FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

ASSIGNMENT OF MASTER'S THESIS

Title:	Orchestration and Monitoring of Manta Flow Processes
Student:	Bc. Petr Gondek
Supervisor:	Ing. Michal Valenta, Ph.D.
Study Programme:	Informatics
Study Branch:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2020/21

Instructions

The aim of the thesis is to design and implement a tool for orchestrating and triggering processes in Manta Flow software and their monitoring. The tool will have a graphical user interface.

Follow these steps:
1. Analyze Manta Flow processes and design an API that allows you to run and monitor processes.
2. Analyze orchestration needs and design a configuration description for more complex process orchestration.
3. Use wire-frame to design a graphical interface for starting and monitoring processes.
4. Implement the prototype as a web application.
5. Test and document the prototype properly.

References

Will be provided by the supervisor.

Ing. Michal Valenta, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague November 5, 2019

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Orchestration and Monitoring of Manta Flow Processes

Bc. Petr Gondek

Department of Software Engineering
Supervisor: Ing. Michal Valenta, Ph.D.

July 27, 2020

Acknowledgements

First of all, I would like to thank MANTA and its employees for creating a pleasant collective and allowing the creation of a thesis under their auspices. Then, above all, to Jakub Moravec for his valuable advices throughout the project that I would have been hard-pressed to finish the thesis without.

Further, I would like to thank the Czech Technical University in Prague, Faculty of Information Technology and its professors for great opportunities and preparing me for my career life. Especially to my supervisor Michal Valenta whose insight and knowledge helped me to finish this thesis. He was always kind and willing to help.

Last but not least, I would like to thank my family, friends and Šárka Weberová for their support, patience and encouragement during studies.

Declaration

I hereby declare that I have authored this thesis independently, and that all sources used are declared in accordance with the "Metodický pokyn o etické přípravě vysokoškolských závěrečných prací".

I acknowledge that my thesis (work) is subject to the rights and obligations arising from Act No. 121/2000 Coll., on Copyright and Rights Related to Copyright and on Amendments to Certain Laws (the Copyright Act), as amended, (hereinafter as the "Copyright Act"), in particular § 35, and § 60 of the Copyright Act governing the school work.

With respect to the computer programs that are part of my thesis (work) and with respect to all documentation related to the computer programs ("software"), in accordance with Article 2373 of the Act No. 89/2012 Coll., the Civil Code, I hereby grant a nonexclusive and irrevocable authorisation (license) to use this software, to any and all persons that wish to use the software. Such persons are entitled to use the software in any way without any limitations (including use for-profit purposes). This license is not limited in terms of time, location and quantity, is granted free of charge, and also covers the right to alter or modify the software, combine it with another work, and/or include the software in a collective work.

In Prague on July 27, 2020 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology
© 2020 Petr Gondek. All rights reserved.
This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis

Gondek, Petr. *Orchestration and Monitoring of Manta Flow Processes.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020.

Abstrakt

Tato diplomová práce popisuje problematiku spouštění mnoha dlouhotrvajících procesů. Zabývá se možnostmi orchestrace takových procesů v kontextu firmy MANTA a část práce je věnována také jejich plánování s využitím možností paralelizace. V neposlední řadě popisuje možnost monitorování a sledování průběhu spuštěného plánu a scénářů.

Klíčová slova MANTA, MANTA CLI, procesy, spouštění, paralelizace, monitorování, plánování, orchestrace

Abstract

This master thesis describes problematics of execution a lot of long-running processes. It describes the possibilities of orchestrating such processes in the context of MANTA, and part of the work is also devoted to planning processes using parallel running capabilities. Last but not least, it details the possibility of logging and monitoring the Workflow and Scenario progress.

Keywords MANTA, MANTA CLI, processes, execution, parallelisation, monitoring, progress, planning, orchestration

Contents

List of Figures

List of Tables

Chapter 1

Introduction

One of the glaring shortcomings of the MANTA flagship product is the often recurring problems users face in creating their own workflows and monitoring their execution progress. MANTA, as software for creating data lineage from customer's systems, uses over a hundred Scenarios to achieve this goal. Each such Scenario takes care of a task, such as extracting metadata from a customer's system or analysing this obtained metadata. Together, these Scenarios create the data lineage, which users may observe in a web application. Using this view, they can more easily study their data flow at their company, finding potential issues and learn important aspects from them on which they can, for example, perform optimisation of these systems.

These Scenarios are executed as separate processes, and shell and batch scripts are now used for their orchestration. Such a modification of a relatively simple script can be quite a challenge not only for an IT inexperienced user, as Scenarios have a fixed execution order, and rearranging them can cause the system to crash and fail to get the desired results.

As a company grows, its systems grow with them and extracting metadata from large systems as MANTA customers may take a long time. Actually, some customers build their data lineage over days. Tracing log generated by a console application may be annoying, and users may lose awareness of what has been executed and what is still waiting to run. Part of the problem with too long Scenario executions is that the running and parallelisation of Scenarios are not 100% effective, due to the use of so-called Master Scenarios.

This thesis is about problematics of execution a lot of long-running processes. Chapters describe the possibilities of orchestrating such processes in the context of MANTA, and part of the work is also devoted to planning processes using parallel running capabilities. Last but not least, it details the possibility of logging and monitoring the execution process and Scenarios.

1.1 Thesis Structure

Thesis is structured as follows:

- Chapter 2
  - informally introduces thesis goals,
- Chapter 3
  - provides the detail description of Manta ecosystem,
  - defines functional and non-functional requirements,
  - analyses existing solutions,
- Chapter 4
  - introduces new business processes,
  - explains application architecture,
  - shows graphical user interface and API,
  - details component architecture,
  - presents data persistence with database model,
- Chapter 5
  - describes implemented prototype,
  - characterises used algorithms,
  - states testing and documentation.

Chapter 2

Thesis Goals

The thesis aims to create a software product to make work with MANTA CLI for users more comfortable. It would help them run Scenarios, allow them to create their customised workflows more easily and efficiently so that users are not reliant only on a few pre-prepared workflows from MANTA, and improve parallel run capabilities and thus speed up the process.

To achieve these goals, the entire MANTA ecosystem must first be analysed. Find out what the MANTA CLI is, what the Scenarios are, how the Scenarios work, what the Scenarios can preform, what the Scenarios depend on, how their configurations work, how the Scenarios are currently triggered, how they are currently orchestrated. Also, the analysis must reveal hidden characteristics and limitations of the existing execution to avoid the risks that might arise in implementing a new run.

After finding out enough information about MANTA CLI and Scenarios, new orchestration options need to be proposed. New Workflow format must be devised with its execution and configuration possibilities. Current customisation options need to be taken into account and ideally preserved to the same extent. The solution must be prepared for possible software updates, so the design must include user data migration - backward compatibility between versions.

Then an API must be designed through which users can easily use the system's services, such as starting Workflow and monitoring its running.

However, ordinary users may not be able to exploit the full potential of the API; thus, a graphical user interface of the web application through which the services can be used also needs to be prepared. The design must be user-friendly and not burdensome and will be designed using wire-frames.

At last, a prototype will be implemented, tested, and software documentation will be created.

Chapter 3

Analysis

This chapter describes MANTA and its product MANTA, what MANTA does, what is data lineage, history of the company, product architecture, what is MANTA Server and CLI, what is MANTA Service Utility. Then, there is described requirements gathering, requirements analysis, analysis of existing solutions and illustration of current processes. And at last, there is a high-level brief of the proposed solution, the new architecture, new activities and processes using the new tool.

3.1 MANTA Ecosystem

MANTA is a tool which automatises data lineage creation (can be seen in figure 3.1) by analysing metadata from report definitions, custom SQL code, and extract-transform-load (ETL) workflows.[4][5] "It can to cope with SQL, altogether with various of its sub-dialects, ETL and reporting tools, and programming languages like Java. The uniqueness of the software product is in its capability of handling code that is hardly readable by a human. Thanks to this feature, MANTA can automatically process databases consisting of millions of records and create a map of data flow across the business intelligence environment - data lineage. Alternatively, the data flow is not visualised directly by MANTA but cooperates with third-party data governance solutions like Informatica, TopQuadrant, Collibra, and IBM IGC." [4]

"MANTA's history began back in 2008 when it was an internal tool of the Czech consulting company Profinit. After a few years of enhancements and adding new features, MANTA won the Czech ICT Incubator @ Silicon Valley competition held by the Czech ICT Alliance. Shortly thereafter, in late 2016, it became an independent company and opened their first office in San Francisco. Then, it opened another one in New York City in 2018 and moved the HQ there a year later. With an office in the US and a development center in Prague, MANTA has become the first-of-its-kind central hub of all data flows that allows information users to understand the where, how, and what of

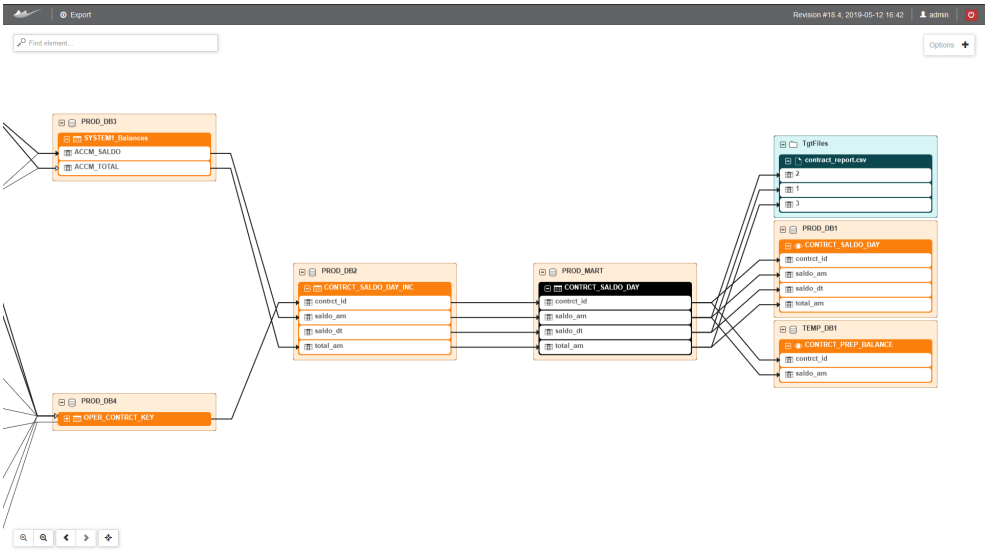

Figure 3.1: MANTA Data Lineage Visualisation [1]

their information assets. MANTA currently operates on a global level through their own offices and an extensive network of partners. It help customers from sectors and industries of all kinds since there's no such thing as having too good of an understanding of what is happening with an organisation's data. Their customers include Teradata, SCP Health, OBI, and many more." [6]

3.1.1 Data Lineage

"Data Lineage or Data Provenance describes data origins, movements, characteristics, and quality. According to Stewart Bond, Data Lineage has typically described where the Big Data begins and how it is changed to the final outcome." [7]

"It is like telling the story of a piece of data, including where does it come from, what transformation it undergoes, and how it interacts with other data. It should provide answers to questions such as where the data in a given solution come from, whether it can be trusted or not, how it gets from point A to point B, and how the data changes over time in the analysed system. Data lineage helps enterprises to gain more in-depth knowledge and understanding of what happens to data as it travels through various interconnected data pipelines that their systems contain." [4]

"Data lineage is important to data quality measurement." [8] "From data-quality and data-governance perspectives, data lineage ensures that existing business rules exist where expected, calculation rules and other transformations are correct, and system inputs and outputs are compatible. Data traceability is the actual exercise to track access, values, and changes to the data

as they flow through their lineage. Data traceability can be used for data validation and verification as well as data auditing." [9]

3.1.2 MANTA Architecture

MANTA consists of three separate components as can be seen in figure 3.2. Two core components MANTA CLI and MANTA Server are responsible for creating data lineage. The third supporting component MANTA Service Utility, which consists of two modules named MANTA Updater and MANTA Configurator, helps users to update and configure the MANTA.[10]

During writing this work[1], MANTA Service Utility is being rewritten to a new application called MANTA Admin UI. The product of this thesis, with the working name MANTA Process Manager, will be a module for new MANTA Admin UI. Simultaneously with this module emerges another tool MANTA Log Viewer which will also be part of MANTA Admin UI. Alongside these all components, there is also another tool named MANTA Installer which installs MANTA and updates the MANTA Service Utility.

This thesis will use shorten names for all these components and modules from this point further to achieve better readability. The word 'MANTA' in names will be left out, but the reader must keep in mind that their shortened name references to its only correct full name containing preposition 'MANTA'.

MANTA CLI

MANTA is implemented as a client-server Java application. CLI is the client which takes care of the main execution plan, which consists of Extraction, Analysis and Export. In the extraction Phase Dictionaries, Scripts, ETL or reporting content are extracted from specified database systems. These extracted data are analysed in the second Analysis Phase altogether with user's SQL scripts. Result of the Analysis is a graph which is posted to the Server where it is stored. This storing operation is named Merge and it creates new Revision (can be seen in figure 3.3). The graph can be later downloaded and exported to the third-party software, and this is the last Phase called Export. [5][11][10]

CLI consists of Scenarios where every Scenario represents a particular task for creating the final graph. These Scenarios are logically divided into three parts representing Phases (Extraction, Analysis, Export), and they are also divided by the technology they serve. There is silently used a naming convention, but it is not guaranteed. A simple plan can look like in figure 3.4 where the Oracle technology was used as an example. There can be seen highlighted Extraction Scenarios (green), Analysis Scenarios (orange) and an example of export Scenarios. The Analysis contains many Scenarios, but only

[1] first half of 2020

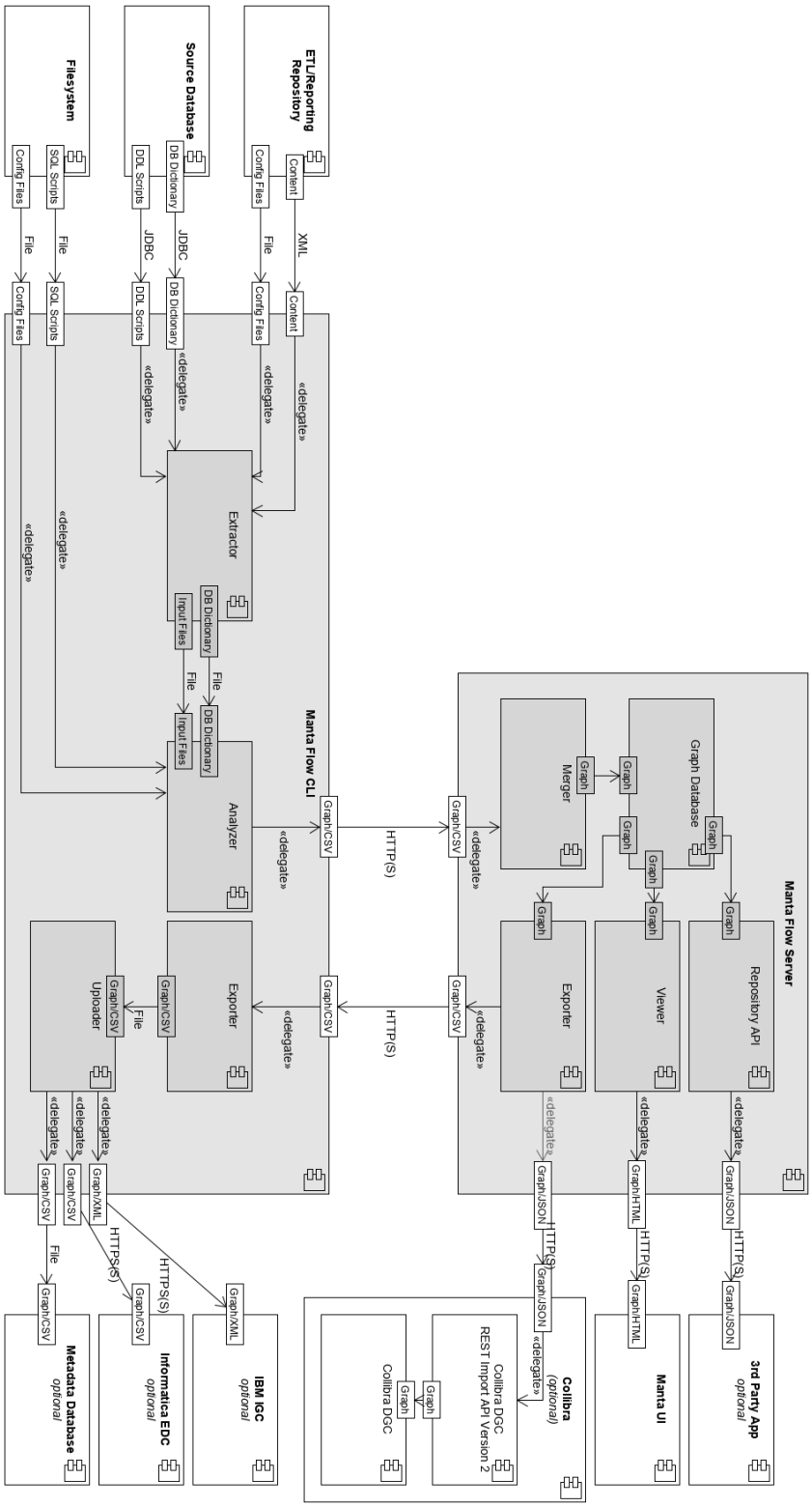

Figure 3.2: MANTA Architecture Diagram [2]

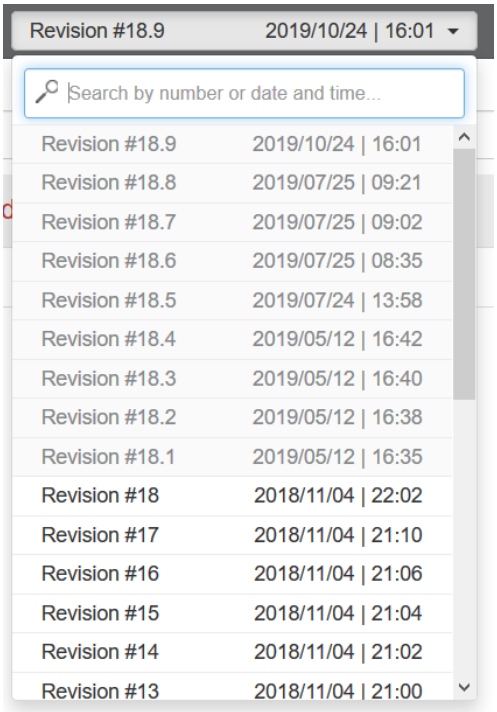

Figure 3.3: Revisions in MANTA Server [1]

a few are tied up with the technology. The rest only performs necessary tasks to maintain the execution environment, e.g. 'Revision Create' and 'Commit'.

Every Scenario is at the end, the own process instantiated by MANTA Platform. These processes are executed sequentially, and this could be very time consuming if there wouldn't be any parallelism. That is why Master Scenarios were introduced. Master Scenarios handle parallelism in the context of one particular technology. So for example, if there is more than one Connection for a technology, then Extractions may be executed in parallel. But this solution is not optimal. Almost all Extraction may run in parallel because databases usually correspond with their own machines and computing resources, so the computing load is distributed, and the process may speed up. However, this is difficult to achieve by current architecture.

The original architecture consists of many `.sh` and `.bat` Scenario scripts. These scripts execute `mantar.bat` script which runs `java` command with MANTA Platform `jar` and appropriate arguments. These Scenario scripts are called from parent scripts logically divided by their Phase. From snippet 3.1 can be seen that Scenario scripts are executed sequentially, and they wait for each other to finish before starting a new one.

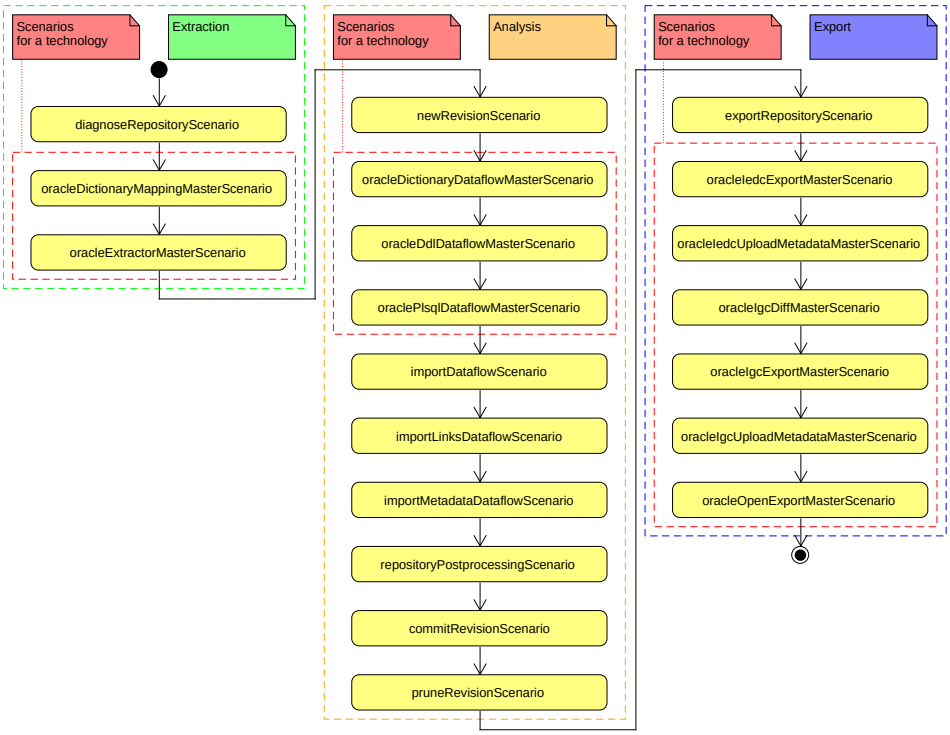

Figure 3.4: Example of Scenarios Execution with all Phases and Oracle Technology

MANTA Server

"MANTA Server is a server application written in Java. The purpose of the server is to store ('merge') in optimal form all the metadata received inside the so-called metadata repository in the form of a graph database. This data will then be transformed by the exporter to export to a third party application, or prepared for display in the web user interface."[2] [10]

MANTA Service Utility (Admin UI) and Installer

Service Utility is a set of tools assisting users to configure and maintain MANTA. This utility consists of two main applications: Configurator and Updater. Configurator helps users to configure Connections to third party systems like Data Governance Tools or database machines. These Connections are stored in CLI and used by Scenarios to extract or export metadata. Updater, on the other hand, was created to help users merge their customised

[2] author's translation

```
8  ...
9  call "\%~dp0.\oracleDictionaryDataflowMasterScenario.bat"
10 call "\%~dp0.\oracleDdlDataflowMasterScenario.bat"
11 call "\%~dp0.\oraclePlsqlDataflowMasterScenario.bat"
12
13 call "\%~dp0.\teradataDictionaryDataflowMasterScenario.bat"
14 call "\%~dp0.\teradataDdlDataflowMasterScenario.bat"
15 call "\%~dp0.\teradataBteqDataflowMasterScenario.bat"
16 call "\%~dp0.\teradataTptDataflowMasterScenario.bat"
17
18 call "\%~dp0.\mssqlDictionaryDataflowMasterScenario.bat"
19 call "\%~dp0.\mssqlDdlDataflowMasterScenario.bat"
20 call "\%~dp0.\mssqlTsqlDataflowMasterScenario.bat"
21
22 call "%~dp0.\hiveDictionaryDataflowMasterScenario.bat"
23 call "%~dp0.\hiveDdlDataflowMasterScenario.bat"
24 ...
```

Listing 3.1: Snippet from Analyze parent script

configurations with new changes in these configurations during an update. It is capable of updating Server and CLI without destroying users' data.

