Open Digital Twin Standard

Foreword

This is the first version of a draft for an Open Digital Twin Standard (ODTS) created by the Technical Committee for the Open Digital Twin Standard which is currently constituted of the authors of the Open Digital Twin Platform (ODTP). The standard is meant to be inclusive and overarching technical implementations of digital twins across all potential applications in business, governance, and research. As such, we encourage each body interested in the subject to contact us to join the

Technical Committee for following drafts and the eventual standard. We are also open to input from International organizations, governmental and non-governmental, to take part in the work.

The procedures used to develop the first draft of this document and those intended for its further maintenance are described as follows:

1. The documentation of the ODTP has been extracted.
2. The text is generalized to describe a generic execution environment for digital twins.
3. The ODTP is taken as the reference implementation of the ODTS.
4. The ODTS may diverge from ODTP to accommodate other digital twin technologies and platforms.
5. Mutual agreement on the text of the standard is sought from all members of the Technical Committee.
6. New members of the Technical Committee are admitted by existing members in an agreed-upon manner.

Any trade name used in this document is information given for users' convenience and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, we refer to the outline by the ISO: https://www.iso.org/foreword-supplementary-information.html. They do not include contractual, legal, or statutory requirements. Voluntary standards do not replace national laws, with whose standards users are understood to comply with and which take precedence.

We follow the ISO guidelines for normative ISO deliverables to understand how to implement the standard:

- "shall" indicates a **requirement** which indicates objectively verifiable criteria to be fulfilled and from which no deviation is permitted if conformance with the document is to be claimed
- "should" indicates a **recommendation** which indicates a possible choice or course of action deemed to be particularly suitable without necessarily mentioning or excluding others
- "may" is used to indicate that something is permitted
- "can" is used to indicate that something is possible, for example, that an organization or individual is able to do something

In order to improve the quality of international standards, we follow the 6 principles introduced by the World Trade Organisation Technical Barrier to Trade Committee that clarify and strengthen the concept of international standards as far as applicable:

- transparency
- openness
- impartiality and consensus

- relevance and effectiveness
- coherence
- development dimension

For details of the principles, see Annex 4 to the Second Triennial Review of the TBT Agreement.

This document was prepared by the Technical Committee for the Open Digital Twin Standard (TC-ODTS) formed at the Swiss Data Science Center and the Center for Sustainable Future Mobility in the ETH domain In Switzerland. Currently, the members of the TC-ODTS are Jascha Grübel, Sabine Maennel, and Carlos Vivar Rios. Additional contributions were made by Robin M. Franken (on semantic validation) and Sabrina Ossey (on authentication). Milos Balac, Chenyu Zuo, and Stefan Ivanovic also reviewed the document.

Any feedback or questions on this document should be directed to the contact on the official repository https://github.com/odtp-org.

Introduction

The ODTS series defines a framework to support the creation of digital twins of observable processes in the real world (physical twins).

A digital twin monitors the observable processes and assists with analyzing these processes. Digital twins allow to increase the visibility into complex processes and make their digital representation transparent.

The digital twins supported by the ODTS framework depend on the implementation of the reference framework (e.g., ODTP) and the available components (and their technology and software requirements). Different domain applications may require different data standards. As a framework, this document does not prescribe specific data formats and communication protocols but informs on architectures that should be supported across different implementations to attain high-level interoperability.

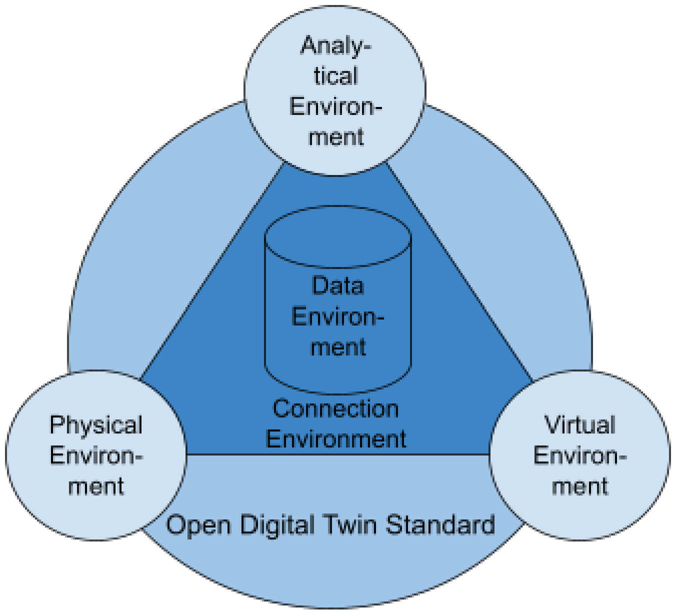

Figure 1: The relation between the five-component model for digital twins and the ODTS. The orchestration (dark blue) is handled fully within the ODTS (medium blue). The components (light blue) rely on external solutions beyond the ODTS to work but follow ODTS to be integrated.

Figure 1 shows the theoretical framework for digital twins with five environments, based on previous work on digital twins. The ODTS framework aims to provide a standard for these five environments. The environments are split into two groups. The first group manages how to orchestrate generic digital twins with the data and connection environment, and the second group provides individualized features for specific digital twins from preexisting independent components of three kinds matching the environments:

- Physical environment: Data loading and preprocessing
- Analytical environment: Data analysis including machine learning
- Virtual environment: Data Visualization, Data Interaction, and Decision-Making

Most digital twins vary in their features strongly depending on the area of application. Requirements of realtimeness differ with regard to data acquisition and data output. Sometimes, data acquisition varies from working only with historical data to the Internet of Things (IoT) real-time data acquisition and data output of simulation results may take days to compute compared to minutely updates of prediction models. Another issue that digital twin development often faces is that these twins are developed by a small group or a developer/researcher with a narrow focus on just one topic (e.g., data visualization, data simulation, data analysis) without the expertise to build an interoperable digital twin.

ODTS facilitates combining these digital twins' features so that a bigger goal can be achieved without forcing developers into an intense collaboration and without having to maintain a complex product. Complex management is taken on the orchestrator, and individual features are developed in the components with little overhead and full control of the development process. ODTS also defines the properties of a marketplace of components named the component zoo to facilitate the exchange of digital twin components to attain interoperability.

Scope

The ODTS series contains the following four parts:

- **ODTS-1 Components**: A recipe to turn a tool into reusable components and a metadata specification for these components.
- **ODTS-2 Workflow**: A specification of how components are chained together to produce workflows in the shape of directed acyclic graphs.
- **ODTS-3 Orchestrator**: A specification for an orchestrator that how to instantiate executions of workflows.
- **ODTS-4 Zoo**: A registry called Zoo where components and workflows can be registered and discovered for reuse and reproducibility.

Figure 2 shows the relationship between the four parts in the series.

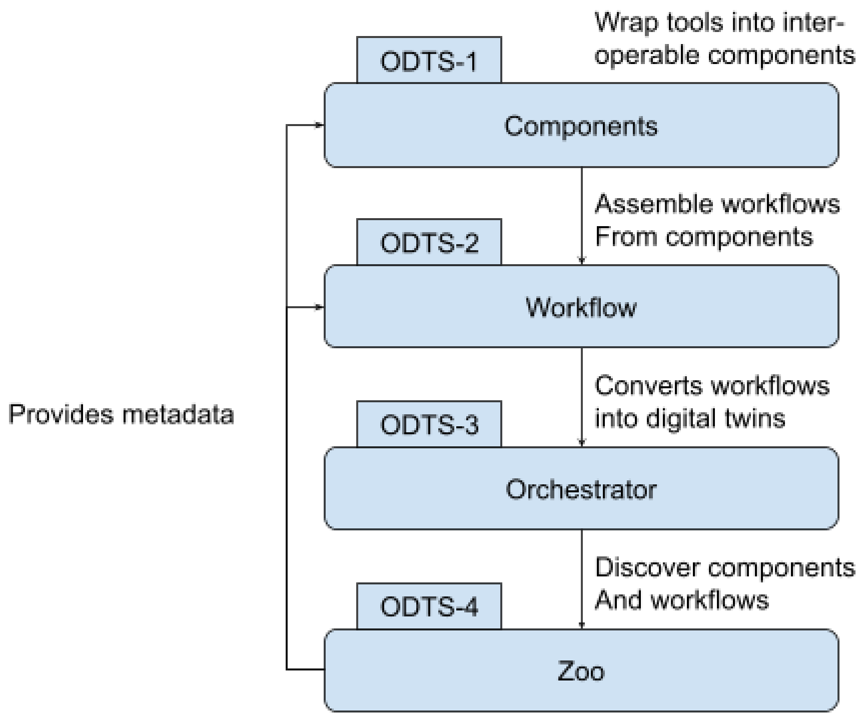
ODTS-1
Wrap tools into inter-operable components
Components
Assemble workflows From components
ODTS-2
Workflow
Converts workflows into digital twins
ODTS-3
Orchestrator
Discover components And workflows
ODTS-4
Zoo
Provides metadata

