- Official Specification
- `Class Diagram`
- `PlantUML`

3.4 Example TOSCA YAML Files

- Project RADON

3.5 Available TOSCA Implementations

- https://wiki.oasis-open.org/tosca/TOSCA-implementations

CHAPTER

FOUR

ARCHITECTURAL DECISION LOG

This lists the architectural decisions for Eclipse Winery.

- ADR-0000 - Use Markdown Architectural Decision Records
- ADR-0001 - Use filesystem as backend
- ADR-0002 - File system folder structure using type-namespace-id structure
- ADR-0003 - Double Encoded URLs
- ADR-0004 - OAuth with GitHub
- ADR-0005 - XML editor does not enforce validation
- ADR-0006 - Wrap properties in TOSCA properties element
- ADR-0007 - Custom URI for lifecycle interface
- ADR-0008 - No support for local git source clones
- ADR-0009 - Manual serialization of SnakeYAML
- ADR-0010 - TOSCA YAML deserialization using SnakeYAML
- ADR-0011 - Use Builder Pattern for Model Classes
- ADR-0012 - Provide Support for Custom Key-Value Properties
- ADR-0013 - Routes in the Repository Angular App
- ADR-0014 - Use Eclipse Orion as Editor
- ADR-0015 - Offer copying files from the source to the files folder
- ADR-0016 - Reflection test for TOSCA YAML builder
- ADR-0017 - Modify JAX-B generated classes
- ADR-0018 - Version Identifier in a Debian-like Form
- ADR-0019 - Versions of TOSCA elements in the name
- ADR-0020 - TOSCA Definitions contain excaly one element
- ADR-0021 - Use logback for logging
- ADR-0022 - tosca.model is more relaxed than the XSD
- ADR-0023 - Use Maven as build tool
- ADR-0024 - Use TravisCI for Continuous Integration
- ADR-0025 - Use same `logback-test.xml` for each sub project

- ADR-0026 - Store `LICENSE` and `README.md` in respective entity's root folder in a CSAR
- ADR-0027 - Use dasherization for filenames
- ADR-0028 - Use hardcoded namespaces for threat modeling
- ADR-0029 - IPSec Algorithm Implementation
- ADR-0030 - Support of multiple repositories
- ADR-0031 - Reuse the pattern refinement implementation for pattern detection

The template.md contains the MADR template. More information on MADR is available at https://adr.github.io/madr/.

CHAPTER

FIVE

GETTING SUPPORT FOR ECLIPSE WINERY

- In case you have concrete issues, please open an issue at https://github.com/eclipse/winery/issues.
- There is a mailing list available at https://dev.eclipse.org/mailman/listinfo/winery-dev.
- General information about Eclipse Winery is available at https://eclipse.org/winery.

CHAPTER

SIX

LICENSE