These two tools are currently[3] being rewritten into a new tool with working name MANTA Admin UI using modern technologies like a React JS. And these two tools will be enriched by two new tools: MANTA Log Viewer [12] and this tool MANTA Orchestration & Monitoring.[4] This business decision, made by MANTA team, lays down limitations for technologies that can be used for development.

Together with these tools, a MANTA Installer was created. It assists users to installs MANTA into customers machines, and it is also responsible for updating Service Utility. This creates a challenge for this thesis because Updater cannot be used for merging user's configuration as same as for Server and CLI.

3.2 Requirements Gathering

This section describes what requirements for Orchestration & Monitoring tool are, what use-cases are and what are typical users. This information was gathered in several meetings and discussion with technical supervisors represented by Lukáš Hermann and Jakub Moravec and later verified from the customer scope by the ordering party represented by Jan Ulrych and Ernie Ostrich.

[3] between years 2019-2020

[4] sometimes referenced by its original working name Process Manager or Process Monitor

3.2.1 Use-Cases

This chapter describes the use-cases for MANTA Monitoring & Orchestration tool. Use-Case Diagram 3.5 helps to define functional requirements and gives an overview of the usability and behaviour from the user perspective. The shortcut `U#` is the identifier of particular use-cases in the Use-Case diagram. The shortcut `F#` is the identifier of a particular Functional Requirement and `T#` is the identifier of a particular Technical Requirement.[13]

There are three persons pictured in the use case diagram. These persons represent business roles which may be used for Orchestration and Monitoring as access roles.

The first role with minimal privileges is Viewer. This person could be a woman or man in a company who is using data lineage daily for his work, e.g. data architect upkeeping their data warehouse or a member of decision management preparing a report for his colleague. He or she can use this tool to check if the data lineage he is currently using is up to date and from which day it is. However, if she or he cannot find its lineage, then this person may look into the last Execution to see what happened and if all required systems were analysed correctly.

The Viewer may tell the Executor that his data lineage is not up to date because a necessary change happened during last week and he or she may request to start a particular workflow which will provide him/her newer and reliable lineage. The Executor may be a member of dev-ops team, and his or her responsibility may be to plan Workflow executions to not overload targeted systems. Alternatively, the Executor may be a robot or another system executing workflows using API.

The last role is the Administrator. This person is someone who has participated in MANTA training program and has learnt about how MANTA works, what every Scenario does, how to create custom Workflows, how to optimise Workflows. He or she has enough knowledge to let him or her edit Workflows without need to worry that she or he breaks the Workflow definition, and his or her co-workers will not have their data lineage next working day.

U1: Create Workflow

Users can create Workflow in a graphic user interface, with basic knowledge of how Scenarios work. The user interface must guide and help users to prepare the desired Workflow to gather their metadata and create data lineage. They can work in levels of complexity, e.g. at the first level, they must know only basics - what Phases are, or they may use prepared templates. The most experienced users should be able to make use of knowledge of every Technology Scenarios and all Processing Scenarios to create particular Workflows to meet their desires. What Workflow does, how it is configured and defined, is called

'Workflow definition'. This replaces workflows in Scenario Scripts.[5]

There should be an option to create Workflows with wildcards, e.g. 'all Scenarios for all Oracle Connections', and so.

There should be an option to create one-scenario specific Workflows, e.g. 'DictionaryDataflowScenario for Connection A in Technology Oracle'.

U2: Update Workflow

It must be possible to change the existing Workflow definition later using the same or similar graphic user interface with similar or same guidance as during creation. There should be no limit for updating existing Workflows, what could be done during creation must be achievable during update too.

U3: Delete Workflow

User must have an option to delete existing Workflows. The deletion must be confirmable. Restoring deleted Workflow is not necessary.

U4: Import Workflow

User should have an option to Import Existing workflow from file or via REST API. Workflow definition backward compatibility should be achieved if possible.

U5: Export Workflow

User should be able to export his creations from the graphic user interface into the file. He can do it during creation in Workflow editor or Workflow detail.

U6: Execute Workflow

Existing Workflow can be executed as defined by the user. Execution uses Scenarios from CLI. This replaces execution from Scenario Scripts. Execution may use MANTA Platform.

U7: Terminate Workflow

User has an option to terminate a running Scenario. Some of the Scenarios cannot be easily terminated because special terminating Scenario must be called. Terminating Workflow may break the Metadata Repository on the server, and a correct recovery approach must be chosen[6].

[5] *.bat* and *.sh* scripts in CLI

[6] ignore, commit or rollback

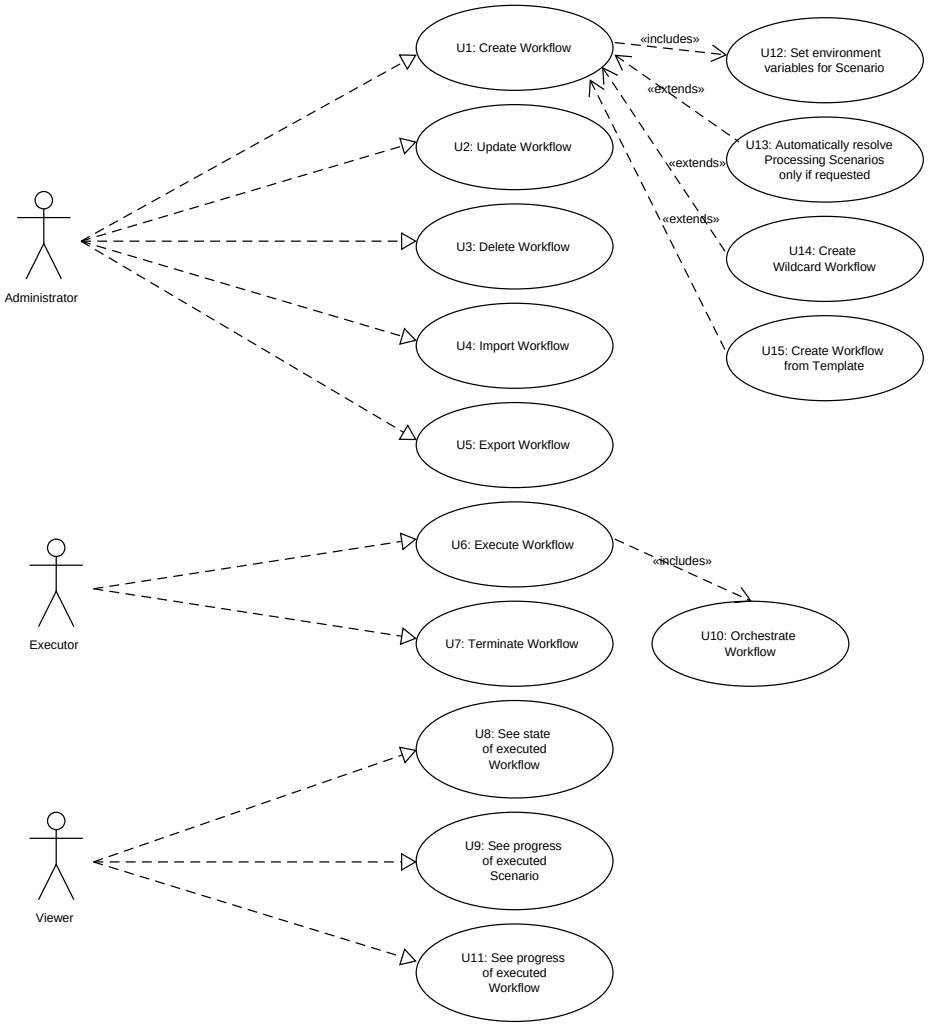

Figure 3.5: Use Case Diagram

U8: See the state of executed Workflow

User must be able to distinguish from Workflow states easily. Workflow states can be seen in the state diagram 4.5.

U9: See the progress of executed Scenario

Today, all Scenarios do logging at some level. Some of them are genuinely verbose; they are tracking their progress and know how much work left. Some of them does not do that so well. For example, graph processing Scenarios do not know how much work left because they do not have knowledge about

nodes count and edges count. Making these Scenarios to count how much work they must do could be time-consuming. Scenario states can be seen in the state diagram 4.4.

U10: Orchestrate Workflow

Workflow execution runs in parallel using the most potential of machine resources. Workflow definition works as a set of Jobs which is organised to meaningful Workflow during run-time.

U11: See the progress of executed Workflow

User may list the Execution Plan and see its progress and steps real-time. They can identify the progress of Workflow by states of each Scenario, by how much time left calculated from previous executions, and by a number of completed tasks[7] and how many tasks left.

U12: Set environment variables for Scenario

User has an option to define Environment Variables for Scenario. Some of the Scenario needs to set Environment Variables before executing. These variables may hold Connection information or libraries referenced needed by Scenario, e.g. `LD_LIBRARY_PATH`.[8]

U13: Automatically resolve Processing Scenarios only if requested

There should be an option to create a Workflow without automatically added Processing Scenarios for specific cases for customers with specific needs.[9] For example, a user may want to run MANTA to the whole environment, which may take days to finish. He or she cannot block his or her systems during working days and weekend is not long enough to analyse all systems. Thus he or she may create a multiple smaller Workflows which will partially process some part of the environment overnight. Using these small Workflows, he or she may analyse the whole environment without preventing others from work.

3.2.2 Functional Requirements

"Functional requirements are product features or functions that developers must implement to enable users to accomplish their tasks. So, it is important to make them clear both for the development team and the stakeholders.

[7] a Scenario work unit; for some Scenarios, this information may be unavailable

[8] this change was requested in mid of March, and it caused a minor change in Workflow Definition and redesigning the Scenario execution

[9] this change was requested on 27. 4. 2020 and the already existing design did not need to be changed so much

Generally, functional requirements describe system behaviour under specific conditions." [14]

F1 (U1, U2, U3): Create, Update and Delete a Workflow

A user enters the application; he or she creates a new Workflow where he or she fills name, description and chooses Scenarios for Workflow definition, which defines what will be executed.

The application shows only available Scenarios that have a Connection configured[10]. These possible Scenarios are divided into groups by Phases: Extraction, Analyse, Export. Each Phase has sub-steps which are specific for every Technology and Connection ID[11]. User should be able to filter Scenarios using filters Phase, Technology, Connection. Step order in Workflow definition does not matter. The Scenarios' execution order will be determined on runtime and will use the most potential of the user's machine resources – the parallelism.

User should be able to choose some options for the Workflow as Minor or Major Revision type representing the Full run and the Incremental run or override the max thread count for a Job.

When the user finishes the Workflow, he or she saves it, and he or she should be able to find it in his Available Workflows list. The list is persistent for system shutdown. He can edit existing Workflow in the same way as he is creating a new one only with the difference that editing will have prefilled all data.

The user can also remove the existing Workflow. This operation will request confirmation, and it will be irreversible.

These operations are performable via API. Workflow creation using API may work the same as Workflow import. The main benefit of Workflow creation is the guiding graphic user interface.

F2 (U12, U13, U14, U15): Create a special Workflows

The application should allow creating a special Workflows called wildcards. These wildcards will always execute all available Scenarios even if new configuration is created when the Workflow already exists. For example, users can define 'execute all for Connection `DB2_PROD`'.

Users can create Workflows by choosing from some predefined Workflow Templates. Default predefined Workflows will match already existing Workflow Scripts, so users will be familiar with these definitions as they already exist in MANTA.

[10] configured in MANTA Configurator, and information may be obtained from Configurator's API or file system

[11] unique Connection identifier is a pair of Technology name and Connection ID

User must be able to override or define environment variables for execution. For example, to make Informatica PowerCenter able to run, a client must define `INFA_HOME`, `PATH` and `LD_LIBRARY_PATH`. MANTA itself provides an option to define Java environment variables as `MEMORY_OPTS`, or `JAVA_HOME` and `JRE_HOME`; or MANTA environment variables as `MANTA_DIR_INPUT`, `MANTA_DIR_OUTPUT` and `MANTA_DIR_LOG`. This must be configurable in each Workflow for each pair (Connection, Scenario), because the user may have multiple versions installed for the same technology.

There will be two modes for creating Workflow: guided mode and advanced mode. In the guided mode, users do not need to specify MANTA related Processing Scenarios[12]but only Technology Scenarios. In this mode, these Scenarios will be automatically added. On the other hand, in the advance mode user may create very specific Workflows without these processing Scenarios to achieve special requirements for MANTA. For example, if a client has a lot of large database system, they may be creating data lineage over five nights, so they will need to run five separate workflows every night. And after that run processing Workflow containing only processing Scenarios.

F3 (U4, U5): Export and Import Workflow

User should be able to export any existing Workflow to JSON using the graphic user interface and importing an existing one from JSON. Importing JSON during Workflow creation will prefill fields and Workflow definition but will not save it. Importing elsewhere will automatically save the Workflow, on name duplication will be added timestamp of the import time.

Exporting Workflow in mid-process of creation Workflow asks a user to store a generated file on their file system. So, this option will create a JSON file from the partial Workflow Definition. The user can save this file into a local disk and upload it later moreover continue where he stopped his work. The Workflow does not have to be valid all the time during creation even though it may be exported at any time. That is not a problem for Workflow editor because during importing mandatory fields it may be filled by empty value and saving will be unavailable until the user fills all requested fields. This Workflow will be impossible to import elsewhere because of its missing mandatory fields - it is not a valid and complete Workflow.

When an error occurs during Workflow creating or updating[13], the application shows a modal window with the error message, and a user should be able to export the work-in-progress Workflow to save it locally and reupload it when the problem disappears. This will prevent from losing user's work. This export must be done on the client-side.

[12]e.g. revision prepare and commit
[13]e.g. system crash

Export and import are available trough API and GUI. Export via API is available only for saved Workflows, and import using API creates a new Workflow, so the Workflow must be valid and complete.

F4 (U6, U7): Execute and Terminate Workflow

User can execute Workflow which will be put it into the Waiting Queue of running Workflows. User cannot start Workflow, which is currently being executed or it waits for the execution.

He or she can also easily remove it from that queue. It is a simple task to removing Workflow which is not running, but Workflows being executed must be terminated. Termination will process-kill the Scenario which is being executed, and no other will be started, and the `terminated` status for the Workflow will be set. Process-kill is not a solution for every Scenario; some of them may lock remote resource; thus, a special Scenario must be run to terminate current Scenario. A good example is the Collibra Export Scenario because `CollibraTerminateExport` Scenario must be executed to stop it. After that, either the Rollback or the Commit procedure will be performed based on user choice.

API will be provided to control Workflows with the same behaviour as above.

F5 (U10): Workflow Orchestration

Orchestration manager can execute only one Workflow at a time from the waiting queue. Orchestration manager optimises inner Workflow jobs (Scenarios) and finds the best Workflow Execution Plan to execute Scenarios in parallel as much as possible, considering available resources and requested threads. The application will have defined max thread count for the Workflow and max threads number for each particular Job (Scenario).

Creating Workflow Execution Plan must consider available resources. These resources will be manually definable, and it will use default values if not. Resources will be implemented in so-called weight system. Every resource will have their max capacity, and every Job will define its weight for this resource. During execution, this capacity cannot be exceeded to prevent its overloading. The weight number for every Job will use a default value from the MANTA team by default, and it will be overridable by users.

List of resources with its weight examples, the resources are final, but its weights are subject to change, and it will be regularly updated:

- MANTA Server Usage: Only # jobs can use MANTA Server - Job Weights (easy to implement)
  - Weight 1, light job, another job can use the MANTA Server simultaneously

  - Weight 3, heavy Job, no other job can use the MANTA Server simultaneously
  - Max capacity 3 (configurable)

- Local disk usage: The ratio between Reading and Writing operations (For balancing this ratio precisely a further analysis is required)
  - Weight 1, the Job mostly reads (for example, 5:1)
  - Weight 3, the Job mostly writes (for example, 1:5)
  - Weight 4, the Job reads and writes similarly (for example, 3:2)
  - Max capacity 5 (configurable)

Usage of remote nodes running the database systems as a resource does not have to be considered because load for them is minimal in contrast for what they are built for.

The parallelism between Workflows will not be part of MVP, but the application must be designed for its future implementation. MVP will allow executing only one Workflow at once, but in future, it should allow executing more simultaneous Workflows, and its orchestration and synchronisation. This feature will make able to start two independent Workflows simultaneously and merge their results into one data lineage. This is a very complex problem because of CLI architecture. Parallelism will be implemented inside of a Workflow, so for example, all extraction in one Workflow will run in parallel. This will replace parallelism of MasterScenario. Parallelism inside of Scenario will remain unchanged.

F5 (U8, U11): See state and progress of executed Workflow

User will be able to see a state of the Workflow. States are available only for executed Workflows or Workflows waiting for execution in the queue. States in detail may be found in 4.5. Users can see the state of Workflow as a whole or they can peek into it and see states of every Job inside of it. These Job[14] states are defining the Workflow progress. For example, if users see Workflow in the state 'Executing' and they look on its details, they can learn that five from thirty-three Jobs are completed, two Jobs are being executed right now, and twenty-six Workflows in the state pending are waiting to be executed.

If a user wants to look at details about a Job state, he or she can see how long the Job ran or how long it will approximately run or what is its error message or why it was skipped.

[14] some Jobs may represent particular Scenario, but not every Job does, some of the Jobs may have a different meaning; for example, parallel Job which executes multiple Jobs in parallel

F6 (U9): See the progress of executed Scenario

As users can easily see Workflow progress, they may go even deeper to see the progress of particular Scenario, and they find a link to corresponding logs in the Log Viewer for already executed Scenarios.

3.2.3 Non-functional Requirements

Next subsection is describing non-functional requirements for the application. MANTA had no requirements for Hardware, Memory, Standards[15], Monitoring and Transactions requirements.

T1: Performance

Application has to be real-time and immediately responding, on click confirmation must be under 1 second. Any more prolonged operation must show a spinner. Less than ten users will use the application at a time.

T2: Documentation

The application will be distributed with its API documentation in Swagger and User documentation as other MANTA applications. The installation manual is not required because the application will be installed with other MANTA Admin Console applications.

T3: Data Model

"JSON should be used wherever it is possible." [15] Use JSON format everywhere user can access any data. Namely, users' configuration for Scenarios, the Workflow Definition, and export and import will be in JSON format. The application configuration may be in property files[16].

T4: Data Migration

All Workflows, user have already created, must survive every update. Workflow can be updated, but it must persist in the application, and the user must be able to find it after the update.

Already existing Workflows in Scenario Scripts must be re-implemented in a new workflow format and used as Template Workflows. Users could customise these Workflow Scripts.

[15] application must meet general MANTA standards and standards of used frameworks
[16] Spring best practices

T5: Compatibility

Backward compatibility with Workflow Scripts must be achieved, ~~and Workflow Scripts should use the new Orchestration application~~[17]. The application must be aware that the user could modify their Workflows, User must not lose their modifications, a one-time upgrade may be needed.

~~Remove Master Scenarios (JARS orchestrating Scenarios parallelization).~~[18] The Workflow will substitute Master Scenario, and it will handle its parallelism.

T6: Configurability

Manta developers can specify a default max threads count for Scenario. This is already implemented in MANTA Configurator, and this information should be considered. User can change Scenario's max threads in MANTA Configurator. In application, users can specify max thread for Workflow to control how many Jobs can run simultaneously; the default value will be used otherwise.

T7: Information security

The used database must be embedded and accessible only from the application. It will be secured for unauthorised access by username and password, and this password will be safely stored in code.

An unauthorised user can reach only the login screen. Everything else requires a login.

T8: Audit logs

The application will use the new logging approach defined by MANTA Log Viewer.

T9: Prescription of used technologies, framework libraries

The application should be built using frameworks commonly used in Manta, namely: Spring, MyBatis, and React JS for Presentation Layer. If any new framework or library is needed, it must be approved by MANTA Librarian first.

T10: Reliability

Any error in the application must be logged, transaction or request may be cancelled, but the application should not be terminated.

[17] The requirement that Workflow Scripts should use new Orchestration application was rejected later by the Ordering Party. Current Workflow Scripts must not change because MANTA wants to keep an option to use MANTA product without this application.

[18] Because workflow scripts must be kept as is; Master Scenarios cannot be removed.

If a problem with CLI execution occurs, show it in Workflow Execution Hierarchy with the error message and hyperlink to the MANTA Log Viewer.

T11: Localisation

The application will be English only. Keep all texts for GUI in external localisation files.

T12: Architecture

Orchestration and Monitoring must be a Maven module, which will be integrated altogether with other modules to one application named Admin UI. The Admin UI is a web application distributed altogether with Apache Tomcat as its application server. This module must manage all of the backend's logic. It must also provide a REST API and asynchronous user interface implemented in React JS. This module may use logic from other MANTA modules.

T13: Integration to other systems and interface specification

The application will provide a secured API for the outer world, which can be integrated into Scheduling application commonly used by a customer.

The application will show hyperlinks to MANTA Log Viewer, and it will subscribe on the MANTA Log Viewer (or its part) to get Scenario state.

The application will use Scenarios in the MANTA CLI.

T14: System installation and update

The application will be distributed with other MANTA Administration applications via MANTA Installer.

MANTA Installer must do updating, so there is no option to use MANTA Updater for merging because Updater and this application will be bundled as one deployable artefact. After every update, the application must be able to migrate its data to the newest version without user intervention.

T15: Testing

The application will have integration tests with MANTA CLI and MANTA Log Viewer (or its part), and it will have JUnit tests.

T16: Administration

There will be no GUI or API administration. The administration will be done using configuration files.

T17: GUI

The GUI must match the graphic design of other MANTA Administration applications which will be done by MANTA graphic designer. Only wireframes are needed.

3.3 Requirements Analysis

This chapter analyses requirements and things related to them, such as business processes related to requirements, their current state - what is to be improved. This section deals only with the most critical aspects of the requirements.

3.3.1 Current Parallelism in Scenarios

The current architecture of Scenarios makes parallelism possible to use. However, it is not the best what could be achieved because of the unfortunate design decisions. It was an easy, quick and smart solution which solved the problem at that moment. It should have been only temporary because it had its drawback; it was not scalable. And because MANTA grew a lot in recent, this became a significant problem. [16]

There are currently two levels of parallelism implemented from three possible. The zero level of parallelism is in the Scenario itself. The Scenario creates necessary threads for its run, and its count may be configurable from the outside. These threads may, for example, process nodes or obtain metadata form resource or parse user scripts. That is already optimized for the best performance by MANTA developers, and there is no benefit from changing this behaviour. However, is configurable trough Connection configuration and it must be taken in mind.