Figure 2: ODTS series structure

Normative references

The procedures used to develop this document and those intended for its further maintenance are described:

- SHACL: https://www.w3.org/TR/shacl/
- RDF: https://www.w3.org/TR/rdf11-concepts/
- Semantic Versioning: https://semver.org/
- Git (Version control)
- MongoDB (Document Database)
- S3 (Storage System): https://min.io/
- Docker (Container Technology)
- Keycloak (Identity and Access Management)
- OpenID Connect (authentication and authorization protocol)

Informative references

- MATSim: https://www.matsim.org/

- Eqasim: https://eqasim.org/
- SDSC: https://www.datascience.ch/
- CFSM: https://csfm.ethz.ch/

Terms, definitions and abbreviated terms

Digital Twin

A digital twin is an abstract construct to represent a physical twin by capturing key characteristics of interest to describe and analyze observable processes to a sufficient degree for a dedicated task and then make a decision or potentially actuate on the physical twin based on the findings. Degrees of realism required by a task may vary and this definition tries to accommodate all variants of digital twins, including subsets such as Digital Shadows (raw data visualization) and Digital Models (no coupling to the physical twin). In some cases, digital twins are coupled to virtual twins in case the physical twin does not exist yet to explore its potential properties. A virtual twin provides data to a digital twin that behaves as if it came from a physical twin. ODTS covers all these variations and henceforth will refer to them plainly as digital twins.

Open Digital Twin Platform (ODTP)

The reference implementation of the ODTS: A tool designed to generate specific digital twins by integrating into a platform how to design, manage, run, and share digital twins. It offers an interface (CLI and GUI) for running and managing digital twins. It wraps different open-source technologies according to ODTS to provide a high-level Application Programming Interface (API) for the final user. Finally, it implements a zoo as a repository of searchable components to (re-)create digital twins. The current version of reference implementation may not implement all features of the standard yet but aims to do so in the long-run. The reference implementation can be found here: https://github.com/odtp-org/odtp

Components (ODTS Term)

Components for a digital twin are used to instantiate tools for specific tasks by providing implementations of the three individualized environments (see Figure 1). Each component provides an extension to the available capabilities under ODTS that implementations (such as ODTP) may use to perform specific tasks in the digital twin. These extensions are generated by the community for the community, and their specific capabilities are not part of ODTS, but ODTS describes the features a component must have to be interoperable. This includes that the input/output of a component is validated semantically and that they run within a (docker) container as an independent micro-service. Typically, they can be one of the following categories:

- Dataloader component (implementing the physical environment of the digital twin)
- Analytical component (implementing the analytical environment of the digital twin)
- Visualization component (implementing the virtual environment of the digital twin)

Core/core-optional modules (ODTS Term)

Modules for a digital twin are used to instantiate solutions for generic tasks with a digital twin by providing implementations for features in the two orchestrating environments (see Figure 1). Each module is tightly integrated into the reference implementation (e.g., ODTP) and is developed together with the maintainer of the reference implementation and deployed as needed. These core modules **shall** include the programmes needed to run the ODTS implementation and wrap the services used into a user interface. Core-optional modules are not mandatory to run an ODTS implementation with the minimal features (e.g., running manually ODTS components) but **should** be implemented to provide a more complete experience of the ODTS standard.

Services

ODTS follows a micro-services architecture to generate a digital twin. Each service refers to one logical unit that performs one specific task in an independent manner to produce a digital twin. In ODTS, both modules and components are instantiated as (micro-)services. These are chained together to produce an effective implementation for a specific digital twin. The digital twin combines generic features for digital twins (modules) with individual solutions for a specific task (components).

Semantic input & output validation

In the ODTS abstraction Digital twin components can be considered data converters. The input to such a converter must be a file structured in a certain way. The tool inside the component converts this input into a different format, such as data needed for visualization, a different structure, an obfuscation of data or other types of processing to the data such as aggregation, simulation, interpolation, and extrapolation. As the functioning of a component is dependent on the data it receives, a structured mechanism of describing the input data requirements ensures proper functioning of a component. A well-described input schema can help a user assess whether their data set will comply with the requirements of the component. The semantic description includes information about the filenames, folder structure, (multi-lingual) labels and definitions of the columns, keys, or properties within such a file, and some information about the datatypes expected for each parameter.

Having the input requirements accurately and structurally described is important, but as components may require many files, with many parameters associated to each file, automated validation of a dataset becomes a requirement for ODTS. The automation of validation of each file in the input dataset may be performed in two steps. First, all files are discovered that exist in the input dataset and their associated parameters which are present within each file. Second, an expectation is formulated about what data is needed. The Instance data is an RDF-compliant representation of the input files. This entails triples in an RDF compliant file format, describing a given input folder, with metadata about which files exist in this folder, and which variables exist in each file. The Schema data (SHACL shapes) is an RDF compliant definition against which Instance data is checked. In other words - this design allows a data owner to check whether their data (or the output of another component) is compatible with the input requirements of a component.

Workflow

A workflow describes a chain of components that are supposed to be executed to produce the functionality of a digital twin. Workflows can be sequential but should also support Directed Acyclic Graphs (DAG). This allows workflows to execute complex tasks for digital twins by splitting the data flow and joining them where necessary.

Execution and Trace

An Execution instantiates the components in a workflow into a set of micro-services that are run once and for which operational data and results are captured. Every step of an Execution is logged in a trace of the digital twin for reproducibility purposes. A Digital Twin can be configured so executions are performed recurrently on time providing basic support for real-time setup.

Step

A step describes the running of a single service (instantiated from a component) in the execution of a workflow. Steps are atomic from the orchestrator's perspective and therefore, the logging is limited to parameters, exposed metrics, input data, and output data.

ODTS-1 Components

ODTS-1 Components are the centerpiece of ODTS: An ODTS-1 Component is a wrapper to an existing tool. This wrapper makes the tool usable in an [ODTS-3 Orchestrator]**.** The instantiation of an ODTS-1 Component is not independent of the orchestrator, as the transformation from tool to component depends on the ODTS-3 Orchestrator's own implementation. However, ODTS-1 Components shall be usable by any compliant ODTS-3 Orchestrator. ODTS-1 Components can be arranged into ODTS-2 Workflows and can be discovered by sharing them in an [ODTS-4 Zoo]. An orchestrator can search ODTS-4 Zoos to find ODTS-1 components to assemble a digital twin according to a ODTS-2 Workflow.

ODTS-1 components consist of the following elements further explained in the sections below:

- **Tools**: explains what is considered a tool in the context of ODTS
- **Components:** explains how a tool can be turned into an ODTS-1 Component
- **Versioning**: explains the versioning of both tool and component and how they are coupled
- **Metadata**: explains the metadata that are expected for a ODTS-1 Component
- **Types**: explains the component types

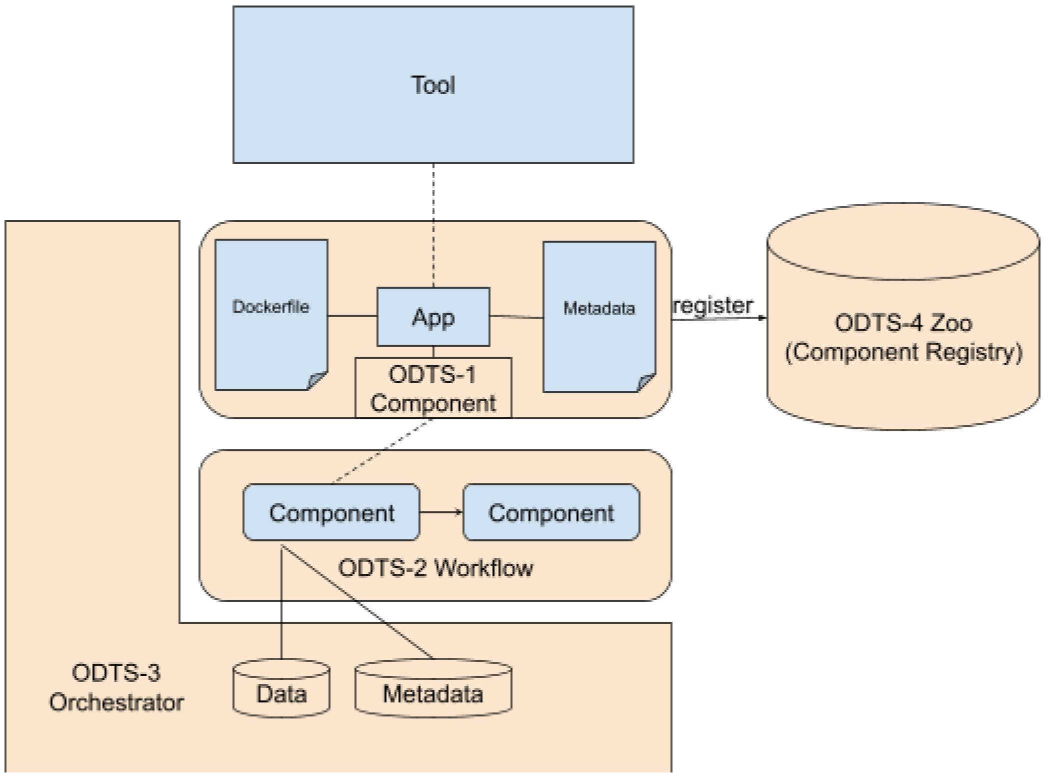

Figure 3: ODTS framework with component, orchestrator and zoo

Tools

A tool is a git repository: hosted by a git registry such as github or gitlab.

- Tools shall provide a license.
- They should be versioned with [semantic versioning].

The ODTS-1 Component is another git repository derived from the tool to integrate the tool into a digital twin workflow:

- It is linked to the tool in a standardized way, as described below
- The component shall add standardization to the tool that makes it compatible with the [ODTS-3 Orchestrator] and the [ODTS-4 Zoo] .
- The component shall separate digital twin features from the tool features, keeping the tool independent of information on how a digital twin is operated.

Components

An ODTS-1 Component is a version-controlled repository (e.g. git) and shall provide the following elements: (here we give a list and below you will find a description of each element in detail)

- A **Dockerfile** that shall be adapted to both the tool and the [ODTS-3 Orchestrator].
  - A dockerfile may support multiple orchestrators but shall support at least one.
- A set of **setting files** as with a standardized name:
  - The parameter may be found in `parameters.yaml` or `parameters.json`
  - The metrics may be found in `metrics.yaml` or `metrics.json`
  - The semantic input schema may be found in `semantic-input.yaml` or `semantic-input.json`
  - The semantic output schema may be found in `semantic-output.yaml` or `semantic-output.json`
- A metadata file with a standardized name.
  - The metadata shall be found in `odts.yaml` or `odts.json`
  - The setting files must be mentioned and described in the metadata file.
- An **app** script that shall check out the tool from a repository. It shall call methods of the [ODTS-3 Orchestrator client] and it shall call at least one method of the tool.

App

An ODTS-1 Component shall contain an app that connects to both the methods of the tool and the methods of the orchestrator. It is a bash script that shall be called in the Dockerfile (defined below) by the ODTS-3 Orchestrator with the following tasks:

- It shall checkout the tool from its git repository
- It shall use the client library for the [ODTS-3 Orchestrator] to capture the operational metadata (e.g. sharing progress updates with the orchestrator)
- It shall run the tool by calling the appropriate methods it provides

Dockerfile

An ODTS-1 Component shall set up the environment for the tool with a dockerfile and shall install the necessary dependencies. It shall prepare a folder structure inside the (docker) container to match the folder structure that the ODTS-3 Orchestrator expects for interoperation. It may create a folder structure that the tool expects. At last, it shall call the **[app]** as a bash script to execute the tool to achieve the desired task of the component.

Metadata file

An ODTS-1 Component shall include metadata on the component with the following parts:

- The metadata on the component shall be stored (authors, license, description, repository).
- Parameters that the component exposes may be stored. These parameters shall be exposed to the ODTS-3 Orchestrator. They can be changed by the user to run the component.
- Parameters as defined above may also be exposed in a parameters file.
- Metrics that the component exposes may be stored. These metrics shall be exposed to the ODTS-3 Orchestrator. They can be read by the user when running the component.
- Metrics as defined above may also be exposed in a metrics file.
- The component should add a link to a semantic schema for the expected inputs and outputs as a SHACL shapes graph that can then be used by the ODTS-3 Orchestrator to perform validation checks on the inputs and outputs. ODTS-4 Zoos may use the input/output schemas to define compatible components.

Parameter Metadata file

Parameters are defined as key-value pairs and may be structured in lists or arrays corresponding to the yml and json standards. Parameters are passed into the tool through the user-defined app script. Only parameters that are properly passed into the tool may take effect.

Metrics Metadata file

Metrics are special outputs of a tool to the ODTS-3 Orchestrator data storage. Metrics shall be provided by the tool developer. They shall be used for control and comparison of multiple executions and digital twins on a high level. They shall implement in the component a function, a path or stream/socket from where to store the metric data in the step representation of the ODTS-3 Orchestrator. Metrics may have semantic descriptions.

Semantic Input & Output Schema

The component author should semantically describe the data inputs and outputs required for the tool in a RDF-compliant format. All required files and parameters should be accurately described with labels, definitions and restrictions to enable semantic reasoning about a component. The output of the component should also be described entirely to create a semantic black-box view on the component.

Versioning

The versioning of the tool and component need to be independent from each other, since there may be various reasons that justify a new component version:

- The ODTS-3 client has changed
- Elements of the Component such as the Dockerfile or the App have changed
- The version of the tool that is used in the component has changed

Since the Tool version is important, it shall be added in the metadata of the component, see [Metadata File]. We recommend adding an automatic check that ensures that the version of the tool mentioned in the odts.yaml file and the version that is actually used in the component match.

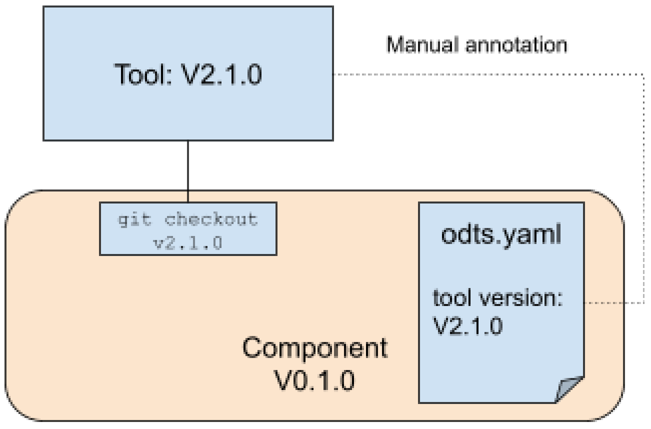

Figure 4: Tool version should be added in the metadata of the component

Metadata

The metadata schema for the Component is described at [ODTS-4 Zoo].

Types

ODTS implementations shall provide the following component types:

- Ephemeral components
- Interactive components
- API components

Ephemeral components

Ephemeral components shall be temporary and shall not persist data (e.g. transforming data from one format into another). They shall be used for short-lived analytical operations and discarded after use. They shall be built when preparing the digital twin execution and shall only be used in a single execution step. Parameters shall be provided as environment variables, and input data shall be placed in one specific directory.

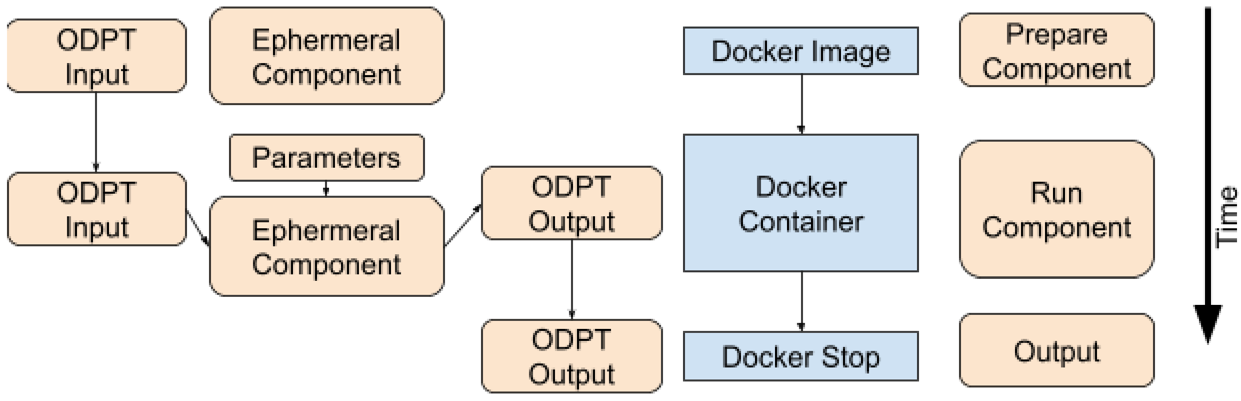

Figure 5: Run of an ephemeral component

Interactive components

Interactive components shall be designed to interact with the user. These components should be used in user interfaces or visualizations. They shall be built in a certain execution step but will keep running as a (docker) container until the user stops them. They are often the last step in an execution.