The first level of parallelism creates multiple processes for processing multiple sources at once. That, unfortunately, works only for Connections in the same technology group, e.g. two independent MSSQL servers. That is because of the Scenarios architecture called Master Scenarios. This parallelism is handled by so-called `ParallelScenario` and executing a Master Scenario (e.g. `HiveExtractionMasterScenario`[19]) does not execute Extraction for Hive as someone could think. It executes `ParallelScenario` instead, which executes multiple `HiveExtractionScenarios` for every existing Connection in the Hive technology group. And these Scenarios are executed in parallel. This rule does not apply for all Scenarios; some of them are not capable of this parallelism because the resources cannot handle this load.

So, there is a space for improvements in level one. For example, to make the execution of all available Connections in parallel possible. And that creates

[19] this is not a correct example, but it is used for illustrating the problem

new challenges. The major problem is that Scenarios must be synchronized at some point because Analysis may start only if all Extraction is done already.

Level two, already mentioned in previous chapters, makes able to execute two Workflows in parallel. Use case behind this is that a user can create a Workflow for Connection One (called `WorkflowA`) and second Workflow for Connection two (called `WorkflowB`). He or she may run `WorkflowA` on a daily basis and execute `WorkflowB` only on a weekly basis. Doing it like this may save some computing power by not executing `WorkflowB` every day because it is not necessary for some reason[20]. The problem is that one day their executions meet, and both must be started at once to make their results be written in one Revision.[21]

There is another option for parallelism[22] that is based on having multiple CLI as computing nodes on different machines. They could be controlled from a single point (a server), and that server could be this application. This feature was rejected because it is not needed.[23]

3.3.2 Creating Customised Workflow for MANTA CLI

Creating a custom Workflow may be difficult for not experienced user. They must understand how MANTA works at least at some basic level and even though creating it is not user friendly. It requires editing batch and shell scripts.

For better understanding, it would be shown on a practical example [17]:

1. At first, a copy of a script executing all Master Scenarios must be created, (for example, `_run_extract.sh`)

2. From the source script (of the copied script) must be removed the Scenario which would be run on an individual basis.

[20] for example, the source is not changed often

[21] This problem was consulted with the Ordering party, and they said it is not necessary to have it now. Solving level one parallelism is enough now. However, they requested to make the design open to this feature. This is hard to accomplish. Created *WorkflowPlan* (which is a graph) must be merged with *WorkflowPlan* of the second Workflow. Nevertheless, both *WorkflowPlan*s are optimized for being executed alone, and because of merging it must be created again. That is a problem because *WorkflowPlan* is already being executed for the first Workflow, and it cannot be destroyed and created again easily. So plans must be merged and recreated from a certain point, or their Phases may be executed in sequence, and only the plan for Analysis and Extraction would be created again.

[22] may be called level 4

[23] A note from the future: the final design of this thesis is not blocking this feature. There may be multiple instances of *ScenarioExecutor* all linked to different CLIs. *ScenarioExecutor* will need the ID of the remote node, and it will identify itself by it. A proxy class could resolve, which instance of *ScenarioExecutor* will get which Scenario to execute. To determining it, a parameter in Workflow or in *ScenarioMetadata* could be used, which would be compared with the remote node ID.

```
1 export SCENARIO_OPTS="-Dmaster.properties.filter=
    ↪ OracleConnection1,OracleConnection2"
2 ...
3 unset SCENARIO_OPTS
```

Listing 3.2: Setting *SCENARIO_OPTS* with filter for particular Connection

3. The user have to remove all other Scenarios except for the desired Scenario from the copied script.

4. Then he or she have to wrap the Scenario of his or her interest by code 3.2.

5. At last, he or she must include his or her new script in the parent script (`_run*.sh`) to be executed together with other scripts.

As can be seen, this is not an easy task for an inexperienced business user, and users are often guided by MANTA Help Desk to achieve this task. [18]

3.3.3 Scenarios Progress and State Knowledge

This chapter describes the actual state of Scenarios' capability to provide their progress. The chapter is divided by Phases, and Scenarios are grouped into logical groups because most of them share progress-knowledge capabilities with others.

Extraction

There are some Extracting Scenarios, extracting from Oracle, MSSQL, Hive and SSRS, which have the tracking of their work already implemented. They provide their progress into log files, and they know their total number of steps beforehand.

Some of the Scenarios like a Teradata, SSIS, PostgreSQL, ODI and Netezza are capable of tracking their progress, but this feature is not implemented. For Teradata, the number of steps may be the number of completed database extractions, the total amount of databases is known, but its granularity may not be sufficient. For SSIS, it may be the size of the packages list and the projects list. For ODI, it knows the number of items to be extracted, and after each extraction, the information may be logged. For Netezza, it is similar as for Teradata, the number of databases extracted may be gathered. PostgreSQL doesn't have progress tracking implemented but logging on databases count is possible, and also a number of schemas may be available after a quick scan for great detail.

For some of the Scenarios, it is hard to tell. For example, DB2 does not log any progress. The schema extraction seems to be time-consuming because

tables are extracted relatively fast in comparison to schemas. But the number of schemas which have to be extracted is unknown.

Analysis

It is complicated for the analysis. For Dictionary Dataflow, the number of Extracted objects may be used for progress tracking. But this information must be given from Extraction Scenarios to Dictionary Dataflow Scenario. Another option is to count nodes in the graph, but that could be a very time consuming regarding graph sizes used in MANTA.

In the DDL Dataflow Scenarios, there is a file iterator that iterates over all files, so obtaining a number of files is straightforward. Processing of these files is already logged.

A lot of Analysis Scenarios work similarly, so a similar approach described above can be used for most of them.

Repository Postprocessing Scenario processes a large graph multiple times, and there is no knowledge of the number of edges and nodes.

There are some other Scenarios which are very fast with short execution time and tracking their progress won't bring any use because the user won't be able to see them for a time long enough to be concerned.

Export

All exporting Scenarios are using a similar concept. Statistics for graph processing are known from the Analysis, and file sending can be tracked by a number of already sent lines.

3.4 Existing Solutions

Considering MANTA requirements for this application, it can be said that finding an already existing application matching their requirements may be a real challenge or almost impossible. MANTA needs software tailored for its ecosystem to be easy to use, user-friendly, matching their existing graphical design and not to tie dependence on another vendor.

In any case, the behaviour and functionality of existing solutions must be analysed. Indeed, it is desirable for an emerging program of this type to match the behaviour and functionality of already existing applications thus to meet and comply with already defined standards. Users familiar with these programs will quickly learn a new one and understand how the new program works. E.g. process states and its names, monitored statistics, what is viewable on first sight in the processes overview, etc.

3.4.1 Prometheus (monitoring)

"Prometheus is an open-source systems monitoring and alerting toolkit originally built at SoundCloud." [19]

Prometheus gathers time series via a pull strategy over HTTP. This solution is not suitable for short live processes because the process can stop existing before it is scratched by Prometheus. This system provides a generic solution to monitor Ephemeral Processes (for example, Batch script). It is a middle layer called Pushgateway. Processes can push their metrics to Pushgateway and Prometheus can scrape them later as it can be seen in figure 3.6. [3]

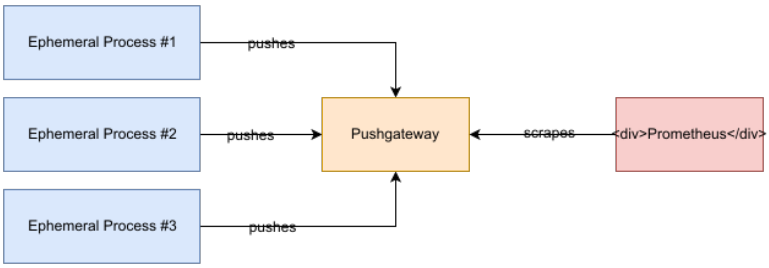

Figure 3.6: Prometheus Scrapes Cached Metrics from Pushgateway [3]

"Prometheus works well for recording any purely numeric time series. It fits both machine-centric monitoring as well as monitoring of highly dynamic service-oriented architectures. In a world of microservices, its support for multi-dimensional data collection and querying is a particular strength." [19] Preview from the Prometheus can be seen in figure 3.7.

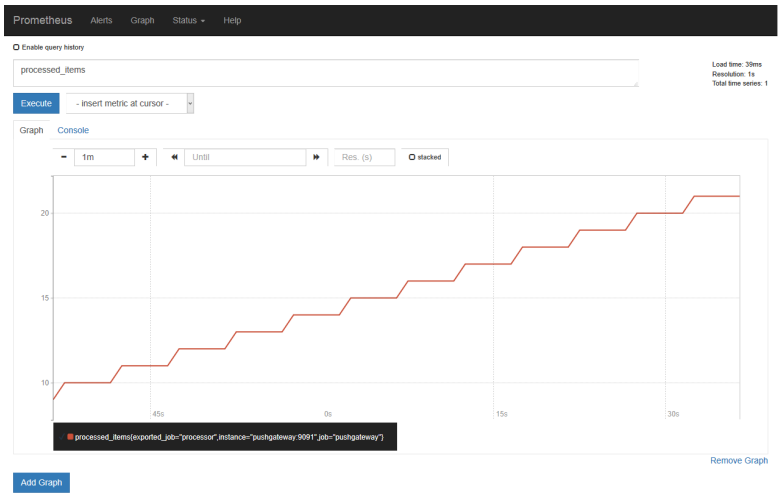

Figure 3.7: Example of a Metric Collected by Prometheus [3]

```
1 #!/bin/bash
2 set -e # if command fails, exit and do not ping
3 urlstring="http://pshmn.com/eaFnY"
4 curl -L "${urlstring}"
```

Listing 3.3: Snippet showing how to push notification to PushMon [20]

3.4.2 PushMon (monitoring)

PushMon is an online service with a free plan which provides monitoring for automation jobs. Usage is very simple and straightforward. User registered on PushMon, will get a generated unique URL. The user then adds short code (can be seen in snippet 3.3) to his job which calls the URL. And if the URL is not called on a scheduled time it generates alert via Email, SMS, Instant Messenger or Twitter. [20]

3.4.3 Activeeon (Orchestration and Monitoring)

Activeeon[24] is a paid on-premises or cloud web application providing a solution for workload automation from a French company Activeeon, SAS. In Activeeon application, a user can create his or her workloads as an orchestration of scripts execution or Java class execution; he or she can execute them, or plan the execution of these workloads, and monitor their state, status and history. The application provides analysis and statistics of previous executions, and it can alert the user if the execution failed (or succeeded), for example, by email. [21]

Activeeon Workload creation (Orchestration)

A workload, called Job, is a sequence of tasks and logical operations. Base tasks are script executions or Java class executions. This sequence is named Workflow. Activeeon looks like a professional tool for solving problems of this domain.

Workflows and tasks are written in XML. The unique identifier for a Task is a name of the Task defined by a user. There are three basic types of Task: Native, Script and Java. Native is a command executed by the default command line in the OS where Activeeon is installed (Shell, PowerShell, etc.). The Script is a hardcoded script in Activeeon GUI or an URL reference link to some script online. Activeeon supports many script languages as Bash, Shell, Powershell, Windows CMD, Javascript, Lua, R, Pearl, Python and more. The Java type, the most interesting type for this project, is Java class extending an abstract class (`Scheduler API`, `org.ow2.proactive. ... .JavaExecutable`) defined

[24] for analysing this tool an evaluation version was used, evaluation version can be obtained from `https://try.activeeon.com/`

by Activeeon. It can also be a JAR, but it must be added to the classpath. For all types, arguments and parameters can be defined. A result of these tasks is output script binding, and it can be used later in Workflow. There is also a special notification Task, which is a predefined Script Task, and it sends an email about progress or result. [22]

There is a significant configuration behind every Task, but it is not mandatory. To make Task executable, the user needs only to define the script. User can also define job priority, error handling, properties, attributes, define pre/-post/clean scripts, data management, multi-node management (nodes are execution resources) and more. However, a user can create predefined Task and use them in other Workflows. If the user specifies variables for a Task or a Workflow, he or she must define them on Workflow execution. These are parameters which can vary for each run.

Examples of error handling:

- Ignore and Continue,
- Suspend Dependencies,
- Pause Job Execution,
- Cancel Job,
- Number of Execution attempts

There are many options what a user can do with the workflow orchestration. For example, a Task can be forked to many tasks, and user can gather output from them; or he or she can wait for another task to finish its payload. Examples of logical operation used in Activeeon:

- If,
- Loop,
- Replicate,
- Task Dependencies,
- Submit Job and Wait,
- Wait for Any

The UI for workflow creation is user-friendly. Each component in the application has a help button which shortly describes what it does and how it can be used. User can go back or next step even hotkeys `Ctrl+Z` and `Ctrl+Shift+Z` work. User can drag and drop Task anywhere in work-screen and link it to other Tasks. After while the work-screen can be a little chaotic because of a lousy arrangement. The application has a button which sorts

and orders Tasks into a pretty and easily readable hierarchy. It may come in handy that every Job can be cloned to a new one. The Workflow is validated on the fly, and not disturbing popup notifies a user about the result in the top right corner. Errors are referring to a line number in the Workflow's XML.

Activeeon Execution and Monitoring

Activeeon helps user execute, plan and monitor Jobs. User can schedule workflow launch by creating a cron expression, or importing a `.ics` calendar and choosing an event from it, or creating an event in the internal calendar in the application.

The GUI is more in technical style; it uses tables with well arranged data. In Job centric view, there is one main table of scheduled and executed jobs called Execution list. A user can easily distinguish pending, currently executing and past Jobs, and he or she can use one-click filters to highlight these states, or he or she can use more complex filters to filter what he or she needs. User can switch to Task centric view, and see Tasks from all Jobs and their details for the period. A job can be in many states; all states are available in ProActive Workflow Guide.

Details of jobs are displayed in two ways. One approach is table view with detailed data as id, state, execution nodes, failures, duration and attempts for each Task. Other approach shows the Job as a Workflow (when Task centric view is active) and shows only the most crucial information as Task names and its results.

User can click on particular Task and see all details like id, result, duration, attempts, started at, finished at and description. User can go to the panel of Statistics or Usage and see some useful information from Scheduler about executions (business-centric details (as graphs) can be seen on Job Analytics Dashboard).

3.5 Summary

These and other non-mentioned applications, are trying to solve the challenges as broadly as possible so they can cover as many use-cases as they can to be competition capable. For example, Activeeon can create a run sequence of apps with a parameter (a Workflow); it can run these sequences in parallel; it can work with variables, and thus it can create more complex execution rules; it can monitor the running processes, and view statistics. Activeeon can do pretty much everything that is in requirements of this application and even a little bit more.

So why not use some of the existing solutions? Because what is their competition capability is a problem for MANTA. They creates unnecessary complexity and demands on users and their knowledge. MANTA needs software that shields users as much as possible from the complexity of the workings

and linkages of the individual parts of the MANTA ecosystem. And in creating the new Workflow, it will help the user as much as possible, automate everything that can be automated, and help the user achieve their goals with minimal knowledge of how MANTA processes work and without the need to know the MANTA technical aspects. Such tailored software is easily adaptable to MANTA requirements and does not create further reliance on third-party software, which may be a risk.

Compared to existing tools, the new one must:

- suit to MANTA ecosystem (functionally),
- fit to MANTA brand - graphical design and logo,
- be usable without special configuration,
- guide Workflow creation, simple even for not IT specialists,
- prevent and correct mistakes made by users during workflow creation,
- automatise orchestration, users do not need to deal with execution order,
- provide an overview of the Workflow execution state.

From a detailed analysis of requirements and a search of existing solutions (both in the previous parts of this chapter), there is a clear decision to design and implement new tool.

Chapter 4

Design

This chapter describes the design of the application. First, it looks at the news in business processes and how they differ from previous. Next sections, describes the architecture of the application, then the architecture of the Orchestration and Monitoring modules and its basic operations in sequence diagrams showing usage of chosen existing solutions. After that, there are explained states of Workflows and Scenarios and is introduced graphical user interface design, including wireframes. Last but not least, there is described database model, Scenario Metadata and application configuration.

4.1 New Business Processes

One of the most significant disadvantages in the current business process is the creation of a custom Workflow. The need to commenting out, editing and adding lines of codes into the Scenario Scripts is not very user friendly. Probably an experienced user can get to use to it, but overall it is not convenient.

The new business process for Workflow creation focuses on a user-friendly approach. The main goal is to provide users with an option to make or edit a Workflow in a more friendly way but the respect of backward compatibility. The new approach of maintaining Workflows will be set side by side with the original CLI way, and it will meet standards already set in MANTA - web application running on Tomcat together with Updater and Configurator.

From the users' perspective, they will be using already emerging application Admin UI where Process orchestration and Monitoring will be a new component. Users will be able to see their already configured Workflows[25] and create a new one there too. Creating a new Workflow for the first time may be confusing, so to help users to reach their goals, they may form a Workflow from predefined templates. These templates will work same as in CLI: run

[25]Only for Workflows created in Process Monitoring and Orchestration. Already existing Workflows in CLI will not be considered.

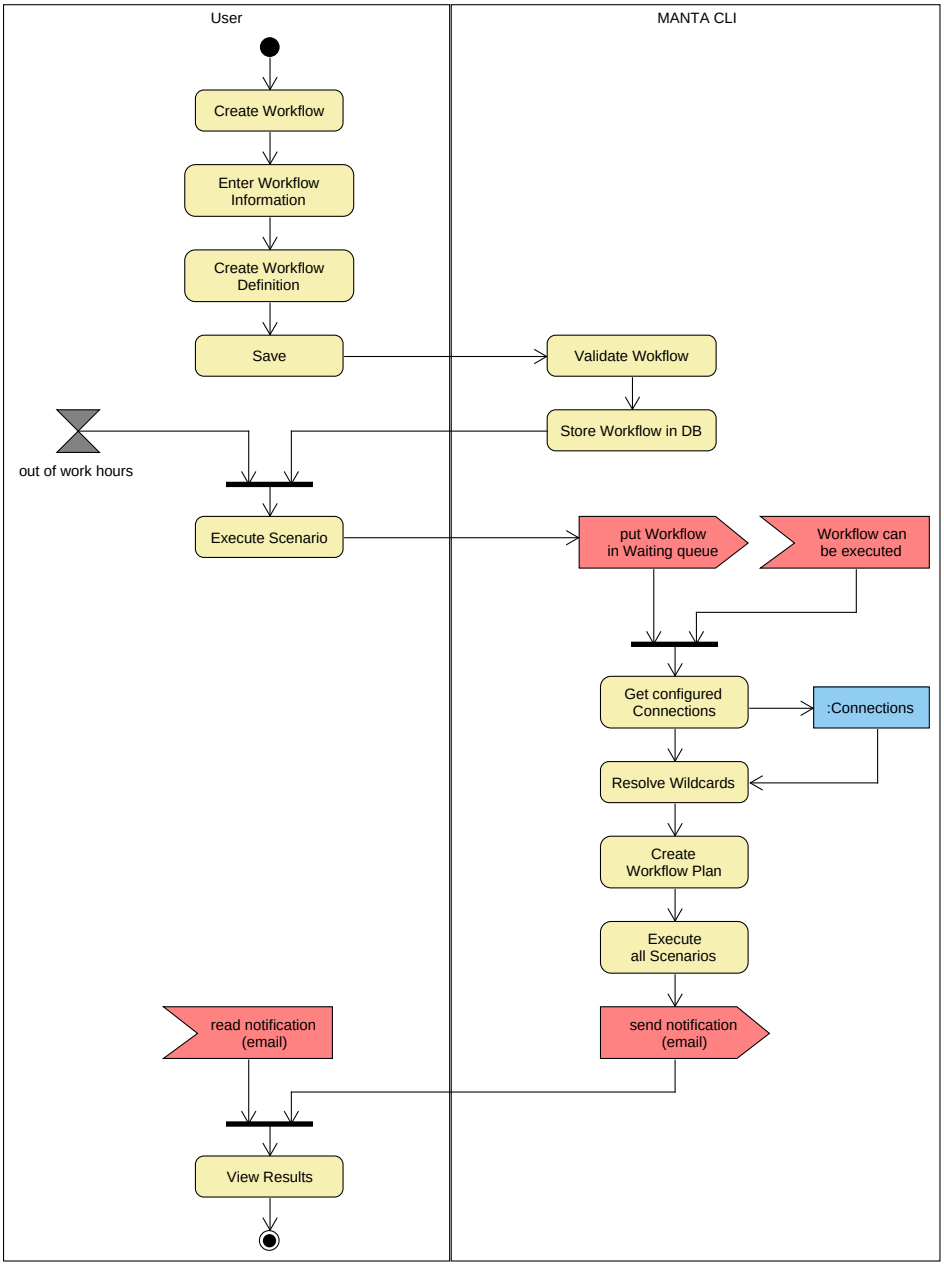

Figure 4.1: New Business Process Activity Diagram

'all Scenarios', 'run all analysis Scenarios' and others. When users choose to create a Workflow from a template, they may edit this newly created Work-

flow, which should be much easier[26] then starting with a blank page. Users may use Scenario wildcards, to execute a particular group of Scenarios, for example, all Scenarios for specific Technology or Connection.

Experienced users may choose to form Workflow in advance mode. It will unlock managing the processing Scenarios, e.g. New Revision, Commit or Postprocessing. In this mode, users are allowed to have full control over Workflow so they may prepare more complicated Workflows to satisfy their complex business goals. Otherwise, these Scenarios will be dynamically included on runtime as application decides.

Scenarios in Workflow will be logically grouped. At first, by their Phase[27], then by their technology and at last by particular Connection name. It will form a tree structure where leaves are Scenarios, and inner nodes are Phase $\rightarrow$ Technology $\rightarrow$ Connection in this order with zero[28] or multiple descendants. This approach may be seen as too much complicated, but all MANTA clients participate in a training program, where they learn how background processes work in MANTA CLI, and this approach should support and evolve this know-how. Moreover, as it was mentioned before, they may start using MANTA only via templates which are executable without modifications, only the configuration in Configurator is required.

In addition, users may create a description for Workflow for quickly reminding what a particular Workflow does; they may choose if a Workflow is a Major revision or update (a Minor revision) of the last revision which might or might not be available only in advance mode. Created Workflow will also have an icon, based on technologies used in Workflow, which may be changed by the user.

When the user executes a Workflow, the orchestration logic will arrange the execution order of all Scenarios, so users do not have to think about it during creation, altogether with adding processing Scenarios for Workflows created in basic mode. Execution order will always be (even in advance mode) resolved by the application, but users may skip some Scenarios. For like this fully expanded Workflow and with wildcards resolved, an execution plan will be created. This plan will consider the relationship between Scenarios and will try to found as many Scenarios that can run in parallel as possible to improve execution time, which is the second significant benefit of using Process Monitoring and Orchestration instead of original CLI approach.