In real time settings, interactive components may expose parameters of other components forwarded by the orchestrator. Changing those parameters may trigger another execution of all components impacted by the change.

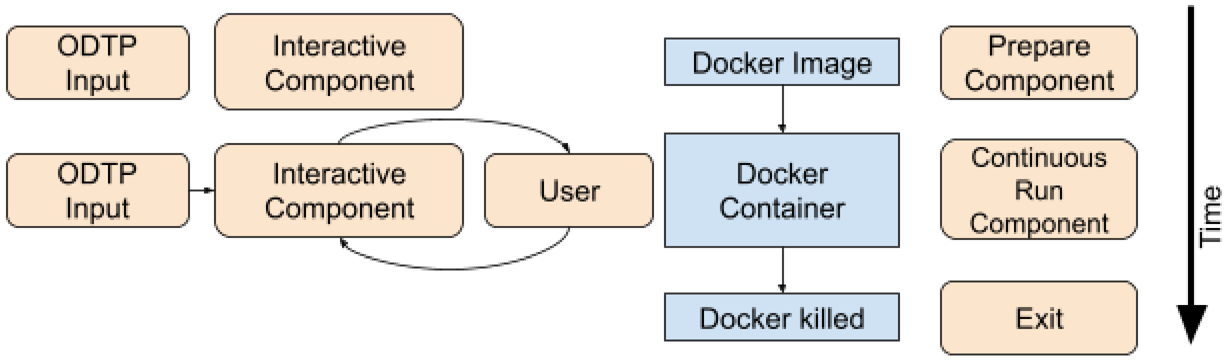

Figure 6: Run of an interactive component

API components

API components shall be built only once and can be reused in multiple executions. We can think of them as continuously running Microservices. The component is going to provide an API-Endpoint in one port that will allow running the tool in parallel or in multiple executions. This kind of component is useful when the component building process (in docker) takes a large amount of time, or when a long-lasting task can be reused in multiple executions. An example of this is the loading of a machine learning model into memory.

The API component receives the variables as JSON in the request's payload. This allows a more complex configuration of parameters than in the Ephemeral type. Input data can be provided in the request, or, if the file size is big, as an item in the S3 storage.

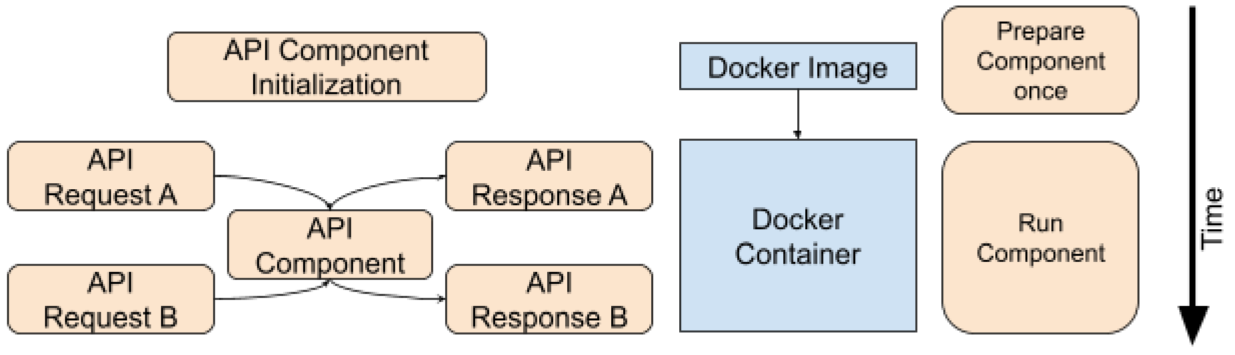

Figure 7: Run of an API component

ODTS-2 Workflow

Workflows are blueprints on how to assemble ODTS-1 Components to perform the tasks of a digital twin. A workflow shall describe the sequence of ODTS-1 components required to perform a task. A workflow may take the form of a directed acyclic graph (DAG).

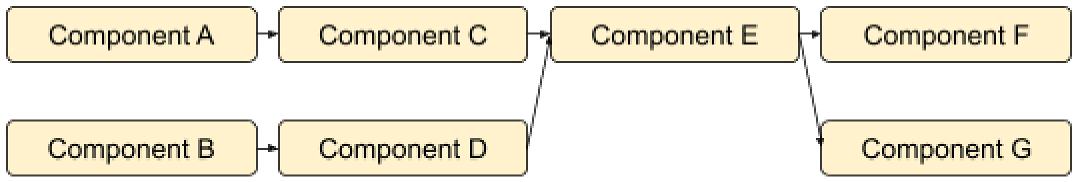

Figure 8: A workflow of components represented as a DAG.

Workflows should be editable with a workflow editor such as [BARFI]. A workflow shall contain all the information necessary to run an execution, including: components, parameters values, and initial inputs.

Examples of workflows are provided in Appendix B and Appendix C as they have been built in the context of [ODTP].

Workflow Templates

Workflows where components are chosen but parameters are not yet set shall be called Workflow Templates. They can be reused for workflows that differ on their inputs. These templates may be used to create and run executions programmatically.

ODTS-3 Orchestrator

An ODTS-3 Orchestrator is a tool that shall combine ODTS-1 components into ODTS-2 Workflows of directed acyclic graphs and shall instantiate them as an Execution of micro-services running as (Docker) containers.

The orchestrator shall provide the following elements that are further described in this section below:

1. [User Interface]: To define, run, and inspect Executions of ODTS-2 Workflows
2. [Data Storage for Component Outputs]: To transfer data between components
3. [Semantic Validation]: Validation engine for validation of semantically annotated input data against semantic component input requirements.
4. [Operational Data Storage]: To document ODTS-2 Workflow executions
5. [Tool Adapter]: To turn tools into ODTS-1 components that are compatible with an ODTS-3 Orchestrator
6. [Client Library]: A library that can be mounted in the ODTS-1 components to communicate with the ODTS-3 Orchestrator

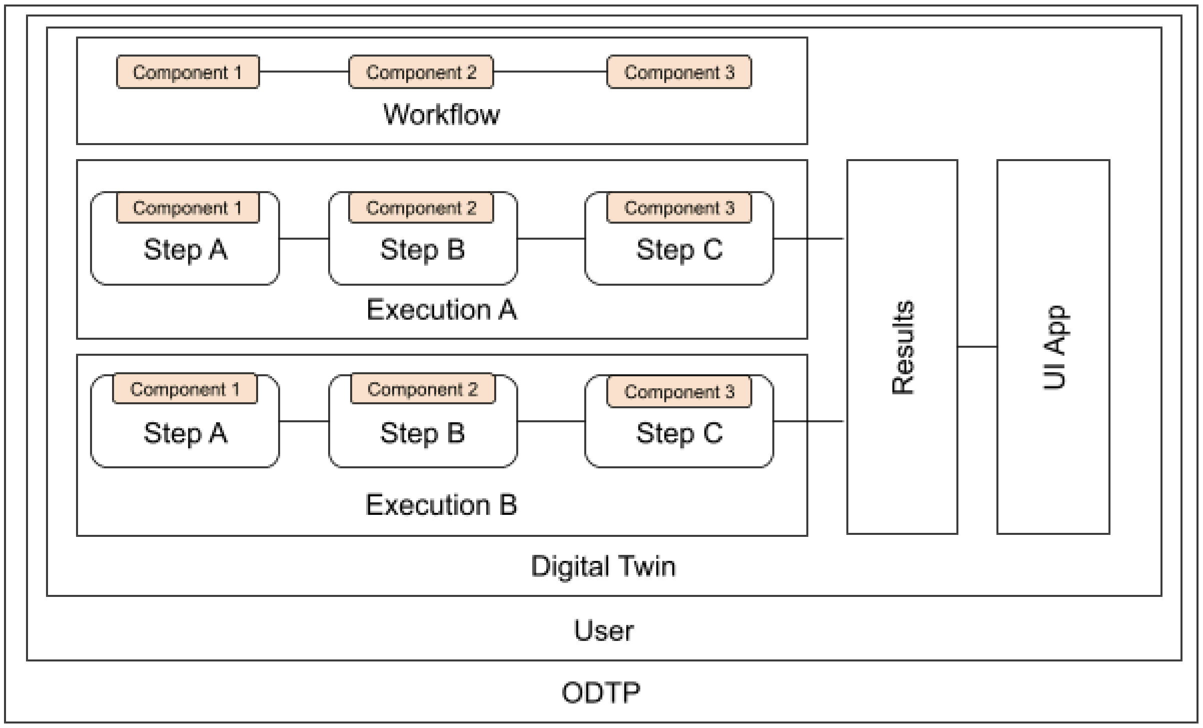

Figure 9: A possible implementation of ODTS in ODTP. Digital Twins are owned by users. A workflow is used to instantiate an execution of steps. The output of selected steps can be visualized as a result.