Although Workflow optimization, it still may run for hours or days, and users must be able to see progress and state of the Workflow. Currently, users have only one option to monitor their Workflow. MANTA logs the progress of Scenarios into log files. Users can see how many steps were done, and for some of the Scenarios, they may also find information about total step count, but

[26] Because of the templates simplicity.
[27] Extraction, Analysis, Export
[28] Zero descendants will be resolved as a 'for all' wildcard

not all Scenarios are capable of giving this information. Logfiles display very detailed information about each Scenario, and it is irreplaceable. Monitoring of this application will focus on an overview of all Scenarios in Workflows. The application will provide essential information like an execution start time, finished time, how many Scenarios are already finished, how many steps were done in a Scenario and how many left to be done. If users want to see details, they will have a hyperlink for the Scenario which will redirect to newly emerging Log Viewer.

4.2 Application Architecture

Application is logically divided into two modules: Orchestration module and Monitoring module. These two modules must be integrable into Admin GUI where they will run alongside with Updater, Configurator and Log Viewer. Orchestration module is responsible for storing new Workflows, preparing an execution plan and performing the execution. It is also capable of handling errors and their fallbacks. Monitoring module gathers states from Scenario processes and links them with the execution.

MANTA has a technical requirement for using technologies. For example, a decision made in September 2019 was to use a new javascript framework for the emerging Admin UI in MANTA. It was mainly between the React JS and AngularJS. In the end, MANTA senior engineers selected React JS as a javascript framework. The application's backend must be written in Spring using Java configuration; frontend must be implemented in React JS, and H2 database must be used as persistent storage. MANTA raised these limitations because they already use these technologies, and they do not want to introduce new technologies if unnecessary.

Application is architecturally designed as two-tier application known as thin-client. The graphical interface is implemented as a web application in javascript using React JS framework, and it is connected to the backend via REST API.

The traditional multi-layer architecture was chosen for the application's backend architecture(see figure 4.2 for details). Unlike traditional three-layer architecture, the used architecture is extended for the fourth infrastructure layer.

React application communicates with the backend via public and private REST API. API is controlled by the Presentation layer, which transforms data transfer objects to domain model objects and calls corresponding services on the Service layer.

The second layer, the service layer, provides the logic for internal processes of the application. For example, in the orchestration section is it: process start-up, raw-format workflow processing, workflow storage, and configuring the application will be addressed. On the other hand, in the monitoring sec-

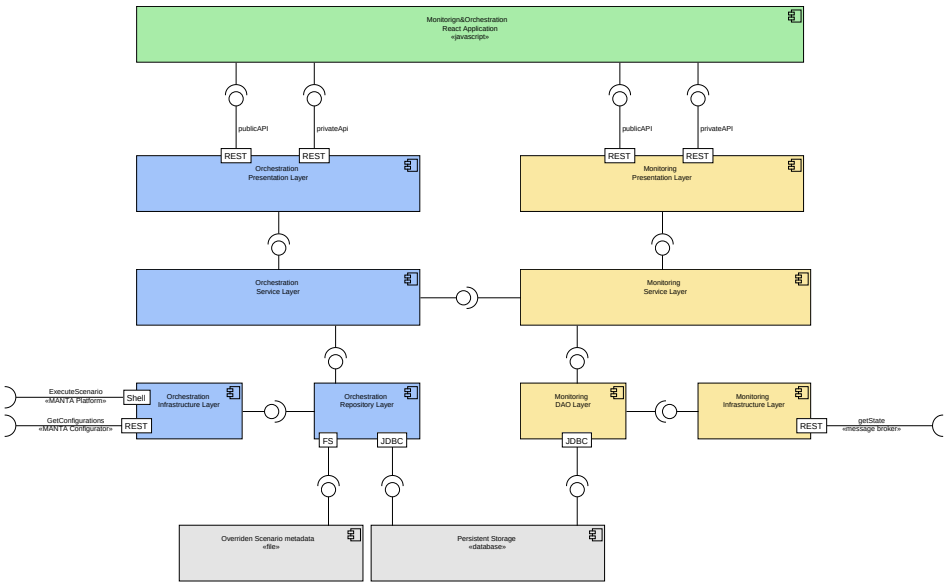

Figure 4.2: Application Architecture Diagram

tion, the collection of data and progress of running Workflow will be gathered and processed using the Message Broker created for the purpose of MANTA Log Viewer. Monitoring Service Layer will also link states gathered from Message Broker to the Workflow Execution Plan.

There is a link between the Orchestration Service Layer and Monitoring Service Layer. Monitoring service Layer needs information about Execution Plan of currently executing Workflow or past Workflow executions.

The primary function of the repository layer is to provide access to the database and filesystem. It also provides functions for communication with other application and modules such as MANTA CLI, Log Viewer and Configurator, but its implementation is separated from the repository layer into the infrastructure layer.

The infrastructure layer is an adapter to the MANTA ecosystem. Orchestration infrastructure layer provides an interface to run scripts through MANTA Platform, and it is responsible for collecting existing configurations from Configurator. Monitoring infrastructure layer gathers states from Scenarios using MANTA Log Viewer.

4.2.1 Deployment Diagram

Monitoring and Orchestration application benefits from the restriction that Configurator must run on the same device as MANTA CLI because it needs access to the file system to create configuration files. This restriction allows

direct executing of Scenarios using command line without implementing a service providing a way to execute Scenarios remotely.

In figure 4.3 are highlighted newly added components and artefacts by orange colour. There can be seen new module Monitoring and Orchestration with API and links to other components and applications.

MANTA CLI does not get by without any change too. As it can be seen in figure 4.3 there are highlighted new links in CLI. CLI can execute any Scenario nowadays. However, Master Scenarios loaded the Connection configuration used by Scenario to connect to a remote resource, and Master Scenarios should be no longer in use when a Scenario is executed and orchestrated by this application. Master Scenarios are capable of filtering a particular existing Connection. Nevertheless, to use this feature, the Spring context must be changed before every execution to tell a Master Scenario which Connections it should filter, and it is just a workaround to avoid Master Scenario although it is used. After discussion with MANTA, they agreed to implement a change in MANTA Platform to make it capable of accepting Connection properties differently without Master Scenarios, and these highlighted links refer to this change in MANTA Platform.

4.2.2 State Diagrams

Orchestration and Monitoring tool distinguishes states only for two objects: Workflow and Scenario Execution. Workflow's state is resolved from states of each Scenario Execution. For example, failure of one Scenario Execution may be resolved as 'Failed' Workflow's state for mandatory Scenario or 'Success with failure' Workflow's state for optional Scenario.

Scenario Execution States

Scenario Execution changes between seven states. These states may be logically divided into three simple groups: not yet executed, executed, not executed because of an error. Error level or return code and result message will be injected into Scenario Execution object after execution. Details of each state may be found in the enumeration below and the figure 4.4.

Pending Scenarios that are waiting for their turn to be executed are set to the Pending state.

Running Scenarios that are currently being executed are set to the Running state.

Succeeded Scenarios that succeeded in their execution are set to Succeeded state.

Failed Scenarios that failed on their execution are set to the Failed state.

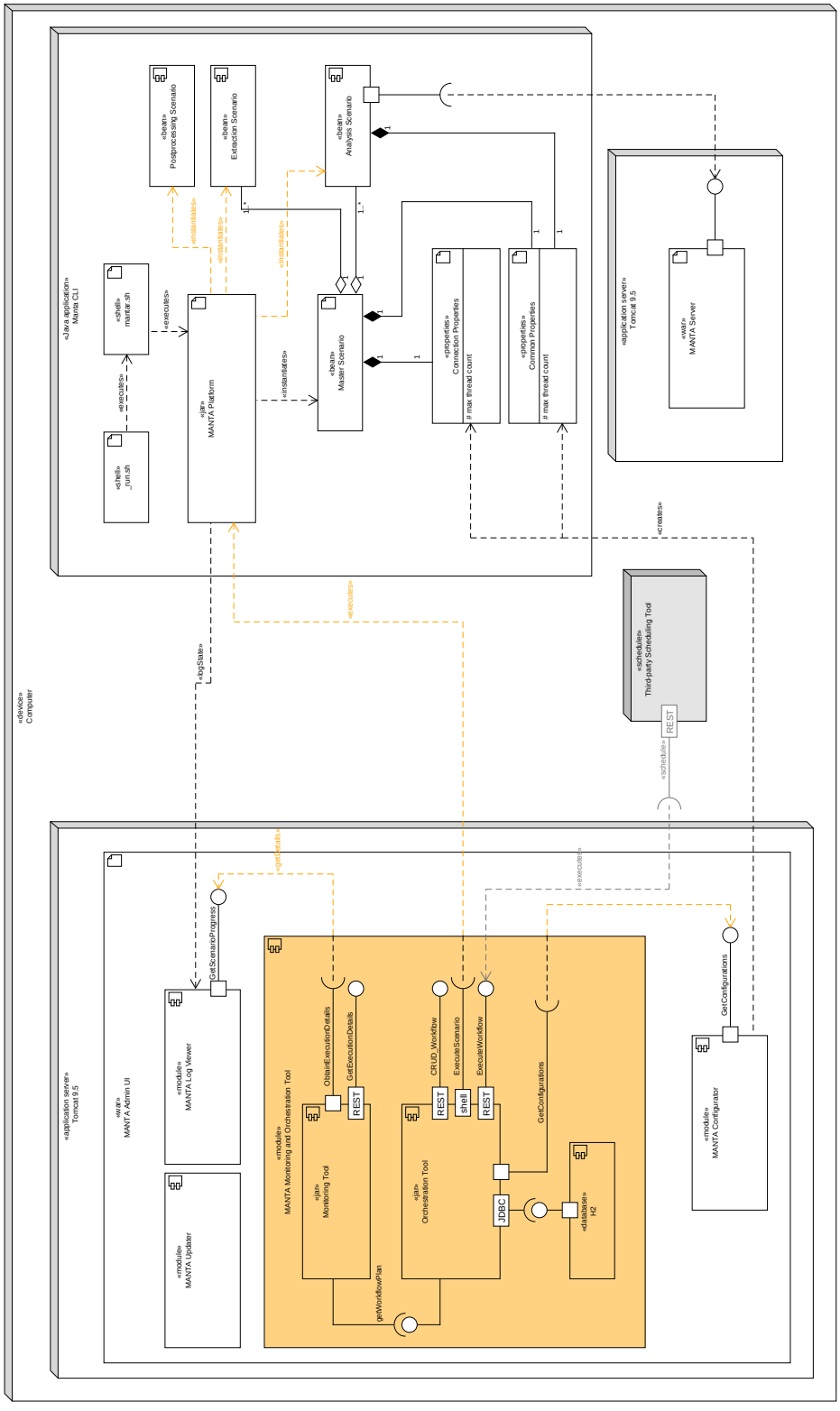

Figure 4.3: Deployment Diagram

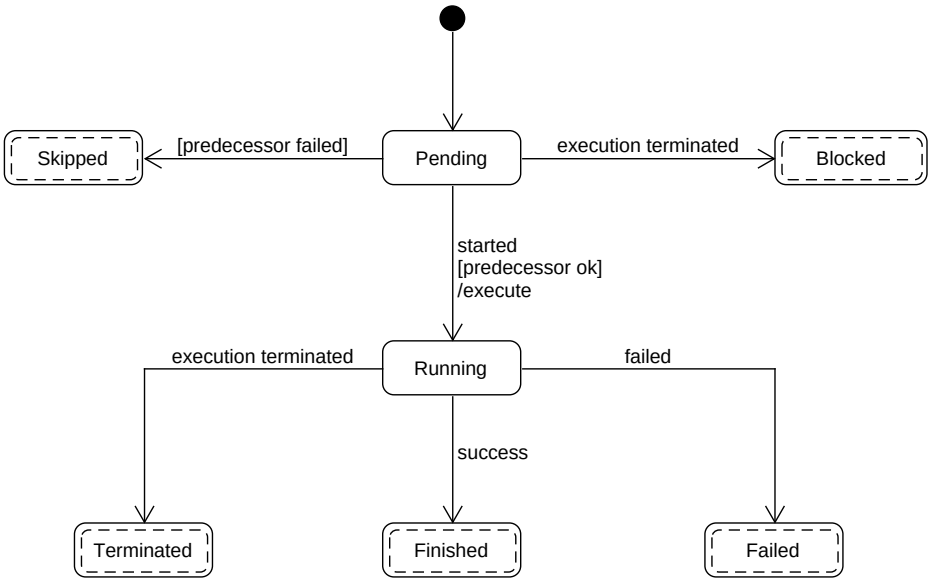

Figure 4.4: Scenario Execution State Diagram

Terminated Scenarios that have just been run are set to the Terminated state when the termination signal is sent to them.

Skipped All not executed Scenarios which cannot be executed because of their mandatory predecessor's failure are set to the Skipped state.

Blocked All not executed Scenarios which cannot be executed because of total Workflow failure or termination are set to the Blocked state.

Workflow Execution States

Workflow execution state is resolved from all Scenario Executions based on a type of failure: total failure or group failure[29]. If a failed Scenario does not trigger total failure procedure, it is resolved as Finished with Error state. If it does trigger it, then the whole Workflow is stopped and the Failed and the Blocked Scenarios are resolved as the Failed state. When the Workflow is stopped, the recovery procedure is started, and after that, the Failed Committed or Failed Rollbacked state is set. Rollback or commit procedure is chosen base on the configuration. The Terminated state is resolved similarly. In the mapping diagram 4.6 may be found all transforming rules.

[29]Scenarios are divided into Execution group by its Connection. If two Scenarios should be executed against the same Connection, they are also in the same group.

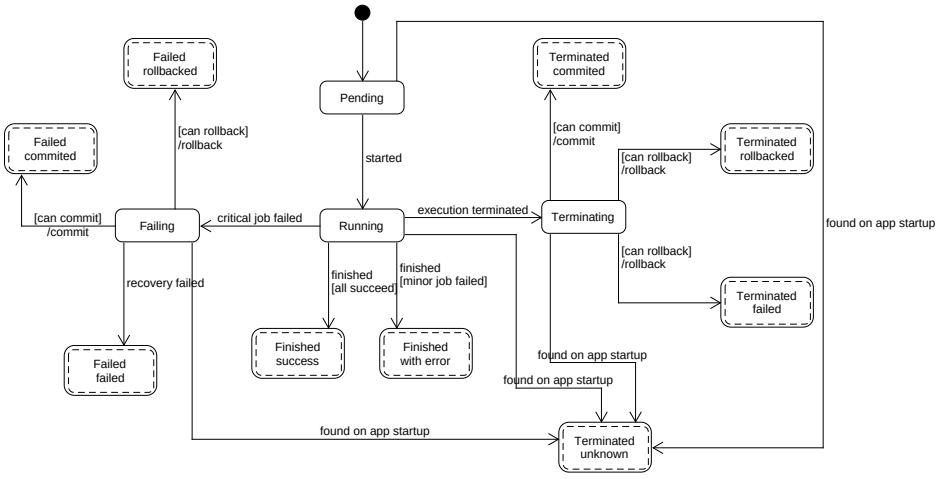

Figure 4.5: Workflow Execution State Diagram

Pending Workflow is waiting in a waiting queue for its turn to be executed. Probably another Workflow is being executed now.

Running Workflow is currently being executed. One of its Scenarios is currently running.

Finished Success Workflow finished successfully; all of its Scenarios finished with zero error level.

Finished with Error Workflow finished, but one of its Scenarios did not finish with zero error level. That Scenario was not mandatory to finish Workflow.

Failing One of the Scenarios finished with non-zero error level. That Scenario was mandatory to finish the Workflow and triggered the failure procedure.

Failed Commited One of the Scenarios finished with non-zero error level. That Scenario was mandatory to finish the Workflow and triggered the failure procedure. After the failure procedure, the commit recovery procedure started, and both finished successfully.

Failed Rollbacked One of the Scenarios finished with non-zero error level. That Scenario was mandatory to finish the Workflow and triggered the failure procedure. After the failure procedure, the rollback recovery procedure started, and both finished successfully.

Failed Failed One of its Scenarios finished with non-zero error level. That Scenario was mandatory to finish the Workflow and triggered the failure

procedure. After the failure procedure, the commit recovery procedure started, and one of them did not finish successfully.

Terminating The terminating signal was sent to the Workflow, and the terminating procedure was initiated.

Terminated Commited The terminating signal was sent to the Workflow, and the terminating procedure was initiated. After the terminating procedure, the commit recovery procedure started, and both finished successfully.

Terminated Rollbacked The terminating signal was sent to the Workflow, and the terminating procedure was initiated. After the terminating procedure, the rollback recovery procedure started, and both finished successfully.

Terminated Failed The terminating signal was sent to the Workflow, and the terminating procedure was initiated. After the terminating procedure, the rollback recovery procedure started, and one of them did not finish successfully.

Terminated Unknown Any Workflow found on application startup in not finite state will be transformed to Terminated Unknown as a result of application failure.

Because multiple Scenarios may be resolved as different Workflow execution states, they are reduced into one state using their preconfigured severity level. Their severity may be seen in table 4.1.

Name	Severity
Finished Success	0
Finished with Error	10
Pending	20
Running	30
Failing	35
Failed Rollbacked	40
Failed Commited	45
Failed Failed	50
Terminating	60
Terminated Commited	70
Terminated Rollbacked	80
Terminated Failed	90
Terminated Unknown	100

Table 4.1: Workflow State Severities

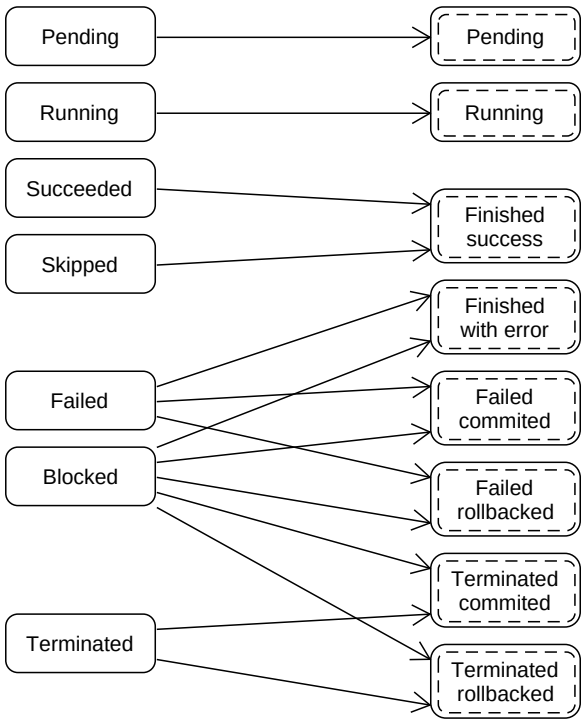

Figure 4.6: Scenario Execution State to Workflow State Mapping Diagram

4.3 User Interface Design

MANTA has already designed a layout (may be seen in the figure 4.8) for the user interface which is implemented for MANTA Configurator and MANTA Updater. For this project, new screens with elements matching the existing layout must be created. Admin UI is a web application implemented in React JS, and MANTA does not support mobile devices; thus, the created UI design is only for desktop.

Admin UI layout is divided into three sections: the top navigation menu, the left content bar and the main content. The layout of the top navigation menu is unchangeable; it is used to switch between Admin UI modules. Only the left content bar and the main content may be changed. The left content bar is used in Updater and Configurator as a menu for the modules, and the main content is the module's screen for showing information. This behaviour must be met to keep consistency between modules.

This chapter defines tasks that user should be able to perform using graphic UI; introduces screen layouts using wireframes, and shows the flow between screens. In this section, there is a subsection discussing the difference between

Scenario centric and Connection centric approach.

Wireframes are input for MANTA graphic designer who will prepare a pixel perfect graphic design from it.

4.3.1 Task Group

This section breaks down actions that are feasible within the application using GUI. These actions are grouped according to logical belonging.

Workflow creation or edditation

- Create a Workflow
- Update a Workflow
- Delete a Workflow
- Choose execution of a specific Scenario for Connection
- Choose execution of a wildcard[30]
- Export Workflow to JSON
- Import Workflow from JSON
- Set revision type for Workflow
- Create Workflow from a template

Workflow execution

- Execute Workflow
- Stop a pending Workflow in a waiting queue
- Terminate currently being executed Workflow

Monitoring of Workflow execution

- See currently running Workflows
- See the status of executed Workflows
- See Workflow's execution details
- See Scenario's execution details
- See the number of processed nodes for any Scenario

[30] for example, all Scenarios for Connection A

- See execution plan (execution hierarchy) of executed Workflow
- See execution history of given Workflow

4.3.2 Wireframes and Screen Transition

Screen transitions are shown in this subsection (see the figure 4.7) followed by a description of all screens with wireframe examples. The rest of the wireframes may be found in the attachment.

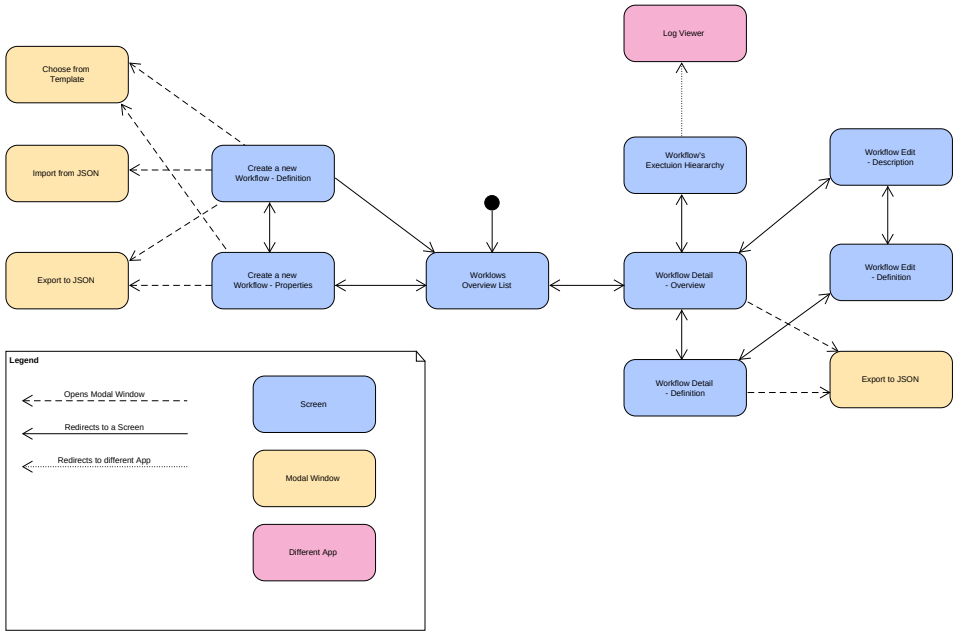

Figure 4.7: Screens Transitions

Workflows Overview List

This is the default and main screen (figure 4.8) of application showing all created Workflows ready to be executed, currently being executed Workflows and pending Workflows. From this screen, users may reach detail of existing Workflow or create a new one.