User Interface

The user interface of the ODTS-3 Orchestrator shall enable the following actions on ODTS-1 Components and their management:

- **Register ODTS-1 Components,** including versions of those components
- **Register Users**
- **Allow Users to create projects called digital twins**
- **Allows Users to create workflows as acyclic graphs**, where each node is coupled to the version of an ODTS-1 Component: such an ODTS-2 Workflow is instantiated as an execution, and each node represents a step in the execution.
- **Allows Users to prepare an execution**: The (docker) images for all the steps are built following the workflow and the filesystem of the user is prepared for writing the outputs.
- **Allows users to run executions**: Once all (docker) images are available and the file system has been prepared the ODTS-1 Components are run as specified in the Workflow as (docker) containers.
- **Execution runs shall be traced** by the ODTS-3 Orchestrator to produce a logging trace.
- **Provide a data storage** to exchange data between the execution steps.

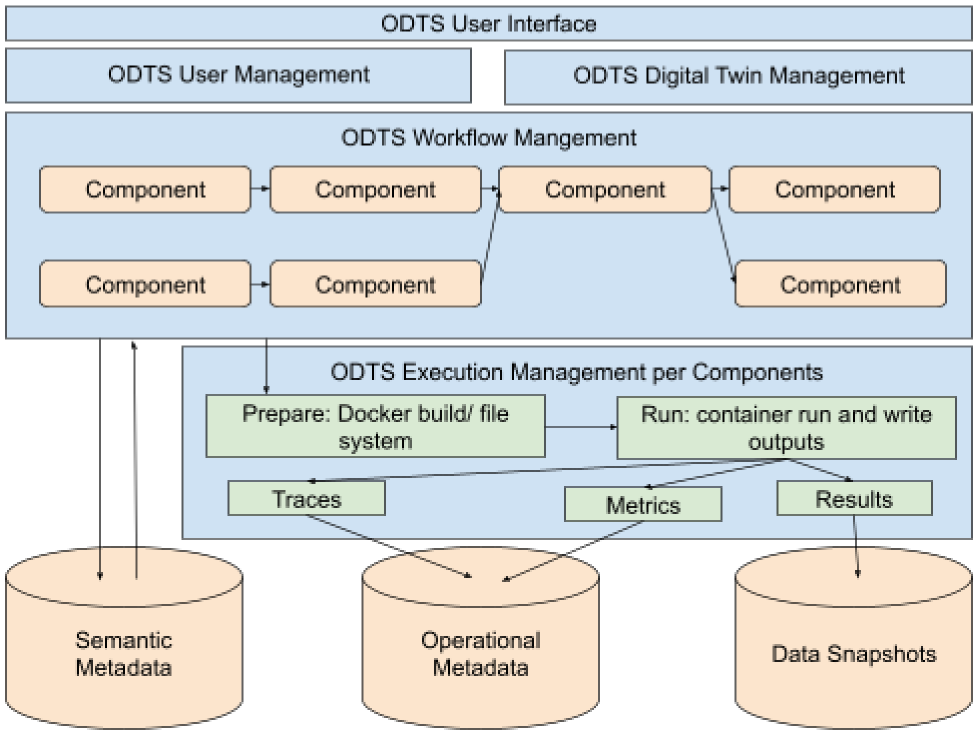

Figure 10: The User Interface needs to provide visual access to user management, digital twin management, workflow management and execution management. Note that digital twin specific visualizations are implemented by the components and not in the Orchestrator.

Usage of Docker

Docker image and container names can be derived in an automated way from the component versions. The ODTS-3 Orchestrator should run on a server and should prepare a project folder to write the file outputs of each step.

Data Storage

The ODTS-3 Orchestrator shall implement a data storage that allows to store outputs of component runs. When components are combined into workflows, the ODTS-3 Orchestrator shall facilitate the exchange of data between consecutive components. In [ODTP] an S3 Data storage is used for that purpose.

The stored data can be used for partial executions of workflows, e.g. starting with the second component in a sequence but reusing the data loader component. This enables the user of the ODTS-3 Orchestrator to reuse the output of computationally heavy components..

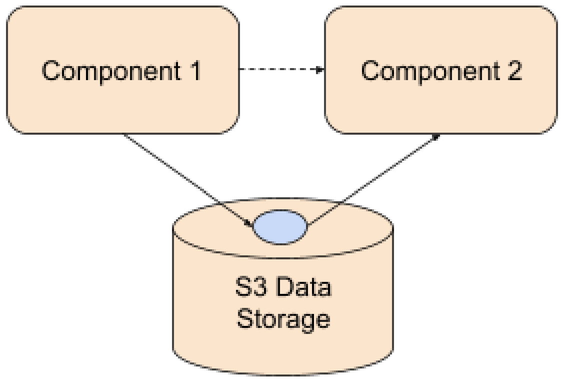

Figure 11: The intermediate storage of data for transfer between components.

Semantic Validation

The ODTS-3 Orchestrator shall automatically extract semantic information about a directory (recursively analyze the files in a folder, and extract some metadata about the columns/properties of each file) and write it to an RDF graph. This “instance data” (metadata about the input dataset) shall be validated against the “Schema data” provided by an ODTS-1 Component. This shall be implemented using SHACL. The ODTS-3 Orchestrator shall run a (SHACL) validation engine on this combination to generate a report that provides information about the conformance of the dataset with the schema.

ODTS-3 Orchestrators should warn users of non-conforming component connections. ODTS-3 Orchestrators may deny the execution of a workflow that does not conform to ODTS-2 Workflows. The mode for a warning is called "Lazy execution", where a workflow may run until an error occurs. The mode for strict conformance is "Safe execution", where a workflow is not run if non-conforming.

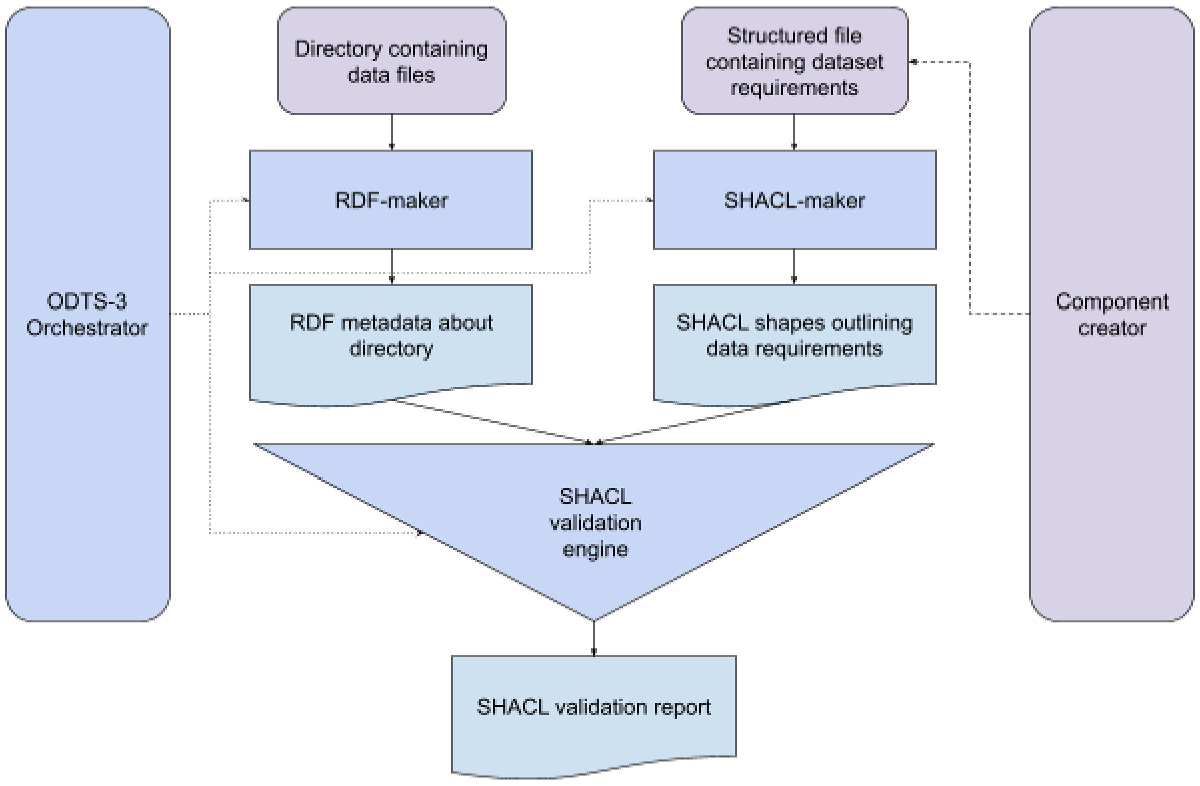
Directory containing data files
Structured file containing dataset requirements
RDF-maker
SHACL-maker
ODTS-3 Orchestrator
RDF metadata about directory
SHACL shapes outlining data requirements
Component creator
SHACL validation engine
SHACL validation report

Figure 12: Semantic validation for ODTS. The component creator (right) defines the dataset requirements and the structure of the directory. The orchestrator (left) provides implementations for the RDF-maker, SHACL-maker and SHACL validation engine. At runtime, the requirements and actual data are fed into the RDF-maker and SHACL-maker to obtain metadata that allows the SHACL validation engine to generate a final report on requirement compliance.