Create a new Workflow - Description

Users are redirected to this screen when they click on the button 'create new Workflow' (or edit Workflow, because these screens are the same). This (figure 4.9) is the main screen for Workflow creation containing a name, description and icon. User may also change the revision type if any Analysis Scenario is

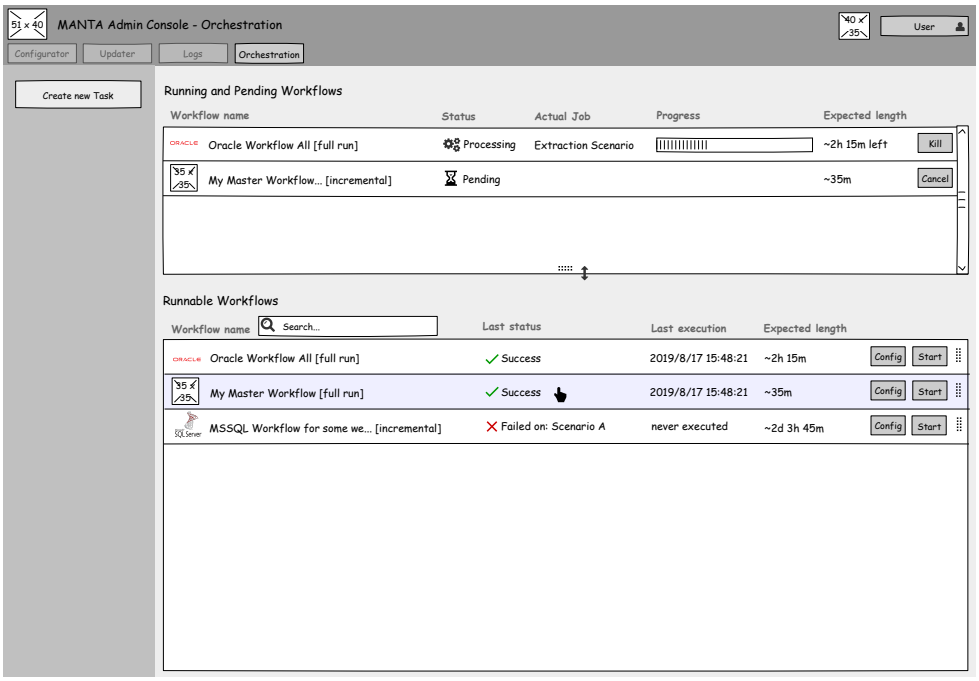

Figure 4.8: Workflows Overview List Screen

selected. Users may also enable an advance mode which enables some planning features for expert users.

On the left content panel, users may export or import a Workflow or open a modal window to create a Workflow from predefined templates which are automatically generated from existing Connections.

Create a new Workflow - Definition

From the screen 'Create a new Workflow - Description' user must change to this screen (figure 4.10) to define which Scenarios will be executed for the Workflow. These available Scenarios and Connections are determined from the existing configuration. Users drag & drop available Phases, Technologies, Connections or Scenarios from available list to execution list. The order in the list does not matter; it is a set which is ordered and planned on runtime.

User may click on every execution object to see details and basic configuration in the most right panel. If user moves a Scenario which is dependent on any processing Scenario and the Advance mode is not enabled that particular processing Scenario is automatically added to the Workflow definition.

The left content panel is unchanged from 'Create a new Workflow - Description'.

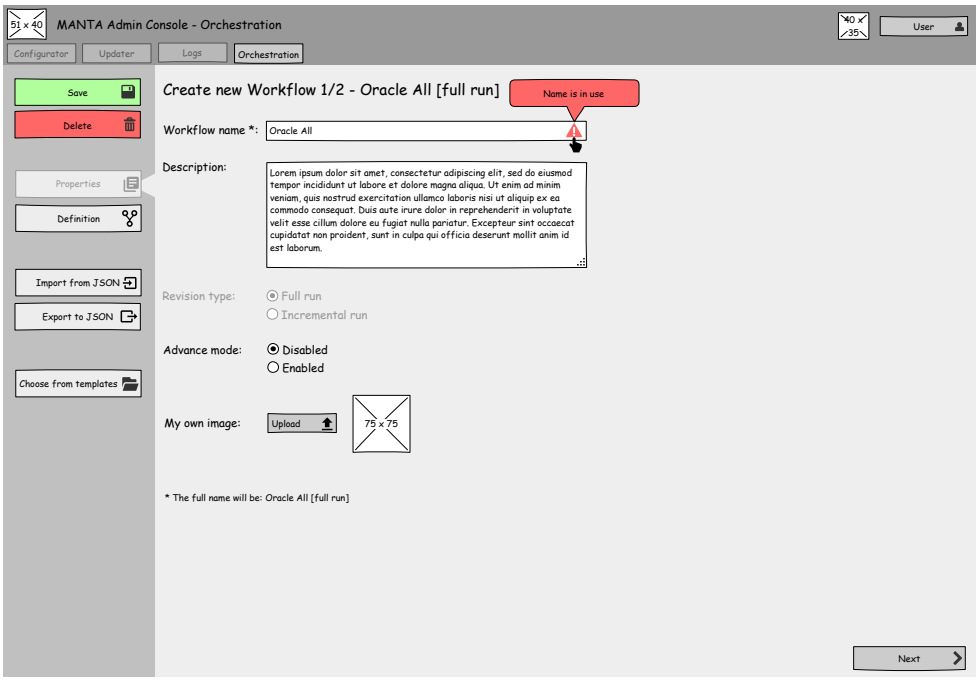

Figure 4.9: Create a new Workflow (description) Screen

Export to JSON

Export to JSON is a browser's window to save a file.

Import from JSON

Import from JSON is a browser's window to upload a file. If users decide to Import a Workflow, they are asked first if they want to replace current Workflow and that they may lose their work. The spinner is shown when the file is being uploaded and processed.

Choose from Template

Chose from Template screen (figure 4.11)is a simple list of predefined Workflows with a preview and description. On confirmation, user is asked if he or she is sure to override existing Workflow by the template. User may open the preview in a larger window (figure C.2).

Workflow Detail - Overview

Workflow Detail - Overview (figure 4.12) provides information about past execution, Workflow description, name and icon. From this screen, users

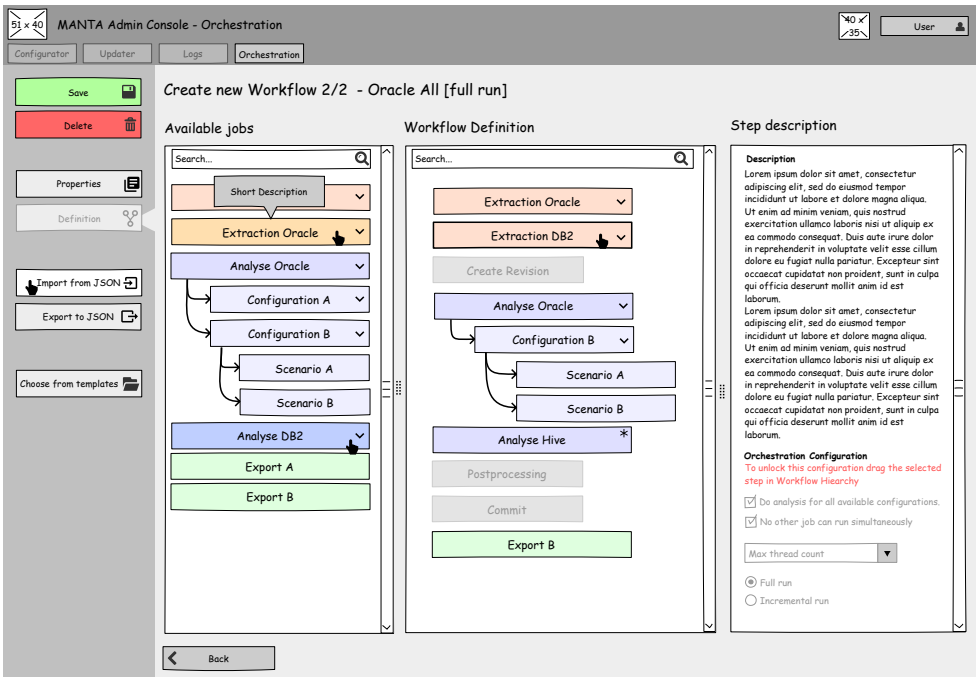

Figure 4.10: Create a new Workflow (definition) Screen

may access details about a particular execution. They can cancel or kill an unfinished Workflow from this screen. The left panel has options like editing the Workflow, exporting it to JSON, deleting it or executing it.

Workflow Detail - Definition

Users may click on a button on 'Workflow Detail - Overview' screen (figure C.1) to switch to Workflow's definition, which shows the collapsible tree structure of the Workflow definition exactly how the users created it. Users may read a description of each Scenario, see the available configuration for Scenarios and other executable objects, but the configuration is unchangeable. To change it the user must click on the edit button in the left panel. The left content panel is the same as on the screen 'Workflow Detail - Overview'.

Workflow's Execution Hierarchy

Workflow's Execution Hierarchy (figure 4.13) screen is execution detail screen displaying Workflow's result in a collapsible tree structure with fully expanded wildcard execution object, for example, if the user creates 'execute all Scenarios' for 'Configuration A' all the corresponding Scenarios will be displayed.

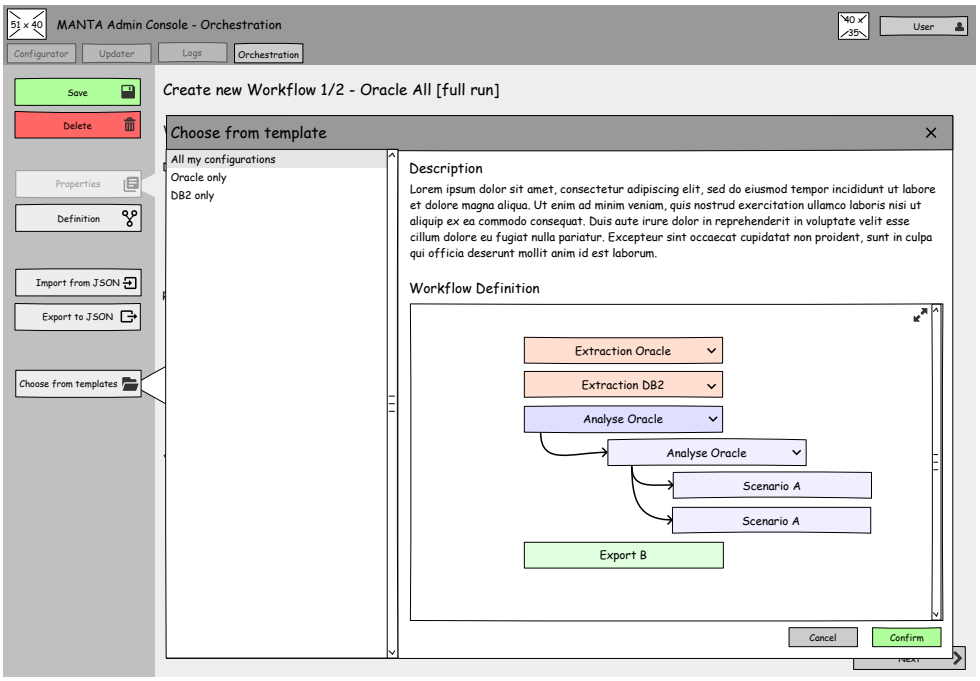

Figure 4.11: Choose from Template Modal Window

Every execution object has a status icon, so users may immediately see the result of each execution object. Every object also has a button which redirects into Log Viewer to see logs for the corresponding object.

When a user clicks on any execution object, the most right panel shows execution details, e.g. start time, finished time, processed nodes count. In the most right panel, users see the object's description and its configuration just as on previously described screens.

In the left content panel, there is a Run again, Stop button and a button to download all logs for the Workflow, which may come handy when a user reports an incident for MANTA Helpdesk.

Workflow Edit - Description and Definition

Screens 'Workflow Edit - Description and Definition' are the same as 'Create a new Workflow - Description and Definition'. Only the title is changed.

Log Viewer

Log Viewer screen is from module MANTA Log Viewer containing relevant logs for given Workflow execution.

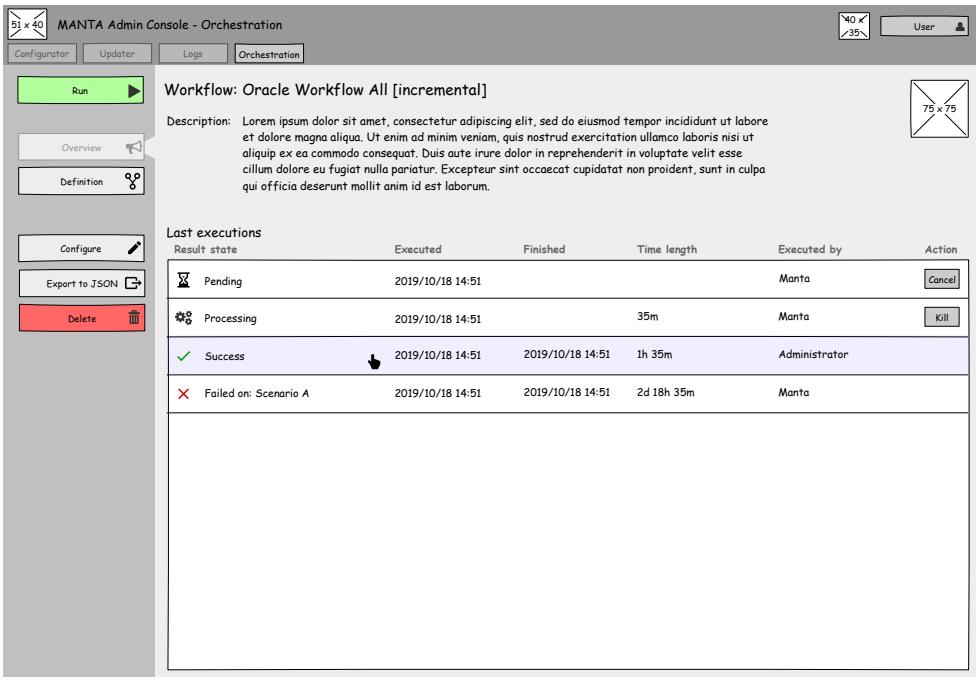

Figure 4.12: Workflow Detail Screen

4.3.3 Scenario Centric vs Connection Centric Approach

During designing GUI and preparing Workflow's JSON format, a question, how to order execution objects in the tree structure, came in mind. At first, two approaches seem to be usable: Scenario centric approach and Connection centric approach (see the figure 4.14 for example).

The Scenario centric approach would use new abstract Scenarios to which is Connection assigned. This assignment defines the specific Scenarios matching the original abstract Scenario. For example, abstract Scenario 'DictionaryMappingScenario' with Oracle Connection would become specific Scenario 'OracleDictionaryMappingScenario' at runtime.

The Connection centric approach does not use abstract Scenarios - users pick Connection first. When the Connection is known, specific Scenarios may be resolved based on the chosen Connection so that user may choose the specific Scenarios directly.

Two use-cases for better understanding:

Use-case for Scenario centric approach Users configure Connections in Configurator. After that, they enter the Orchestration and Monitoring application and creates a new Workflow. Then they define a Workflow by picking Scenarios from the available Scenarios list. After that, they assign one or more Connections to each Scenario. Users will have an

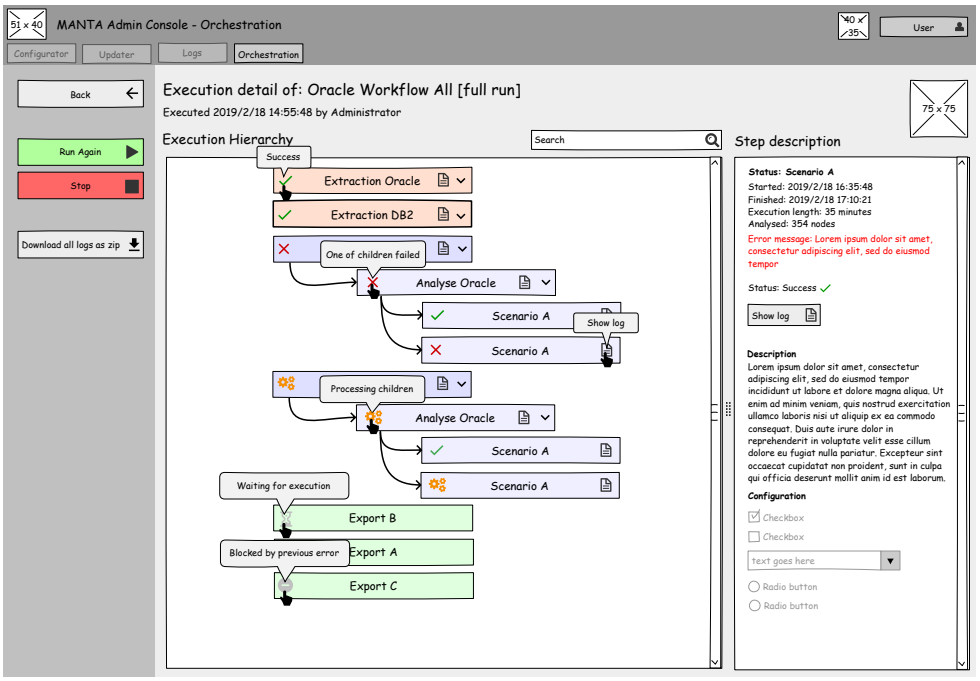

Figure 4.13: Execution Detail Screen

option to create Scenario groups; thus, they will be able to assign one Connection to multiple Scenarios.

Use-case for Connection centric approach Users configure Connections in Configurator. After that, they enter the Orchestration and Monitoring application and create a new Workflow. In Workflow, they create one or more Connections; then they define for each Connection a list of Scenarios to run for a given Connection.

From both use-cases, both approaches seem to be valid alternatives, and the Scenario centric approach seems to be more usable then Connection centric. However, that is not true when it comes to execution wildcards and execution of only one specific Scenario.

If users want to define a Workflow for executing all Scenarios for one Connection[31], which is very common use-case, they would need to specify all Scenarios first, and for all Scenarios, they must specify the Connection if Scenario centric approach would be used. On the other hand, if users want to define the same Workflow using Connection centric approach, they define the Connection with a wildcard mark which may be, for example, an empty list of specific Scenarios.

[31] a Connection wildcards

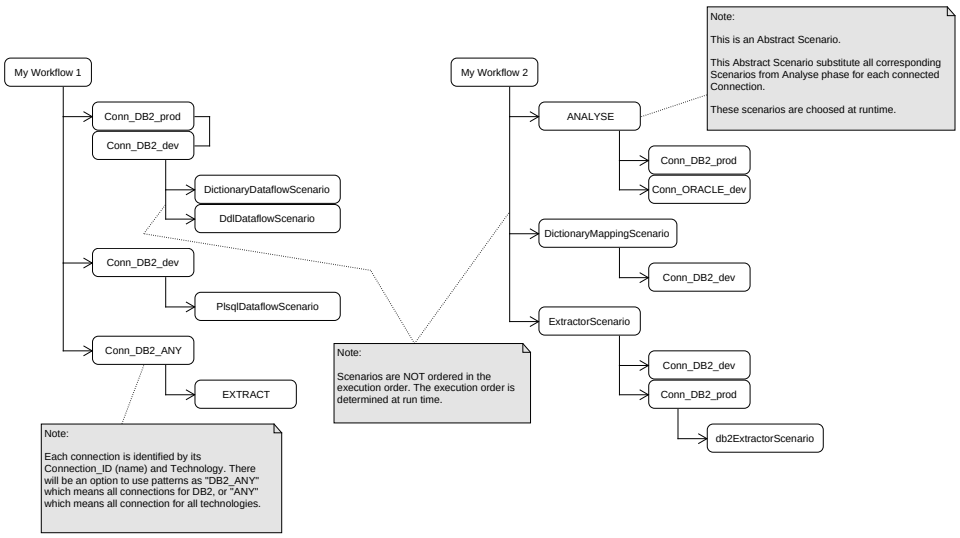

Figure 4.14: Connection Centric (left) vs Scenario Centric (right) Approach

The execution of only one specific Scenario using Scenario centric approach leads to the same tree structure as the Connection centric: Abstract Scenario → Connection → specific Scenario.

In the end, the Connection centric approach was chosen for the tree structure, but its format is not final yet. In the next chapters, the Connection centric approach will be extended for Phases which is using the idea of Abstract Scenario from Scenario centric approach, and Connections are divided into Technologies and Connections objects.

4.4 Orchestration Component Architecture

This section desribes orchestration component architecture, focusing on used Spring Beans and their usage represented by the sequence diagram. Part of this chapter is looking on Workflow definition class structure, including its JSON format. The end of this chapter is focused on a deeper analysis of MANTA Platform and its capabilities.

4.4.1 Component Architecture

The diagram 4.15 shows the high-level architecture of the Orchestration components in scope of layers. The diagram also shows links to the database tables and dependency between components.

WorkflowExecutionController WorkflowExecutionController defines and handles application interface (API) for Workflow Execution logic which

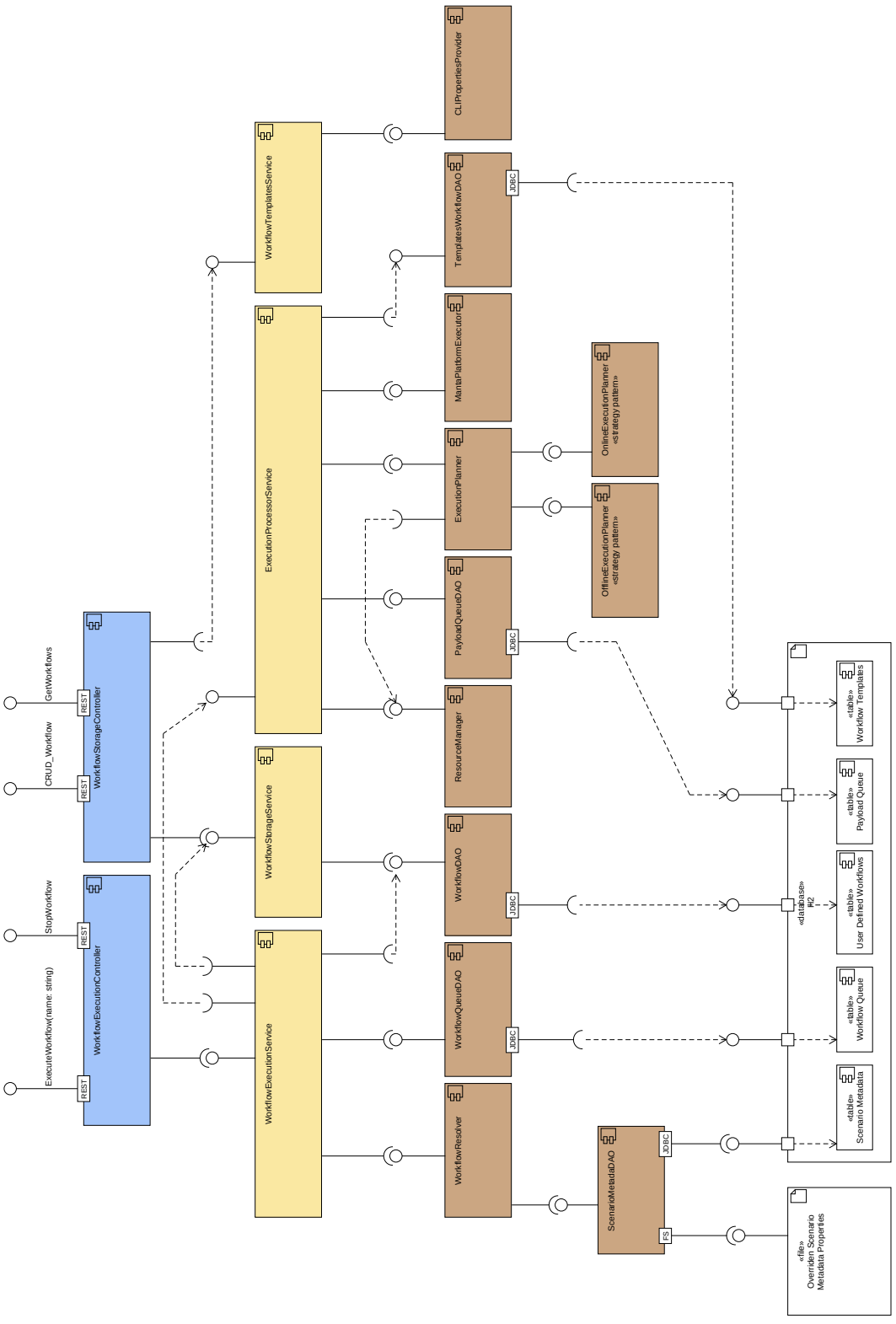

Figure 4.15: Orchestration Component Architecture Diagram

is primarily used by React application.