Operational Data

There are three kinds of operational data in ODTS:

- **Building Material:** Base classes that are used to build Executions. These are the building blocks that are shared in the ODTS-4 Zoo
- **Governance of Digital Twins**: Data that defines Digital Twins and its Executions. Digital Twins are usually set up by users, that intend to execute them in the ODTS-3 Orchestrator
- **Execution of Digital Twins**: Run Data: Data that capture the run of an Execution

Building Material

The following classes exist in ODTS-3 to provide building material for Execution: this building material is usually registered in the ODTS-4 Zoo and shall be imported from there into the ODTS-3 orchestrator:

- **Components:** ODTS-1 Components as registered in the ODTS-4 Zoo
- **Versions**: Different Versions of Components, see the section [Versioning] in ODTS-1
- **Workflow-Templates**: Acyclic Graphs of Components as defined in ODTS-2
- **Workflows**: Workflow-Templates with specified inputs and parameters, see [ODTS-2 Workflows]

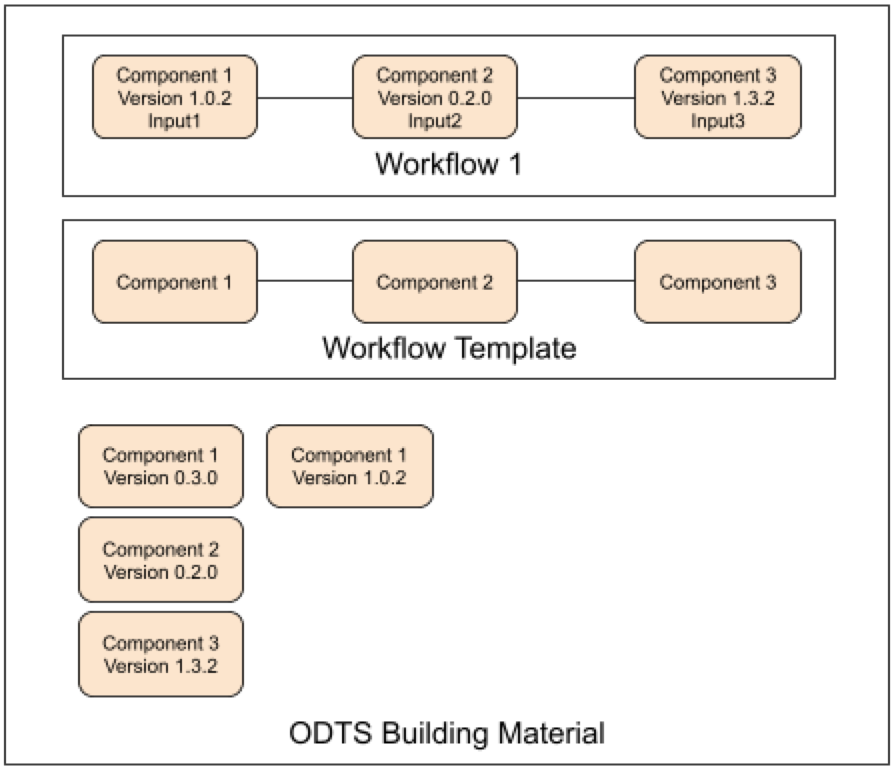

Figure 13: Building Material for a Digital Twins

Governance of Digital Twins

The following classes provide governance for the Digital Twins:

- **Users**: the users of the ODTS-3 Orchestrator. The users should be authenticated, see section [Authentication]
- **Digital Twins**: The Digital Twins are the projects of ODTS-3. They can be seen as the use cases. They usually contain Executions of Workflows that share a result, see Figure 9 above
- **Executions:** Executions are Workflows intended to be executed as a pipeline of docker containers. Preparing and run
- **Steps**: Steps in an Execution correspond to versions of components

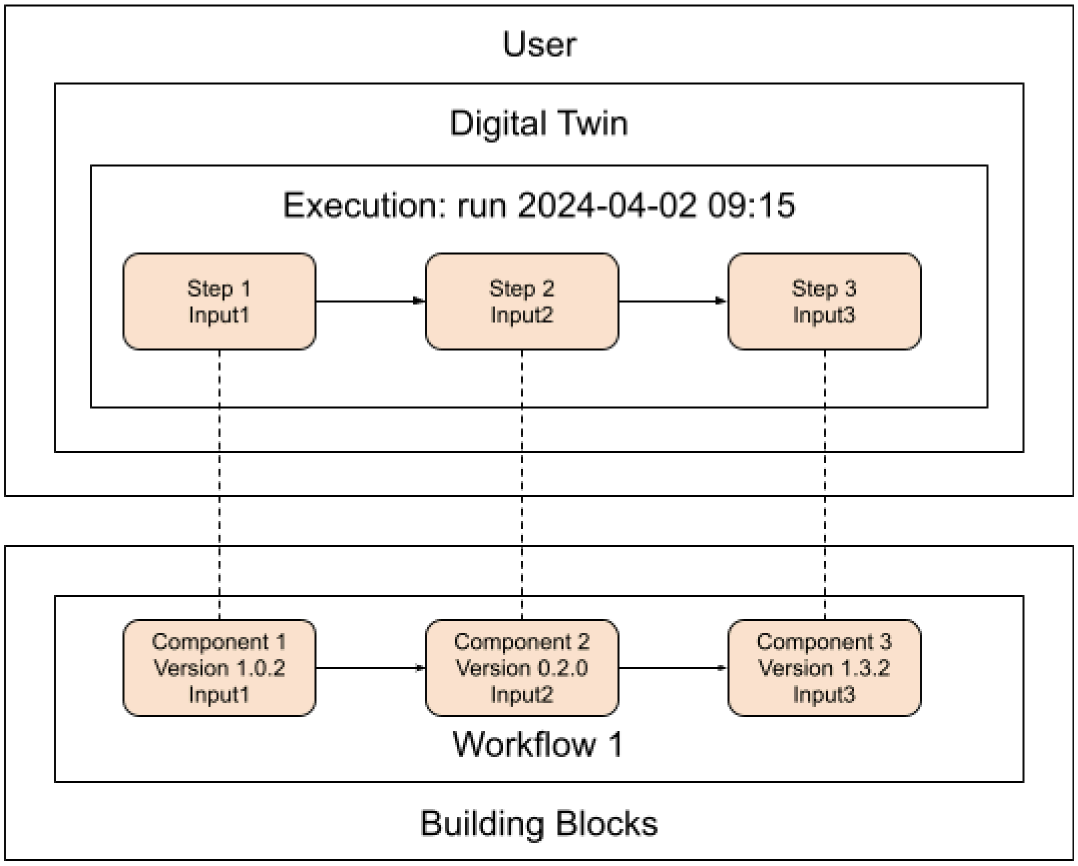
User
Digital Twin
Execution: run 2024-04-02 09:15
Step 1 Input1
Step 2 Input2
Step 3 Input3
Component 1 Version 1.0.2 Input1
Component 2 Version 0.2.0 Input2
Component 3 Version 1.3.2 Input3
Workflow 1
Building Blocks

Figure 14: Governance of Digital Twins

Execution of Digital Twins

The following classes allow to run and monitor executions:

- Executions: Execution have already been mentioned in the Governance
- Steps: Steps in an Execution correspond to the version of an ODTP-1 Component: that is the component that runs as a Docker container in this step
- Outputs: Outputs are the output of each step

- Results: Results belong to a Digital Twin and can combine the outputs of several executions of that Digital Twin.

Executions

A digital twin shall be produced in an ODTS-3 Orchestrator by executing an ODTS-2 workflow. The ODTS-3 Orchestrator shall create an execution by instantiating and running the ODTS-2 Components according to the ODTS-2 workflow. The orchestrator should be able to copy executions and make them reusable to be rerun with minor adjustments.

Executions shall have the following mandatory properties:

- Workflow template: ODTS-2 Workflow template on which the execution shall be based or an acyclic graph of ODTS-1 components
- Title: Title of the execution
- Description: Description of the execution
- Start Timestamp: The time when the first Execution step started to run
- End Timestamp: The time when the last step of the execution stopped running
- Logs: A summary of all logs belonging to the execution as printed out by each component in the workflow

Steps

A Step is the execution of a single component. The ODTS-3 Orchestrator shall maintain these mandatory properties of Steps:

- Component Version: The version of the component that is run in this step
- Start Timestamp: The time when the step started to run
- End Timestamp: The time when the step stopped running
- Logs: Reference to a Log Object that stores the logs of the step
- Output: Reference to an output object
- Parameters: JSON Object of Parameter Keys and Values as derived from the component
- Metrics: Reference to a Metrics Object that stores the metrics obtained from running a step

Output

The orchestrator shall store the output of each Step. Intermediate outputs shall be stored in the ODTS-3 Orchestrator’s data storage. The output of a Step shall be described with these Mandatory properties:

- Output Type: Snapshot (in the data storage) or file output (in the project directory)

- S3_snapshot:
  - Bucket: in the orchestrator's data storage
  - Key: to access the bucket
- File:
  - Name: File name for the output
  - Size: Size of the output
  - File type: Type of the output

Result

The result shall provide a combined output of selected executions within a digital twin. The result shall allow the comparison of the outputs of several executions. This provides the ability to compare the performance of different digital twin formulations.

Results provided by the ODTS-3 Orchestrator shall have these mandatory properties:

- Title: Title of the result
- Description: Description of the result
- Digital Twin: Reference to the Digital Twin
- Executions: References to the Executions that the result relates to
- Outputs: References to the different outputs. These shall be the outputs of all executions.

Authentication

Users

Users shall create Digital Twins using ODTS-2 Workflows or ODTS-1 Components. The ODTS-3 orchestrator needs to support user-owned digital twins that contain the executions and the data produced by running the executions of a Digital Twin. The ODTS-3 Orchestrator should offer user authentication and authorization and use secure methods to verify user access.

The ODTS can also provide team ownership. This feature allows collaboration on Digital Twins by granting access and permissions to designated team members and fosters teamwork and knowledge sharing within a project. It ensures data from different projects remains separate and secure. It is crucial for maintaining confidentiality and preventing accidental data leakage.

Authorization

The ODTS-3 orchestrator shall have two categories of users:

- Normal users who access non-sensitive data
- Authorized users who access non-sensitive and sensitive data

OpenID Connect (OIDC) is an industry-standard authentication and authorization protocol that can be leveraged by ODTS for user identification and authorization. The OpenID Connect plugin allows the use of bearer access tokens to verify user identities before granting access to the orchestrator. The ODTPS orchestrator shall implement an identity and access management (IAM) supporting the OIDC protocol, such as Keycloak for:

- User Management: handles user registration, login, and profile management.
- Authentication: verifies user credentials and grants access tokens.
- Token Management: issue and manage tokens used for ODTS-3 orchestrator usage

Tool Adapter

The ODTS-3 Orchestrator shall provide a recipe on how tools can be turned into ODTS-1 components in such a way that they will be compatible with the ODTS-3 Orchestrator. An orchestrator may provide a component template that can be copied and contains further instructions on how to get from the provided template to a component for the tool, see ODTP as an example of how that can be done.

Client Library

The ODTS-3 Orchestrator and the ODTS-1 Components shall communicate via a client library consisting of functions and methods provided by the orchestrator to the components. Any ODTS-1 Components shall install this library when instantiating a microservice to communicate with the ODTS-3 Orchestrator. The client library is orchestrator specific. The client library shall implement the following features:

- Writing the logs
- Storing and retrieving the operational metadata
- Starting the app script of the component

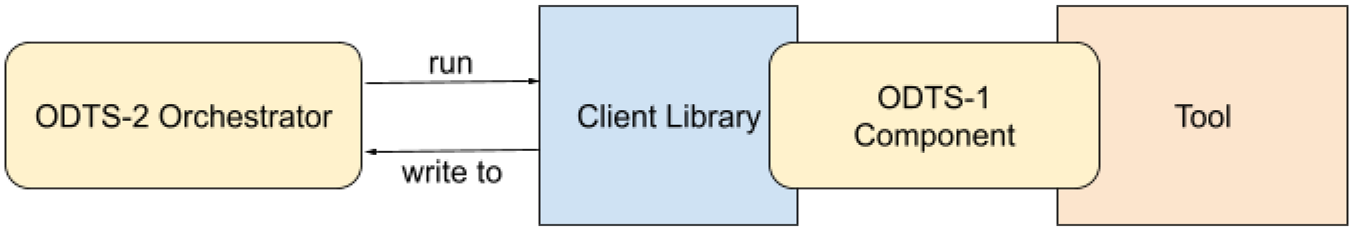

Figure 16: The ODTS-1 Component installs the Client Library to communicate with an ODTS-3 Orchestrator. The component runs the tool in a Docker container.

ODTS-4 Zoo

An ODTS-3 Orchestrator should be able to discover ODTS-1 components proposed for an ODTS-2 Workflow in a registry called an ODTS-4 Zoo. An ODTS-4 Zoo is coupled to one or more ODTS orchestrators and shall ensure that the ODTS-1 components and ODTS-2 Workflows that it registers are interoperable in the context of these ODTS-3 Orchestrators. The zoo consists of the following parts:

- **Repository**: A repository that contains the component /workflow registry and makes them available via an index
- **Registration Method**: A recipe on how to register and unregister components / workflows from the zoo.
- **Metadata Specification**: This specifies the metadata that are expected for the registration of a component / workflow

Repository

The ODTS-4 Zoos shall consist of a repository that lists the ODDS-3 Orchestrator(s) with which its components are interoperable. Compliance with these orchestrators shall be guaranteed for all components it registers. The index may be provided as an `index.json` or `index.md`. The zoo may choose to register only ODTS-1 Components or ODTS-2 Workflows. The ODTS-4 zoo must provide the index as a list. It should provide a search by tags (see metadata for components below).

Registration Method

The zoo shall offer a method to submit a component or workflow and add it to the index. The method to add a component or workflow shall be documented for submitters and shall guarantee compliance with the orchestrators by implementing a review process before submission. Pull Requests may be used as a submission method and to unregister components or workflows.

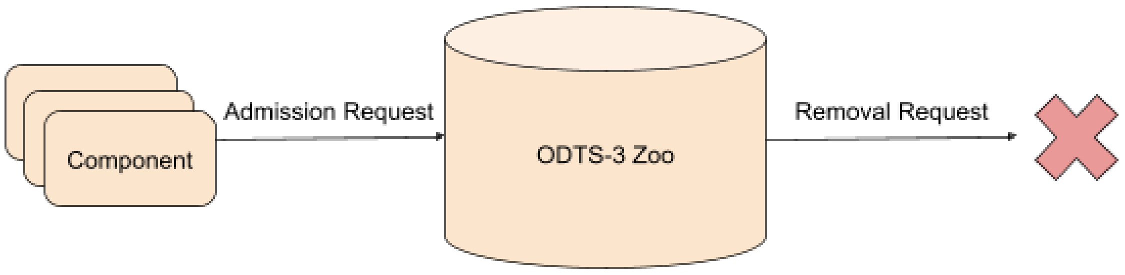

Figure 17: Components are added to and removed from the zoo via requests.

Metadata for Components

The ODTS-4 Zoo shall reuse the metadata of the ODTS-1 Component. The metadata file of the component in the zoo shall be provided in a yaml or json file with a standardized name, such as `odts.yml` or `odts.json`.The ODTS-4 Zoo shall expect these metadata for the ODTS-1 Component:

Required:

- Name: Component name
- Authors: Authors of Component (may differ from tool authors)
- Version: Version of the Component, see [Versioning] for how it is coupled with the tool version
- Repository Tool: Repository of the Tool
- Repository Component: Repository of the Component
- Component License: License of Component: it shall be compatible with the license of the tool
- Type: Type of Component
- Description: Short description of the Component

Recommended:

- Tags: A list of tags describing the component
- Ports: ports shall be provided for interactive components
- Parameters with Default and Data Type for Parameters that should be exposed
- Configuration Files: a list of further configuration files with descriptions, see [Parameter files] and [Metrics files]
- Data Inputs: a list of file inputs with type, path and description
- Data Outputs: a list of file outputs with type, path and description, see [Semantic Input and Output File]
- Schema-Input: Path to input schema for automatic input validation, see [Semantic Input and Output File]
- Schema-Output: Path to output schema for automatic output validation
- Devices: indicate whether a GPU is needed

Metadata for Workflows

For the registration of ODTS-2 Workflows in the ODTS-4 Zoo, workflow files shall be provided in a YAML format, specifying Acyclic Graphs of Components and including the versions of these components.