WorkflowStorageController WorkflowStorageController defines and handles application interface (API) for creating, updating and deleting Workflows stored in the database.

WorkflowExecutionService WorkflowExecutionService provides the business logic for preparing Workflow for execution.

ExecutionProcessorService ExecutionProcessorService handles the logic of execution of particular Workflow in a separate thread. It creates an execution plan, prepares the parallelisation and calling the MANTA Platform Executor when a Scenario should be started.

UserWorkflowStorageService UserWorkflowStorageService provides logic and transformation for storing, updating and deleting of existing Workflows.

WorkflowTemplatesService WorkflowTemplatesService provides operation for handling Workflow Templates created by MANTA.

WorkflowResolver WorkflowResolver transforms Workflow from JSON to class structure. It resolves abstract Scenarios and Scenario patterns.

WorkflowQueueDAO WorkflowQueueDAO handles storing Workflows waiting for execution. It is an adapter for accessing the persistent queue.

ResourceManager ResourceManager monitors usage of machine resources as MANTA Server Read or Write weights and machine CPU weight.

UserWorkflowDAO UserWorkflowDAO handles storing Workflows created by a user. It is an adapter for accessing the database.

PayloadQueueDAO PayloadQueueDAO handles storing Workflow's jobs waiting for execution. It is an adapter for accessing the persistent queue.

ExecutionPlanner ExecutionPlanner is logic for online and offline planning Workflow execution. It orders jobs and plans their parallelisation. It is a composite of offline and online planner.

CLIExecutorDAO CLIExecutorDAO executes Scenarios via MANTA Platform.

TemplatesWorkflowDAO TemplatesWorkflowDAO is an adapter for accessing stored Workflow Templates created by MANTA.

CLIPropertiesProvider CLIPropertiesProvider provides access to properties created by MANTA Configurator and used by MANTA CLI.

4.4.2 Workflow Class Diagram (Workflow Definition)

The following diagram 4.16 shows the Workflow JSON format and configurable Scenario metadata. The diagram has two views showing the class structure with their links, and the keyword 'exported' shows what can be stored as JSON. This final structure is motivated by chapter 'Scenario centric vs Connection centric approach'.

The definition format has a tree structure. If the inner node has no descendants, then its descendants are dynamically added at the run-time from a known and existing MANTA CLI configuration created by MANTA Configurator. This feature has been already mentioned as execution object wildcard, or shortly wildcard. The structure is illustrated in figure 4.16.

Scenario metadata are stored in persistent storage (a database) or it can be overridden in the special file which contains overridden attributes in JSON format.

As it was mentioned in the functional requirement 'F3 (U6, U7): Execute and Terminate Workflow', for every node, it must be possible to define new environment variables or override existing global environment variables. These variables are included in Workflow JSON. During run-time and after resolving JSON, the application gathers all user overridden environment variables, links them to the particular MANTA Scenario, and merges them with environment variables from MANTA linked to ScenarioMetadata.

All hierarchical arrangements have the same expressive capabilities and are transformable to each other, and therefore do not restrict display in GUI. This arrangement was chosen because it is simpler to create it manually, and creating in this order is the most likely use-case. Described by examples:

Example A: If users want to create a workflow 'ALL for Technology A', then they must:

1. define Workflow,
2. in the Workflow, they must define all Phases (max. 3)
3. and for each Phase, they must specify Technology A.

That is maximally seven records at total (Workflow, 3x Phase, 3x Technology). If Technology would be a parent of Phase, there are only two records maximally.

Example B: If users want to create a workflow 'Analysis for ALL Technologies' and Technology would be a parent of Phase, then they must:

1. define Workflow,
2. in the Workflow, they must define all Technologies (max n),
3. and for each Technology, they must specify Analysis as the Phase.

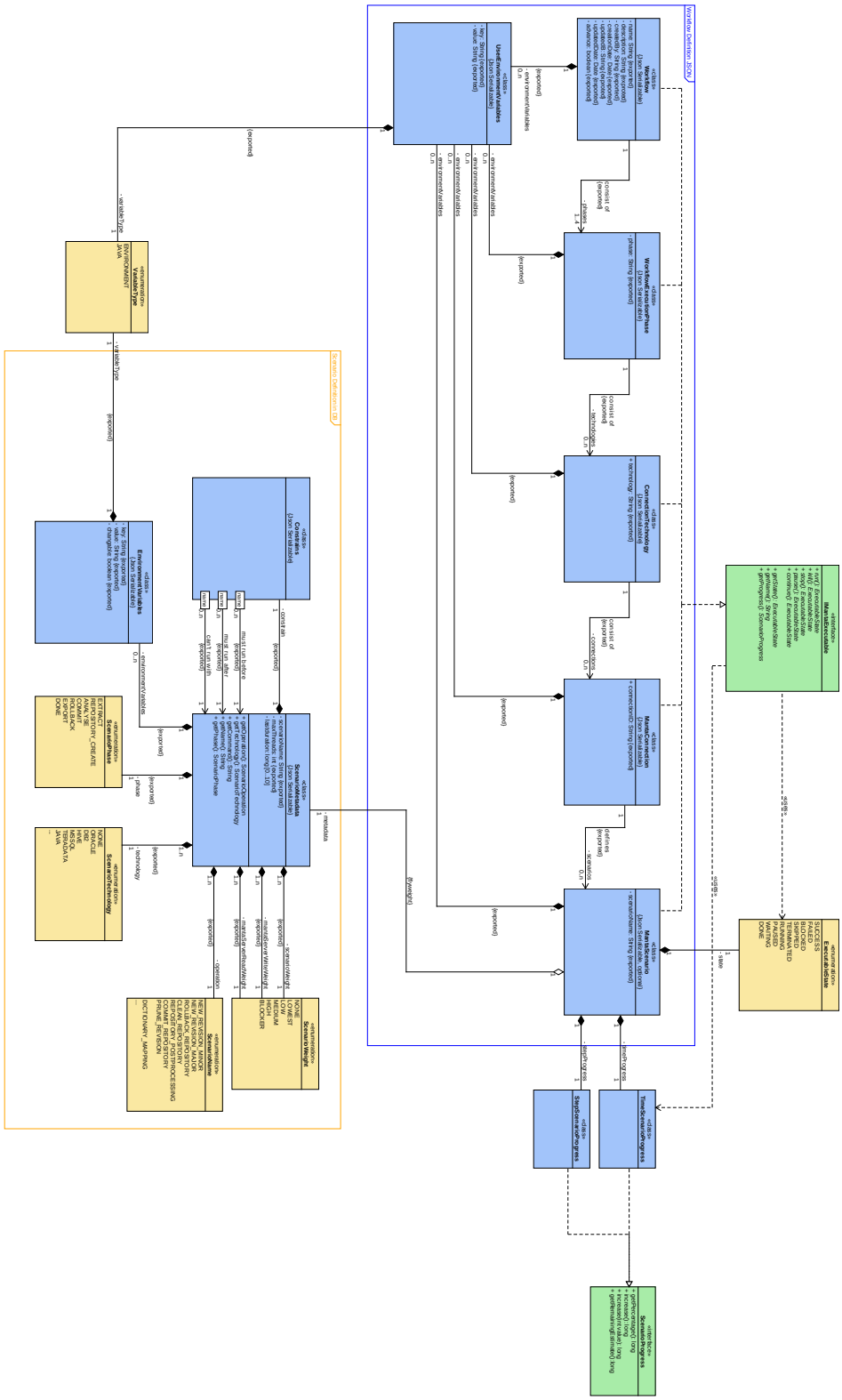

Figure 4.16: Workflow Class Diagram

That is 1+n+n records for the JSON output. In the other hand, if the Phase would be a parent for Technology, it would be two records maximally.

So to summarise this approach. To chose the correct hierarchy, it is better to construct it from the nodes with the lowest variability, considering the most probable use-case. From Example B, it is evident that making Phase parent of Technology leads to fewer records in the JSON.

4.4.3 Sequence Diagrams

Sequence diagrams in this chapter and attachments show usage of the components from the previous chapter. Diagrams were created primarily for understanding the complex program and finding critical sections.

In this section, there is displayed the most important diagram 4.17 of the Workflow execution, which is one of the core functionalities. The diagram shows the Workflow's lifecycle:

1. the translation Workflow from JSON,
2. storing it in the waiting queue,
3. obtaining it from the waiting queue by `WorkflowProcessor`,
4. expanding wildcards, including:
    a) obtaining all existing Connection configurations,
    b) Scenario Metadata injecting
5. creation of the workflow plan,
6. obtaining a job from the plan and its execution by `JobExecutionProcessor` including:
    a) obtaining Connection properties
    b) the Scenario execution.

In the same diagram 4.17, there can also be found critical sections highlighted by a colour box. There are three sections: storing and obtaining a Workflow to and from waiting queue; grabbing a job from the Workflow to execute it by `JobExecutionProcessor`; and access to the `ResourcesService`.

4.4.4 Executing MANTA Platform

MANTA Platform is a Java application with the Main method instantiating MANTA Scenario, and Scenarios are passed by its name as an argument. This Platform now mainly launches the Master Scenarios for each Technology, and the Master Scenario provides a parallel run of the corresponding Scenarios in

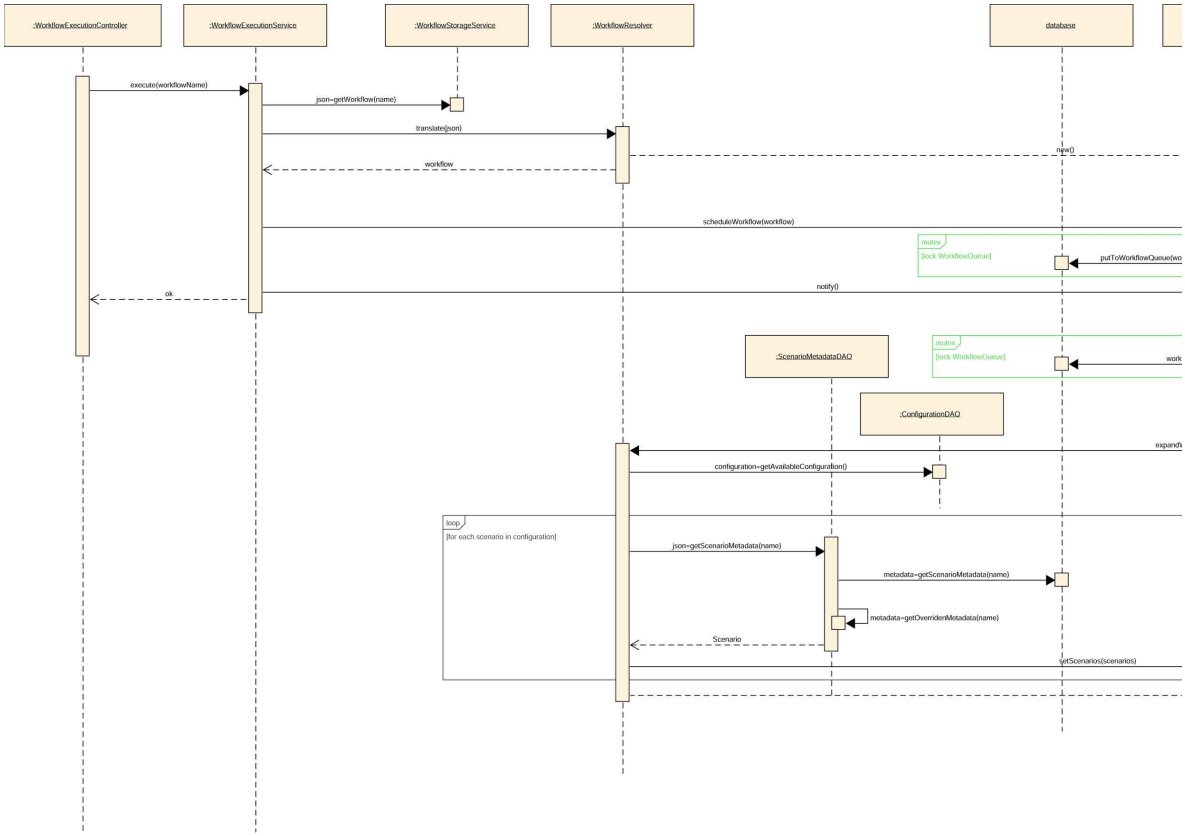

Figure 4.17: Workflow Execution Sequence Diagram (left)

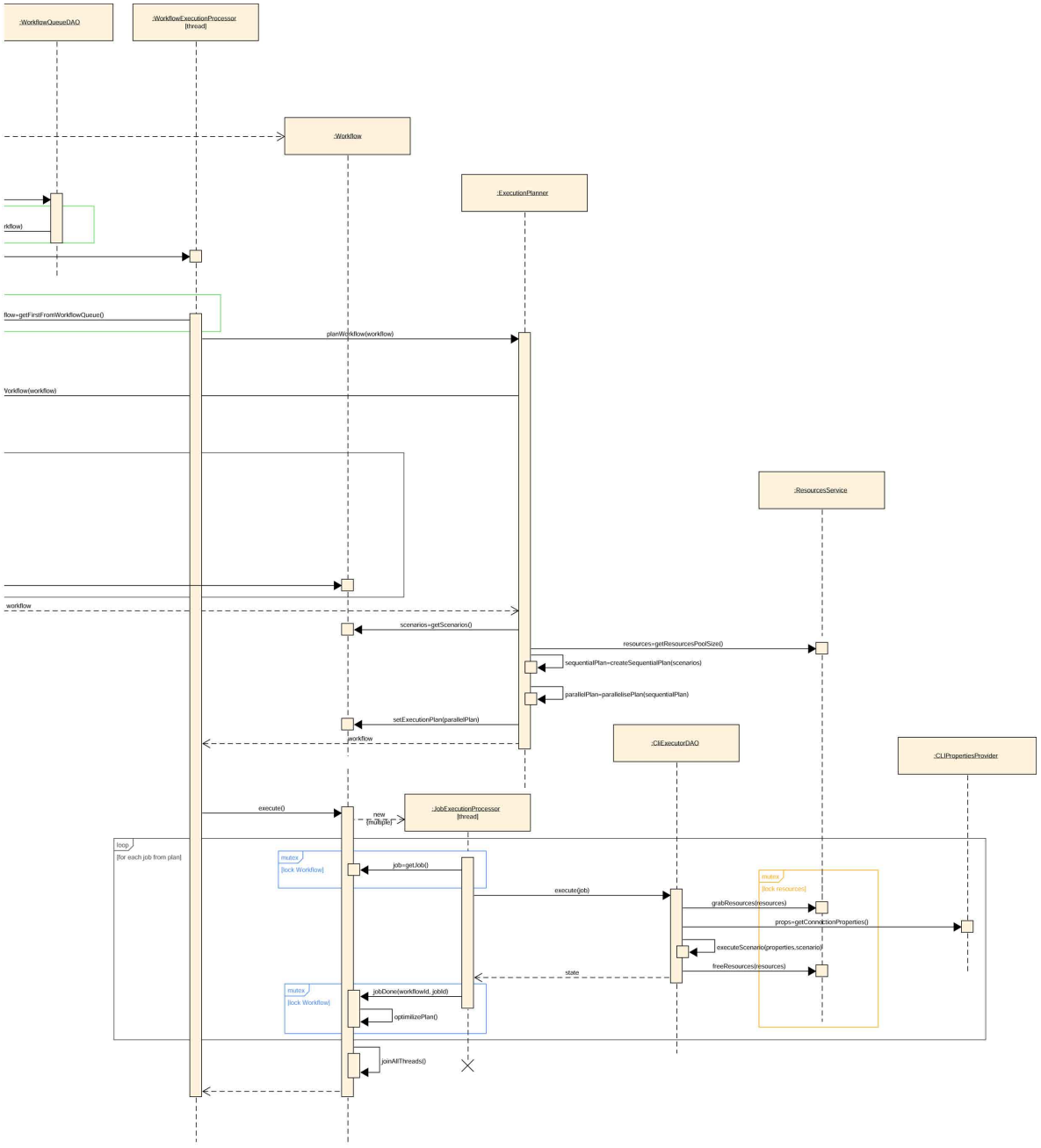

Figure 4.18: Workflow Execution Sequence Diagram (right)

the Technology for all Connections. The Manta Platform started using several level launch scripts: .bat for Windows or .sh for Linux.

The first level is just one script, which is the main MANTA CLI launch point, called _RUN and executes a complete MANTA CLI process for all existing Connections. This script executes three second-level scripts. These scripts are three aggregation scripts for individual technologies divided into three Phases: Extraction, Analysis and Export. Each of these three scripts contains a list of call commands to run scripts of the third level for all technologies for the particular Phase. The third-level scripts so-called Master Scenario scripts launch the MANTA Platform script with two arguments. The first argument is the module, and the second is the Scenario name, more specifically, the name of Master Scenario.

Historically, MANTA has supported a total of three modules: the general module, the module for IGC and the module for IMM. MANTA already integrated IGC module into the general module, and they are now backing away from the IMM module and slowly merging it into a generic module. MANTA Orchestration and Monitoring (and the entire MANTA Admin UI) supports only the general manta-dataflow-cli module.

MANTA Platform script (`mantar.bat`, `mantar.sh`) is a lowest-level script that runs the Java virtual machine with JAR applications MANTA Platform and passes the script name as an argument to it. This script sets up a lot of environmental variables, which are then passed along as arguments to the aforementioned JAR. MANTA Platform script also calls script configure.bat (or `configure.sh`), which sets other environment variables. These variables (in the configure script) are designed to be changed by users. See the list below for more information on individual variables.

Execution of the MANTA Platform is done by `ProcessBuilder` [23] because it is native, simple, and it does not require a third-party library. Its support for process controlling operation is also done by a native class `Process` which seems to be enough for requirements by this application [24]. The use of these libraries must be properly encapsulated so their future change will be smooth and clear.

Environment Variables used during Scenario execution by MANTA Platform

Environment Variables in `mantar.bat` and `mantar.sh`:

MEMORY_OPTS Variable `MEMORY_OPTS` is only configurable variable in MANTA Platform script. Its value defines initial and maximal memory used by Java Virtual Machine. In some cases, the default value defined by MANTA is not large enough, and it must be overridden. This value will be gathered from the system, and if it is undefined, the default value will be used. The default value is `-Xms128m -Xmx3072m -Xss4m`.

JAVA_CMD `JAVA_CMD` variable contains a path to Java executable. Its default value is 'java' unless `JRE_HOME` or `JAVA_HOME` is defined. The hierarchy is: always use `JRE_HOME` if it is defined, otherwise use `JAVA_HOME` if it is defined, otherwise use default value java. This logical functionality will be kept as-is, but it will be implemented in the application.

MANTA_DIR_HOME It is a path to MANTA CLI root directory. Originally this value was determined as the grandparent of the working directory (for windows `%~dp0..\..`, for a Linux `$MANTA_DIR_BIN/../..`). This variable is passed to MANTA Platform by redefining a property value `-Dmanta.dir.home` via `PLATFORM_OPTS`. This variable was originally used to define `PLATFORM_OPTS`. This logic will be changed a little bit. If this variable is not defined, then it will be resolved on run-time using a knowledge of the relative path of the working directory of this application to MANTA CLI.

MANTA_DIR_BIN This variable is a parent directory of MANTA Platform script where this variable was also set. It was originally used to define `MANTA_DIR_HOME` and to call configure script, but configure script will not be called anymore. This variable is unchangeable, and it will be resolved from `MANTA_DIR_HOME`. Its default value is `<MANTA_DIR_HOME>/platform/bin`.

MANTA_DIR_PLATFORM This variable is a path to the parent directory for MANTA home directory. It was originally used to define variable `MANTA_CLI_JAR`. This variable is unchangeable, and it will be resolved from `MANTA_DIR_HOME`. Its value is `<MANTA_DIR_HOME>/platform`. It is passed to the MANTA Platform by redefining a property value `-Dmanta.dir.platform` via `PLATFORM_OPTS`.

MANTA_CLI_JAR In this variable is stored path to MANTA Platform jar. This variable is unchangeable, and it is used as an argument for Java executable. MANTA Platform jar contains a CLI version. It must be obtained during run-time and dynamically resolved from `MANTA_DIR_PLATFORM` variable. Its possible value could be, for example, `<MANTA_DIR_HOME>/platform/lib/manta-platform-cli-1.27.jar`.

MANTA_DIR_SCENARIO This variable contains the path to the directory of MANTA CLI module which were being passed to MANTA Platform script. As it was mentioned before, this application will not support other modules than the general one. This variable is passed to the Platform by redefining a property value `-Dmanta.dir.scenario` via `PLATFORM_OPTS`. Its default value is `<MANTA_DIR_HOME>/scenarios/manta-dataflow-cli`.

```
1 -Dmanta.dir.home="<MANTA_DIR_HOME>"
2 -Dmanta.dir.platform="<MANTA_DIR_PLATFORM>"
3 -Dmanta.dir.scenario="<MANTA_DIR_SCENARIO>"
4 -Dmanta.scenario.name="<MANTA_SCENARIO_NAME>"
5 -Dmanta.license.file="<MANTA_LICENSE_FILE>"
6 -Dmanta.license.loader="<MANTA_LICENSE_LOADER>"
7 -Dmanta.license.warndays=<WARN_DAYS>
```

Listing 4.1: Default value of PLATFORM_OPTS variable

MANTA_LOG This variable is used to define the logging configuration for `LOGGING_OPTS`. Its default value is `<MANTA_DIR_HOME>/platform/etc/log4j2.xml`. This variable is unchangeable.

MANTA_LICENSE_FILE In this variable is stored path to MANTA License file. Its default location is `<MANTA_DIR_HOME>/platform/etc/license.key`. This variable is changeable, that means if a user redefines this variable in his environment, then this redefined value will be used instead of its default value. This variable is passed to the MANTA Platform by redefining a property value `-Dmanta.license.file` via `PLATFORM_OPTS`.

WARN_DAYS This variable originally did not exist, but it will be introduced to achieve a unified approach for every variable. It defines when the user will be notified of his license expiration. This variable is passed to the MANTA Platform by redefining a property value `-Dmanta.license.warndays` via `PLATFORM_OPTS` and is changeable. Its default value is 7.

PLATFORM_OPTS This variable contains arguments which redefine property value for MANTA Platform. Originally, it was passed as an argument for Java and dereferenced by Shell. This dereferencing will be done in the application. Its default value may be seen in the listing 4.1.

LOGGING_OPTS This variable contains an argument which redefines property value `-Dlog4j.configurationFile` for MANTA Platform. Originally it was passed as an argument for Java and dereferenced by Shell. This dereferencing will be done in the application. If the `log4j` configuration file exists then is used, otherwise `<MANTA_LOG>` is used. This variable is changeable. The most priority has users specified value then the existence of `<MANTA_DIR_SCENARIO>/etc/log4j2.xml` and the last is the default `<MANTA_LOG>`.

MANTA_SCENARIO_NAME This variable did not exist, but same as for `<WARN_DAYS>` it will be introduced to achieve a unified approach for every variable. It is unchangeable, and it contains the Scenario name

usually passed as an argument for MANTA Platform script. It is passed to MANTA Platform via `PLATFORM_OPTS`.

MANTA_LICENSE_LOADER This environment variable is defined in the third-level Master Scenario script. This variable contains the reference to java bean loading the license. Reason for exposing this is not valid anymore, and it will be hardcoded for better safety and for preventing it from foisting by a forged implementation. This variable is passed to the MANTA Platform by redefining a property value `-Dmanta.license.loader` via `PLATFORM_OPTS`.

SCRIPT_DIR Environment variable `SCRIPT_DIR` is defined in the third-level Master Scenario script. It contains parent directory of the script to which script does 'change directory' and makes it as a working directory. The working directory is `<MANTA_DIR_HOME>/scenarios/manta-dataflow-cli/bin`. This application will resolve the working directory from `MANTA_SCENARIO_DIR` and will not let the user change it. This variable will also be renamed to `MANTA_SCRIPT_DIR` to keep naming convention, but it affects nothing because this variable will be unknown for a user.

JAVA_OPTS This variable is used only in few of third-level Scenario Scripts. For example, it is used for `mssqlExtractorMasterScenario` and both operating systems. By default, its already set value is kept unchanged and in the script is added a reference to MANTA Scenario library directory. Its default value can be easily understood from the next bash code: `export JAVA_OPTS=("${JAVA_OPTS[@]}" "-Djava.library.path=\"$SCRIPT_DIR/../lib\"")`

Jakub Moravec asked for keeping this behaviour as is if it is possible. That means defining this variable only for those Scenarios which need it. That means its definition will be in the database for particular Scenarios.

DEBUG_OPTS This variable is defined in the script `mantad`, and it is used by MANTA Developers to debug MANTA CLI in Eclipse. If the `mantad` script is called, then its default value is:

```
DEBUG_OPTS=-Xdebug -Xrunjdwp:transport=dt_socket,server=y,address=8000,suspend=y
```

The `mantad` script calls MANTA Platform script (`mantar`) and MANTA Platform script passes this variable to MANTA Platform. This variable will be in default blank because its default value is undefined/blank, but it will remain changeable. So a user (MANTA developer) can define its value for his debugging needs. However, they will not probably be using this application during development for debugging purpose, because of backward compatibility they may still use already existing scripts.

SCENARIO_OPTS Variable `SCENARIO_OPTS` is usually defined by users to customise MANTA Scenarios somehow. Its default value will be blank, and a user should be able to redefine it.

Environment Variables in `configure.bat` and `configure.sh`:

MANTA_USER `MANTA_USER` contained the username of the user who executed the configure script. This username will hold the username of the user who started this application. This variable is passed to MANTA Platform via `WORKSPACE_OPTS` as `-Dmanta.user`. This variable is unchangeable.

MANTA_DIR_USER This variable, if the user does not redefine it, contains `<MANTA_DIR_HOME>`. That means this variable is changeable. This variable is passed to MANTA Platform via `WORKSPACE_OPTS` as `-Dmanta.dir.user`.

MANTA_DIR_INPUT In this variable, if the user does not redefine it, is `<MANTA_DIR_HOME>/input`. That means this variable is changeable. This variable is passed to MANTA Platform via `WORKSPACE_OPTS` as `-Dmanta.dir.input`.

MANTA_DIR_OUTPUT `MANTA_DIR_OUTPUT`, if the user does not redefine it, contains `<MANTA_DIR_HOME>/output`. That means this variable is changeable. This variable is passed to MANTA Platform via `WORKSPACE_OPTS` as `-Dmanta.dir.output`.

MANTA_DIR_LOG In this variable, if the user does not redefine it, is stored `<MANTA_DIR_HOME>/log`. That means this variable is changeable. This variable is passed to MANTA Platform via `WORKSPACE_OPTS` as `-Dmanta.dir.log`.

MANTA_DIR_TMP This variable, if the user does not redefine it, contains `<MANTA_DIR_HOME>/temp`. That means this variable is changeable. This variable is passed to MANTA Platform via `WORKSPACE_OPTS` as `-Dmanta.dir.temp`.

MANTA_LICENSE_VALIDATOR Variable `MANTA_LICENSE_VALIDATOR` is similar to `MANTA_LICENSE_LOADER` apart from the fact; it is defined in configure script. It will also be ignored and replaced by its default value.

WORKSPACE_OPTS This variable contains arguments which redefine property value for MANTA Platform. Originally, it was passed as an argument for Java and dereferenced by Shell. This dereferencing will be done in the application. Its default may be seen in listing 4.2.

```
1 -Dmanta.dir.home="<MANTA_DIR_HOME>"
2 -Dmanta.dir.platform="<MANTA_DIR_PLATFORM>"
3 -Dmanta.dir.scenario="<MANTA_DIR_SCENARIO>"
4 -Dmanta.scenario.name="<MANTA_SCENARIO_NAME>"
5 -Dmanta.license.file="<MANTA_LICENSE_FILE>"
6 -Dmanta.license.loader="<MANTA_LICENSE_LOADER>"
7 -Dmanta.license.warndays=<WARN_DAYS>
```

Listing 4.2: Default value of WORKSPACE_OPTS variable

Most of the variables become locked from changes. It is not bad because they were never meant to be changeable by users. It even adds some security to MANTA, because customers will not be able to change them inappropriately and break their MANTA CLI.

Most of these variables will have their default values hardcoded in Java in an Infrastructure package. It also is not bad because their default values never change, and they all are common for all scripts, and most of them refer to the working directory structure. These which are not shared are open to change in ScenarioMetadata or by users in GUI and/or in Workflow Definition JSON.

Manta Platform is nowadays executed on Windows by command: `%JAVA_CMD% %DEBUG_OPTS% %JAVA_OPTS% %MEMORY_OPTS% %PLATFORM_OPTS% %WORKSPACE_OPTS% %LOGGING_OPTS% %SCENARIO_OPTS%-jar "%MANTA_CLI_JAR%" %2 %3`

These all variables and its order must be kept in the application. Interesting is the third argument `%3`, which is "app context filename", and has been used in special situations to do tricks with MANTA CLI by MANTA developers. After discussion with the head developer (L. Hermann), he approved not to implement it because it is not relevant anymore.

Bash arrays used by MANTA Platform

In all-levels Bash scripts for Linux executing any Scenario, variables are created and used as Bash array. To verify possibilities to support the format of these variables a prototype-test which exported an array variable and tried to obtain it via `System.getenv(String)` or `ProcessBuilder.environment()` (which uses `System.getenv(String)`too) was created [23].

This test verified that standard way to work with environment variables does not support bash arrays. That is not unexpected behaviour because it conflicts with the definition of the environment variable :

"The value of an environment variable is a string of characters. For a C-language program, an array of strings called the environment shall be made available when a process begins. The array is pointed to by the external variable environ, which is defined as: `extern char **environ;`" [25]

"These strings have the form name=value; names shall not contain the character '='. For values to be portable across systems conforming to POSIX.1-2017, the value shall be composed of characters from the portable character set (except NUL and as indicated below). There is no meaning associated with the order of strings in the environment. If more than one string in an environment of a process has the same name, the consequences are undefined." [26]

That means Bash arrays cannot be easily supported in Java application and they will not be supported in this application.

Changes in MANTA Platform

MANTA Platform allows run any Scenario - Master Scenario and ordinary Scenario. The problem with running ordinary Scenarios is that they assume that the properties from the Connection configuration files that the Configurator creates are already loaded in the application's context and are directly available to them. However, this loading is handled by Master Scenarios, which are skipped because of the requirement of this application. Somehow, therefore, these property needs to be acquired and added to the context.

Some of the properties may be encrypted, and a decryption procedure must be called on these properties before the Scenario uses them.

Three variants are possible:

Passing all properties as application's arguments This first option expects the Monitoring and Orchestration to read the configuration file and forward all properties as `SCENARIO_OPTS` variable to the MANTA Platform. MANTA platform only reads and decrypts these properties. The advantage of this solution is the minimal change in the CLI and the MANTA Platform, as it is only needed to do decryption properties in `SCENARIO_OPTS`.

Passing Connection file path as application's argument The second option moves the reading of the property file containing the Connection to the MANTA Platform. The decryption remains in MANTA Platform. Location of the file is passed as an argument to MANTA Platform with Scenario name. This option is not optimal because a major change is required in MANTA Platform.

Executing special initial Scenario This last option is in basic same as the previous one, but the change is not in MANTA Platform, but a new special Scenario, which will handle the file reading and injecting properties into application's context, will be created. This extraction of the logic into separate Scenario, which will be executed only when required by Monitoring and Orchestration application, is a nice and clean solution and the most bullet-proof solution because no change in MANTA Platform is required.

For the need of the prototype, the first solution will be implemented, but it will be changed to the third one later.

4.5 Monitoring Component Architecture

Monitoring Component Architecture is not as sophisticated as the Orchestration Component. Its core functionality is to inject gathered states from CLI Scenarios into Scenarios Execution Data Objects that then may be shown to users. This will happen asynchronously when any Scenario finishes its work.

This solution is tightly coupled with Jakub Kováč's Log Viewer. Jakub created as his Bachelor thesis a logging solution for MANTA. Very simply said, he created a unified Logging API which should be widely used in MANTA as a logging framework. Its solution is focusing primarily on Errors and Warnings level logs which are parametrized because they will be displayed to users via GUI. These new parametrized logs contain, besides other parameters, a 'how-to-fix' parameter which should guide users to do a self-help fixes without needing to contact MANTA Help desk. [27]

After several discussion, he prepared as part of his solution a similar way to log the Scenario's states. This solution expects from the Orchestration and Monitoring tool to provide Workflow execution ID to identify to which Workflow the Scenario's execution belongs, and Scenario execution ID to identify particular Scenario's execution across the application.

Based on these two IDs, the Monitoring component can obtain Scenario's states very easily by calling Log Viewer' API with filters. The only drawback of this solution is that MANTA developers must rewrite current logging to new logging which uses Jakub's Logging API.

4.6 Public API

Like all modern tools, Orchestration and Monitoring has a public API designed to facilitate integration with the outside world. APIs will mainly be used by partners, who will be able to extend their tools with functionalities provided by this tool, but it will also be used by customers to automate their MANTA-related processes.

Most other tools from the Admin UI has the public API already implemented, and thus the API implementation in this application will follow the standards already defined. The API on the presentation layer is implemented using Spring Web Binding Annotation and documented in Swagger using the Swagger Annotation library. The API is secured using a shared configuration across the Admin AI using Spring Security. The API version is resolved using the version number in the resulting URI.

We can split APIs into two logical units: Workflow Storage, which handles the persistence of user Workflows and templates; and Workflow Execution,

which handles the startup and termination of Workflows and provides its state or output from Extraction. Description for Workflow Storage may be found in table 4.2 and description for Workflow Execution may be found in table 4.3.

Method	**URI**
Description	
GET	**/workflow/templates**
Provides a list of all available templates with their information as a description and name but without definition.	
GET	**/workflow/templates/{templateName}**
Provides information about template as the previous endpoint, including Workflow-template definition.	
GET	**/workflows**
Provides a list of all existing Workflows created by users with information as creation time, creator's name, updated time, updater's name and description. This endpoint does not provide the Workflow definition.	
GET	**/workflows/{workflowName}**
Provides same details as the previous endpoint, including Workflow definition.	
POST	**/workflows?name={name}**
To create a user-defined Workflow, this endpoint must be called with the Workflow definition in body and its name as parameter.	
DELETE	**/workflows/{workflowName}?force={false}**
This endpoint deletes not running Workflow. If the Workflow has a pending execution, the force parameter may be used to cancel it and delete the Workflow.	
PUT	**/workflows/{workflowName}**
Updates the Workflow definition and description. The updated timestamp and username is set for the Workflow.	

Table 4.2: Workflow Storage Endpoints

4.7 Data Persistence and User Customization

The main problem with the configuration files in MANTA (which caused that MANTA Updater was created) is that the configuration files are already created and defined in advance and shipped with the product together. Therefore, if a user changes something in these files, they will lose their changes when

Method	**URI**
Description	
GET	**/executions**
List all execution with information but without the execution details for every Scenarios. This list may be filtered using parameters and it is returned in pages.	
POST	**/executions?name={workflowName}**
Executes user-defined workflow.	
GET	**/executions/{executionId}/status**
Returns the detailed information about particular execution with list of planned Scenarios with their execution details.	
GET	**/executions/{executionId}/output**
Downloads output from export Scenarios	
DELETE	**/executions/{executionId}/terminate?force={false}**
Cancels pending Workflow or if the force parameter is true terminates running Workflow.	

Table 4.3: Workflow Execution Endpoints

these files are updated. This is a consequence of the application update, as the update does not merge these files in any way and they are not in any way process-updated, a new version only replaces them. This design (described below) solves the problem by creating property files on runtime, and these files are not part of the shipment thus they are not replaced. Default MANTA Scheduling and Monitoring configuration is defined in the database, and the user only overrides it in a separated file.[32] This solution is update-safe in MANTA context. Another benefit to keeping user-defined configuration in external files instead of another structure (e.g. database) is the ability to change it directly without GUI.

Once installed, the user should be able to use this application with minimal limitation even if there isn't any record in the user-defined configuration files. This means that once installed application should not require any other user's intervention to start using it. To achieve this goal, anything that must be configured in advance to start using this application will be written by the MANTA Installer during installation (or during update), and a default value will be used. For example, the application should not require additional links to other MANTA applications because Installer can provide it, the application should not require resources configuration because the application can use default values and it will be sufficient. Or a user doesn't have to create

[32] recommended by Jan Ulrych

their own Workflows because he or she can use default workflows prepared by MANTA in advance.

The database update of the structure and entries will be done using Liquibase, where so-called change (update) scripts contain change-set which describes the changes from the previous version to the new version. If there is a multi-version update, then more change scripts are applied to ensure table transformation and data migration.

"Liquibase is an open-source solution for managing revisions of your database schema scripts. ... The feature that is probably most attractive in Liquibase is its ability to roll changes back and forward from a specific point — saving you from needing to know what was the last change/script you ran on a specific DB instance."[28]

The developer provides scripts (referred to as "changesets") during the implementation phase. He or she creates a change script which transforms a database from the previous state to the new state. The framework executes these scripts on the application's startup. The library checks their previously saved metadata about the database, which are stored in the database. Based on this information, the framework knows which change scripts weren't applied and applies them. The framework also checks check-sums of update scripts, if there is some inconsistency the application won't load. To prevent this, the developer must not change already published change scripts.

Most of the data database holds are MANTA declared data because the user's configuration is stored in separated and easily accessible files. This minimalise changes over user-data and the database is more open to data-wipe. Nevertheless, developers must proceed with caution because the database holds execution statistics and user-defined workflows. Losing it would be critical. These user's data are loaded on-demand by the appropriate DAO bean, whether from the database or configurable properties files.

User-configuration properties are always stored in separated files in key-value format. User will be able to find configurable properties in the application's user documentation. The update process must never change these files directly, and a few rules must be kept in mind:

- Adding new property is trivial. The default value is known in the database, and if the user needs to change it, they simply add it to the configuration file with their own value.
- If a property is no longer needed, it will be ignored by the application. The application will log a warning.
- All changes to the name, format or data type of property should be avoided. The suggested solution is to start ignoring the old property and add a new one with a default value. If this happens, then users must be notified in Log Viewer and in application GUI about old property in configuration, and this change must be written in release notes.

The other solution is to implement a property file transformation. This forces property files to begin to be versed, and the need to implement its transformation and migration from version to version, similarly as Liquibase does. MANTA wants to avoid this if possible, Jakub Moravec said.

The database used for this project is H2. The reason for its use is a non-functional requirement from MANTA, which was explained by the fact that it is a lightweight database with which, from their experience, clients do not have any problem.

4.7.1 Database Model

This subsection shows the Entity-Relationship model (figure 4.19) used by this project. The known drawback of this model is that Workflow's definition, the JSON, is stored as a CLOB. Structural change in this JSON will cause a migration problem that liquibase cannot solve easily. A special migration script will be required for some cases of structural change.

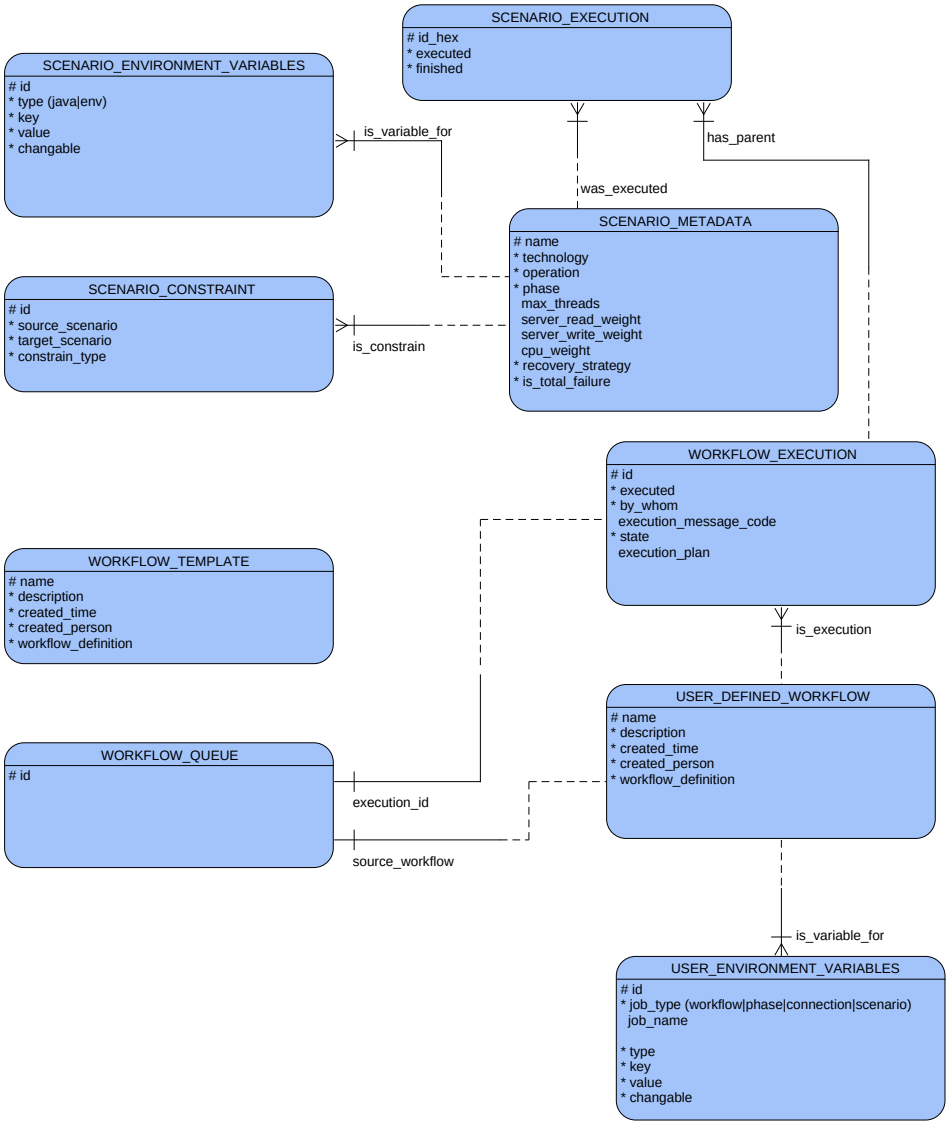

Figure 4.19: Monitoring and Orchestration Database Model

Chapter 5

Implementation

This chapter describes the prototype implementation, shows some examples from implemented application and shortly talks about testing.

5.1 Implementation

An application-module without a graphical user interface with implemented API was created as a prototype implementation. Infrastructure, repository and service layers with all required logic were implemented for this thesis. The API was implemented in cooperation with other MANTA Developers. The application supports getting, creating, updating and deleting User Defined Workflow; getting Workflow templates; executing Workflow, cancelling pending Workflow, getting existing Executions, getting the detail of Workflow Execution and getting a zip file with output from Export Scenarios. The application supports over thirty technologies that are over two hundred Scenarios.

Module Orchestration and Monitoring uses services provided by other MANTA modules, so it is not executable without them. For example, it uses Configurator to find a location of the MANTA CLI and also gets existing Connections from it.

Not implemented features: termination of running Workflow, Workflow failure recovery, Connections for technologies having special requirements on the environment (e.g. Csharp connector needs .NET Core) and parallelisation for Analysis Phase.

5.1.1 Workflow's Life Cycle

This subsection describes a Workflow's life cycle from its creation to its first execution.

1. `Workflow` is created and stored into DB.

2. – a user starts `Workflow` –
3. The copy of the `Workflow` is obtained from DB.
4. `WorkflowExecution` is created for the `Workflow`, and reference on the `Workflow` is stored inside it.
5. `WorkflowQueueItem` is created, and reference on the `WorkflowExecution` is stored inside it.
6. `WorkflowQueueItem` is pushed into `WaitingQueue` (in DB).
7. – time elapses –
8. `WorkflowExecutionProcessor` obtains `WorkflowQueueItem` and starts its processing.
9. The `WorkflowQueueItem` is destroyed, and the `WorkflowExecution` is reconstructed.
10. Base on the `Workflow`'s reference in `WorkflowExecution`, the `Workflow` is constructed.
11. Wildcards in `Workflow` are resolved using existing `Connections`. The `ExpandedWorkflow` is created.
12. If advance mode is turned off, the Essential Processing Scenarios are added to the `ExpandedWorkflow`, and a base on the `Workflow`'s revision type the `RevisionScenario` is chosen (major or minor).
13. `ExpandedWorkflow` is posted to the `WorkflowPlanner`.
14. `WorkflowPlanner` gets a copy of the `ScenarioDependencyGraph`. If the graph does not exist yet its created now.
15. `WorkflowPlan` is created based on inputs from `ExpandedWorkflow` and `ScenarioDependencyGraph`.
16. `WorkflowPlans` with all jobs (Scenarios) is executed using MANTA Platform.
17. – execution ended –
18. Results and states are gathered.
19. `WorkflowExecution` is updated by results and states from `WorkflowPlan`.