Appendix A ODTP

The reference implementation of the ODTS: A tool designed to generate specific digital twins by integrating into a platform how to manage, run, and design digital twins. It offers an interface (CLI, and GUI) for running and managing digital twins. It wraps different open source technologies according to ODTS to provide a high level Application Programming Interface (API) for the final user. The reference implementation may not implement all features of the standard but aims to do so in the long-run. The reference implementation can be found here: https://github.com/odtp-org/odtp. As such, ODTP is a Proof-Of-Concept of the ODTS with the focus on specific digital twins in mobility context. It was built in order to establish the ODTS Standard via a POC in collaboration between [CSFM] and [SDSC].

In the following section we list the alignments and differences between ODTS and ODTP: therefore this sections structure aligns to ODTS:

- **ODTP Components:** ODTP components are very aligned to ODTS-1 Components.
- **ODTP Workflows:** the concepts of ODTS-2 Workflows have not been implemented in ODTP yet.
- **ODTP Orchestrator**: The ODTP orchestrator is aligned to the ODTS orchestrator except that it does not support X.
- **ODTP Zoo**: the zoo has been planned but not yet been implemented and is also not connected to the Orchestrator yet.

ODTP Components

ODTP Components served as role models for the concept of ODTS-1 Components. In this section the implementation of ODTS by ODTP comes very close. The [versioning] as described in ODTS has not yet been implemented. Also the [API Components] are not yet supported. But both topics are on the roadmap and will be available in releases that have already been planned.

See here for an example components in the context of ODTP:
https://github.com/odtp-org/odtp-component-example

ODTP Workflows

The concepts of ODTS-2 Workflows and Workflow Templates have not been implemented in ODTP. But the need for them has surfaced through the user experience with the ODTP orchestrator:

- Executions of workflows are often very similar, so that it makes sense to add templates for them, so that they can be created with more ease.

- The Combination of Components into Workflows is not arbitrary, but mostly components have been built for certain use cases that can be translated into workflows of components. Therefore it makes sense to register those established workflows once they have been tested and proven to work.

Therefore ODTP Workflows are on the roadmap of ODTP and will be implemented in an upcoming release.

ODTP Orchestrator

The ODTP Orchestrator aligns to the ODTS-3 Orchestrator and implements an orchestrator for the ODTP Components. It offers a command line interface and graphical user interface. It is still a POC so even some mandatory features of ODTS have not been yet implemented there. ODTP was also used to evaluate beyond what itself currently offers the demands and needs for the digital twin orchestration. It thus directly enabled the formulation of the ODTS.

Features

The features that have been implemented from ODTS-3 Orchestrator are the following:

- **User Interface:** both GUI and CLI are available as interfaces
- **Usage of Docker:** the usage of Docker is exactly as described in ODTS-3 Orchestrator
- **Data Storage:** has been implemented as described in ODTS-3 Orchestrator
- **Operational Data**: operational data for workflows and workflow templates have not yet been added, also the results in ODTP are not yet adjusted to results in ODTS-3, that span over several executions. But the need for this has been identified and it will be implemented in ODTP the way it has been suggested in ODTS-3
- **Tool Adapter**: implemented as suggested in ODTS-3 by proving a component template with instructions: https://github.com/odtp-org/odtp-component-template
- **Client Library**: implemented as in ODTS-3 by providing a github repo that can be mounted via git submodules in on the components, see: https://github.com/odtp-org/odtp-component-client and https://github.com/odtp-org/odtp-component-example

Features on the roadmap are:

- **Authentication:** planned with keycloak as suggested in ODTS-3
- **Semantic Validation:** it will be possible to automatically check outputs and inputs as described in ODTS-3 [Semantic Validation]

Technologies

The orchestrator for ODTP is implemented with python using the following technologies:

- Dashboard (Nicegui)
- Command Line Interface (Typer)
- Snapshots/Data transferring (MINION S3)

Not yet implemented but planned in one of the next releases:

- Authentication (eduID, GH)
- Semantic Input and Output Validation, see [Semantic Input & Output Schema]
- Workflow manager (Barfi): to create Acyclic Graphs as workflows
- License manager: to check for the compatibility of component licenses
- Additional Data governance (for example an integration with the Swiss Data Custodian
- Performance Logging (Grafana)
- KG/Ontology storing (GraphDB)

ODTP zoo

A zoo has been implemented with examples from the mobility sector: ODTP-org Zoo. For practical purposes, the zoo also has a web frontend: ODTP zoo.

The zoo gives its README a recipe on how to add a component to the zoo. Components are added by PRs and they are also removed by PRs. At all times the repo offers an `index.json` file of registered components. There is a github action implemented that add the components metadata from a `odtp.yaml` file to the index.Then an static webpage hosted in GitHub pages gets updated.

Appendix B Eqasim/Matsim

We provide a workflow for a digital twin to create synthetic populations (Hörl & Balać, 2019) and run MATSim-based mobility simulation of three scenarios: Ile de France, Corsica, and Switzerland:

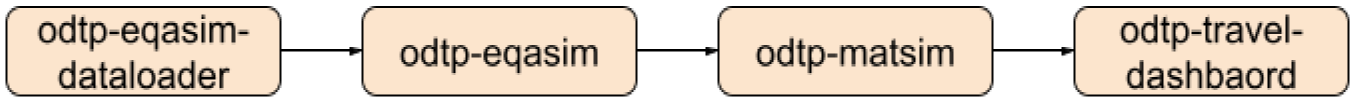

Figure 18: Overview of Eqasim workflow in ODTP.

The data loader component prepares statistical data on the population of the respective scenario and the geographic data from standardized data sources. The French scenarios are publicly available, the Swiss scenario requires a valid contract with the Swiss Federal Statistics Office (FSO).

Overview of the Components:

- Odtp-eqasim-dataloader The data loader component prepares statistical data on the population of the respective scenario and the geographic data from standardized data sources. The French scenario is publicly available, the Swiss scenario requires a valid contract with the Swiss Federal Statistics Office (FSO).
- Odtp-eqasim The eqasim component generates a synthetic population based on statistical data and links them with travel profiles according to statistical properties.
- Odtp-eqasim-matsim The MATSim component uses the synthetic population to generate transport simulations.
- Odtp-travel-data-dashboard The travel data dashboard visualizes the Origin-Destination data output of the MATSim simulation to communicate mobility patterns.

Reference:

Hörl, S., & Balać, M. (2019). Eqasim: An open-source and extensible platform for building agent-based models. *NSL Newsletter*, *44*.

Appendix C Mobility Causal Interventions

We provide another workflow for a digital twin that implements the mobility causal intervention framework to evaluate the robustness of deep learning models towards data distribution shifts, with the application of individual next location prediction (Hong et al, 2023):

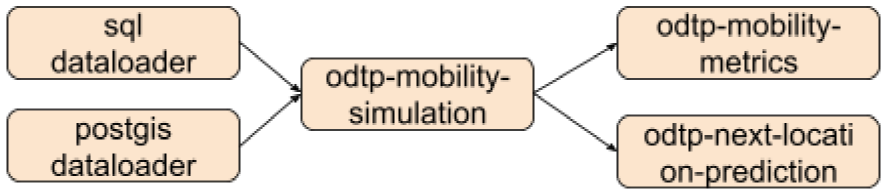

Figure 19: Overview of the mobility causal intervention workflow in ODTP.

The mobility simulation module is used to generate individual location sequences. It also incorporates the causal intervention mechanism to generate intervened synthetic data that represent different data distribution shifts. These synthetic data are fed into the next-location-prediction module to quantify a model's robustness against interventions. Meanwhile, a mobility-metrics module is used to monitor the change in the characteristics of mobility data.

Overview of the components:

- odtp-sql-dataloader: This component performs SQL queries to a compatible database and create a dataframe output in csv format.
- odtp-postgis-dataloader: This module performs postGIS SQL queries to postgresql databases containing and provides the output in csv format.
- odtp-mobility-simulation: This module generates synthetic individual location visit sequences based on mechanistic mobility simulators (including EPR, IPT, Density-EPR, and DT-EPR models). The module also generates intervened synthetic mobility data based on causal interventions through the specification of parameters to be intervened and levels of the interventions.
- odtp-mobility-metrics: This module includes multiple metrics to quantify the characteristics of individual mobility sequences, e.g., location visitation frequency, radius of gyration, real entropy, mobility motifs etc.
- odtp-next-location-prediction: This module includes two deep learning models for individual next location prediction, the LSTM model and the Multi-Head Self-Attentional (MHSA) model.

Reference:

Hong, Y., Xin, Y., Dirmeier, S., Perez-Cruz, F., & Raubal, M. (2023). Revealing behavioral impact on mobility prediction networks through causal interventions. *arXiv preprint arXiv:2311.11749*.