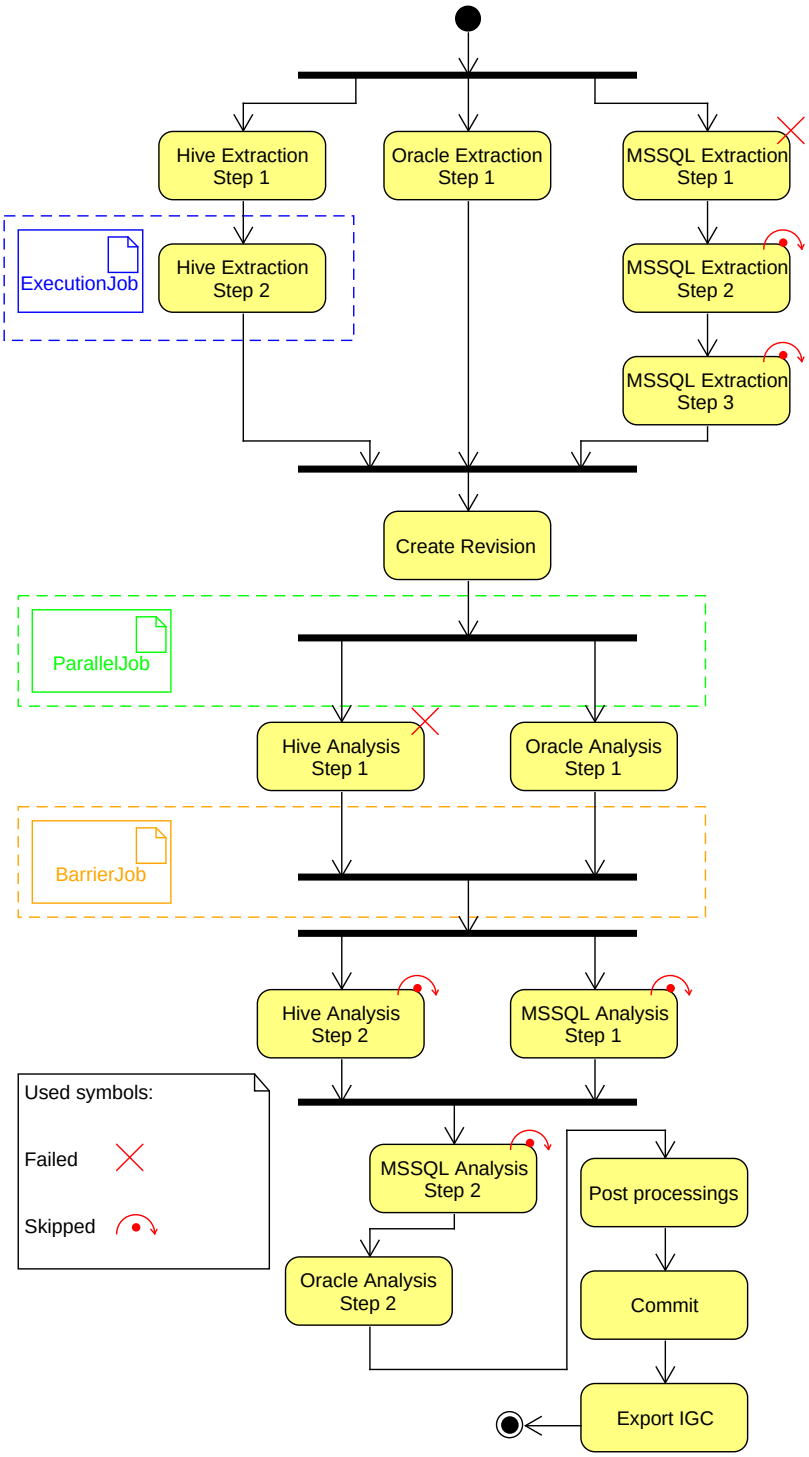

Figure 5.1: Workflow Plan Execution Example

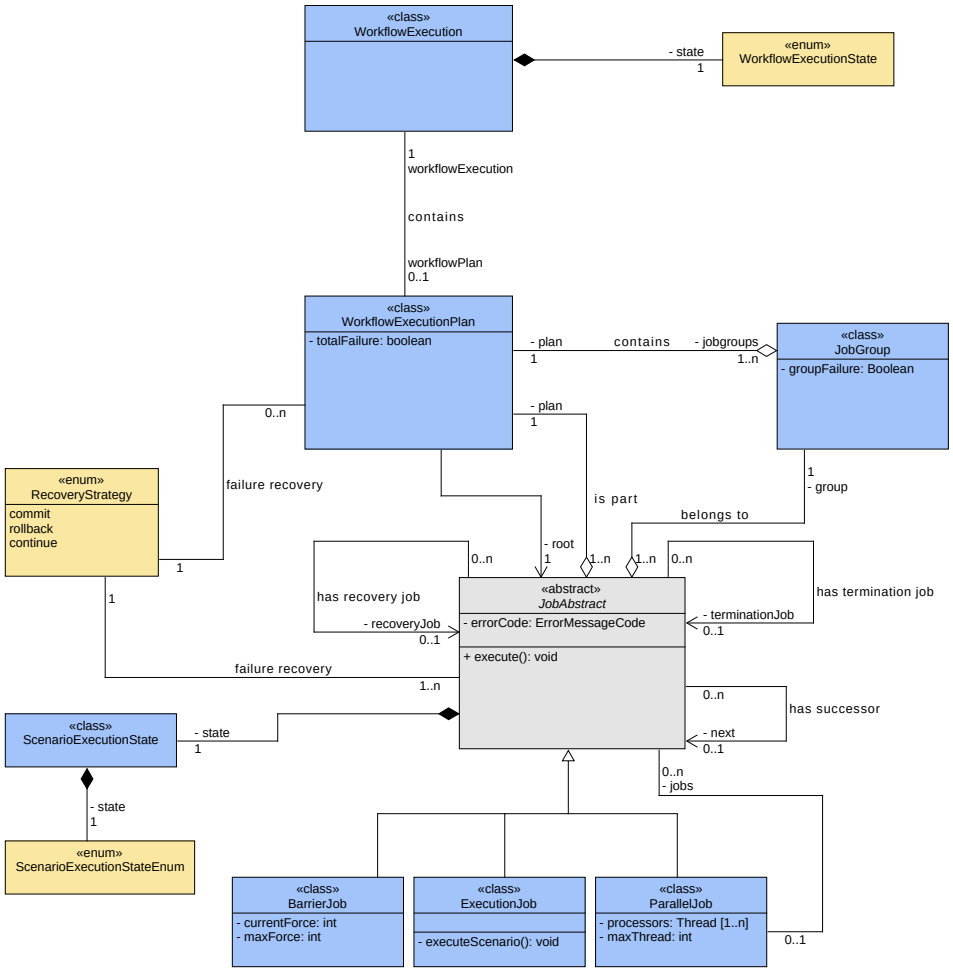

Figure 5.2: Workflow Plan Class Diagram

5.1.2 Workflow Plan

The Workflow Plan consists of three types of jobs: `SingleJob`, `ParallelJob`, `BarrierJob`. Each Job belongs to the execution group (`JobGroup`) identified by Connection technology and Connection ID. If one of the Jobs from execution group fails all other, which were not executed yet, are skipped. Single Job contains a Scenario to execute; Barrier Job synchronizes all threads and decides if it was broken and execution can continue, and Parallel job creates `FutureTask`s for all its children. All relations may be seen in figure 5.2.

The figure 5.1 shows an example of a Workflow plan with `ExecutionJob`, `ParallelJob` and `BarrierJob`. There also can be seen a usage of `JobGroup`, when 'MSSQL Extraction Step 1' failed the rest of Scenarios in the same group (also named MSSQL) are skipped.

Essential Processing Scenario List

If a user does not create the Workflow in advance mode, these Scenarios are added into it automatically at runtime.

- `diagnoseRepositoryScenario`
- `newRevisionScenario` or `newMinorRevisionScenario`
- `importDataflowScenario`
- `importLinksDataflowScenario`
- `importMetadataDataflowScenario`
- `commitRevisionScenario`
- `pruneRevisionScenario`
- `repositoryPostprocessingScenario`
- `exportRepositoryScenario`

The `diagnoseRepositoryScenarios` is always added but the rest is only added if any other Scenario from the same Phase exists.

Simple Planning Algorithm

The implemented Simple Planning Algorithm is an offline planning algorithm that creates `WorkflowPlan` base on Scenarios from the `ExpandedWorkflow` and `ScenarioDependencyGraph`.

This graph is created from Scenario Constraints which are predefined in the DB and contain rules like a 'Scenario A must be executed BEFORE—AFTER Scenario B'. Every vertex in `ScenarioDependencyGraph` represents one Scenario and holds its name. The directed edges represent an execution order, e.g. $(A) \rightarrow (B)$ means A must be executed before B. The `ScenarioDependencyGraph` contains all existing Scenarios in MANTA.

The algorithm takes an editable copy of `ScenarioDependencyGraph` and starts deleting vertices which are not in `ExpandedWorkflow`. Vertex deletion means that all its predecessors must be linked to all successors.

When this smaller or equally sized graph is created, all nodes are recursively processed one more time. The currently processed vertex is transformed to `ExecutionJob`, and the relevant `ScenarioMetadata` is linked into it.

Then the algorithm looks at the number predecessors of currently processed vertex, and if their number is greater than zero, a `BarrierJob` is created, and the `ExecutionJob` is linked as its successor. This `BarrierJob` will be returned from the recursion; otherwise, the `ExecutionJob` is returned.

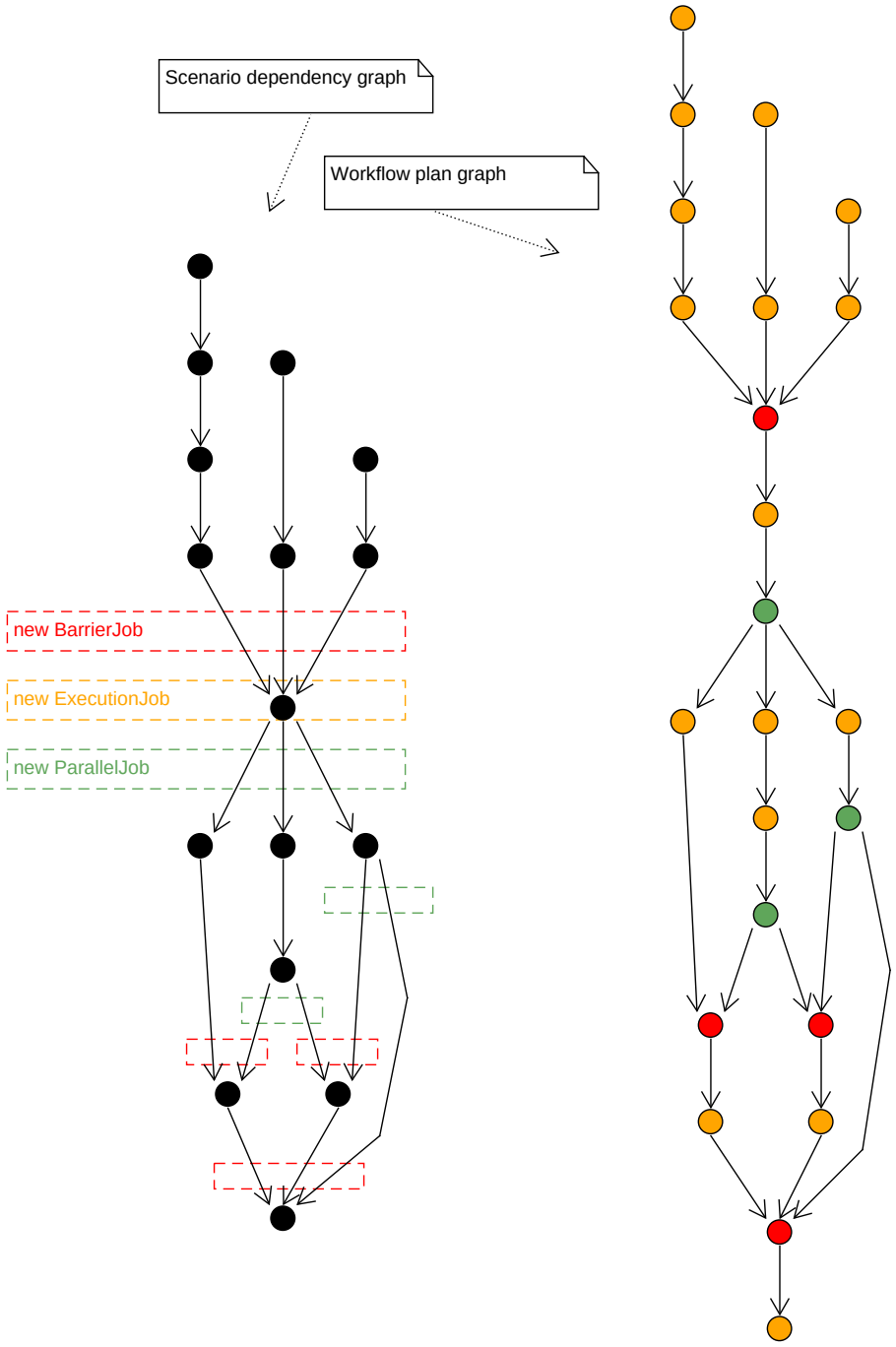

Figure 5.3: Simple Workflow Planner - transmutation from Scenario Dependency graph into Workflow plan example

Then the algorithm looks at the number of successors of currently processed vertex, and if their number is greater than zero, a `ParallelJob` is created, and the `ExecutionJob` is linked as its predecessor. Then this recursion is called on each successor of currently processed vertex, and the returned values are linked to the `ParallelJob` as its successors. If only one predecessor of the currently processed vertex exists, the `ParallelJob` is not created, and the `ExecutionJob` is linked directly on returned value from next recursion call. If no predecessor exists for the currently processed vertex, the `ExecutionJob` will not have any successors, and it becomes a leaf.

Every vertex remembers its return value, and when it should be processed again, it simply returns its already calculated value.

This last steps of the algorithm may be seen visualised in the figure 5.3.

The algorithm must be aware of that not all Scenarios can be executed in parallel. It is achieved by special cross-technology dependencies which the algorithm can handle. With Extraction, this is not a problem, and everything can be run in parallel. The local machine is not so overloaded by Extraction, but remote sources are. This is no longer the case with Analysis. For Analysis, the order to be maintained is that inputs from database machines must be analysed first, then integration technologies, and finally reporting tools. For export it is the same only for SSAS technology is an exception and it has to be exported before all other reporting tools.

Developers must bear in mind that the algorithm is working only on trees without backward references, but they may have multiple components and forward references. So a unit test were created, which builds the dependencies graph and looks for cycles.

5.1.3 Scenario Execution

This chapter shortly outlines the execution algorithm and implemented parallelisation.

The execution algorithm starts by executing the root node, which is always only one. It may be either a `ParallelJob` or `ExecutionJob`. From the planning algorithm above is evident that every Job has one successor or it is a leaf which means that execution for that branch ended. So before the Job finishes the execution, it creates a task for its successor, and that `FutureTask` is put into `ServiceExecutor`. Besides, the ending task sends the `Future` into WorkflowPlanBO where the main thread is notified and checks all known `Future`s which one has finished[33].

Both `ParallelJob` and `BarrierJob` are simple. The `ParallelJob` contains multiple successors which are all put into `ServiceExecutor`. BarrierJob has a final variable `power` and variable `Force`. If these two variables are equal,

[33]The main thread is also woken up periodically because the last notification is received before the Job is finished.

the barrier is broken, and it starts its successor. Otherwise, the Job ends and waits for its another execution to increase a `force` by one.

`ExecutionJob` is more complicated than the previous two. At first the `WorkflowPlanBO` is checked if this Job may be executed. If `WorkflowPlanBO` has the termination flag set, the Job changes its state and immediately executes the next one. Otherwise, it checks the `JobGroup` and again if `JobGroup` has termination flag set the Job changes its state and executes the next Job. If no termination flag is set, the Scenario execution may be proceed.

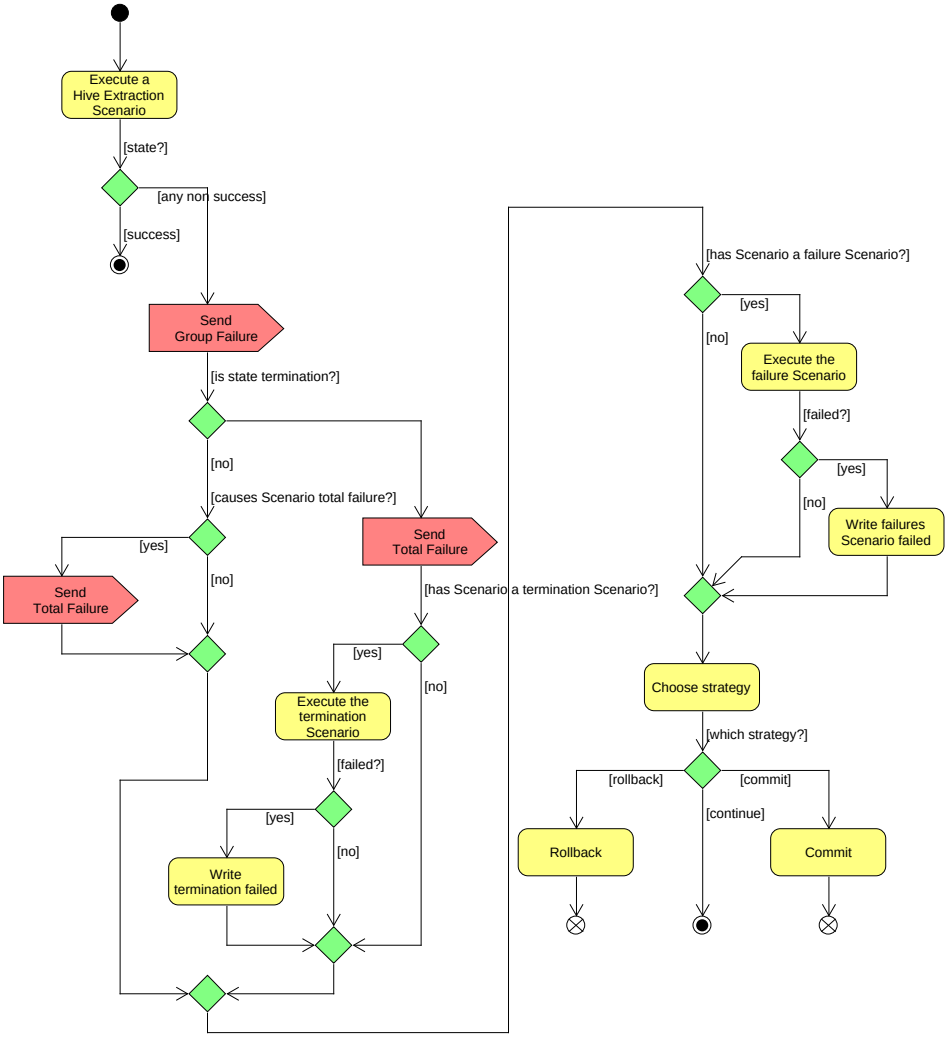

Figure 5.4: Scenario Execution with Failure Recovery Activity Diagram

5.2 Testing

During implementation, the integration tests and JUnit tests were created using Mockito to supplement dependencies. There are ninety-six tests which cover over 90 % of the infrastructure and repository layers. In order to save time for developing more vital functionalities and make the product on time, the service layer is covered only partially and creates a technical debt which must be atoned in the next iteration. Thus, the rest of the application and the API were tested primarily using Scenario tests.

Before the module's beta release and again before official release MANTA Quality Assurance team prepared and performed other thorough Scenario tests and acceptance tests. Three weeks before the official release, the module was sent as a beta to one of MANTA partners who started implementing against its API.

As MANTA policy defines, the regular code reviews were done by MANTA developers to assure code quality.

5.3 Documentation

The most important part of the documentation is this work itself. It contains important information about the data model, a complete description of functionalities, the Workflow life cycle, algorithm descriptions, the architecture of the application and other diagrams such as component diagrams, activity diagrams, state diagrams.

The code, all methods, classes, enumerations, interfaces and member variables, are completely documented with Javadoc which may be found in enclosed SD card. The API is documented by annotations which generate Swagger documentation.

More complicated parts are enriched by comments to make them easier to understand for the next developers. The most complicated parts are mainly the Scenario execution and Workflow planner.

Conclusion

As part of this thesis, the MANTA ecosystem was analysed with its processes, also known as Scenarios. A REST API was designed to allow processes to be executed and monitored. Orchestration requirements were analysed, and a planning solution was proposed for a more comprehensive workflow containing a larger number of processes. A graphics interface design using wireframes has been created. A fully functioning prototype that can start, orchestrate and monitor MANTA processes has been implemented. The application has been tested and documented. All thesis assignments were fulfilled.

On 24th July 2020, the Orchestration and Monitoring module with implemented API was released together with other MANTA products in version 1.29.1, and it is available to be used by MANTA's customers and partners. The product has the benefits of creating more customised workflows with wider usage, allowing for cross-technology parallelism, which Master Scenarios did not allow, and allowing users to create automated processes that use the already mentioned REST API. The application supersedes the non-existent API of the flagship product MANTA CLI.

Once released, the Orchestration and Monitoring awaits maintenance and further development. A user interface needs to be developed and missing features implemented, especially Scenarios which need special treatment must be implemented as soon as possible. As MANTA Help Desk team gathers feedback from users; newly found bugs have to be fixed, and users request fulfilled. Whenever missing features are implemented, the planning algorithm should be extended by an online planner which will reflect Scenario Weights so the analysis can be paralleled too.

Bibliography

[1] Andrš, J. MANTA Screenshots & Screencaptures. *MANTA Confluence [online]*, June 2019, [cited 2020-5-3]. Available from: `https://mantatools.atlassian.net/wiki/spaces/SM/pages/743669977/MANTA+Screenshots+Screencaptures`

[2] Hermann, L. MANTA Flow Architecture. *MANTA Confluence [online]*, April 2020, [cited 2020-6-26]. Available from: `https://mantatools.atlassian.net/wiki/spaces/MTKB/pages/70230122/MANTA+Flow+Architecture`

[3] Triller, N. Monitoring Batch Jobs with Prometheus. *Nick's TechBlog [online]*, Dec 2018, [cited 2020-5-6]. Available from: `https://www.nicktriller.com/blog/monitoring-batch-jobs-with-prometheus/`

[4] Drobný, D. *Extracting Information from Database Modeling Tools*. Charles University, Faculty of Mathematics and Physics, 2019.

[5] Eliáš, R. *Analyzing Data Lineage in Database Frameworks*. Charles University, Faculty of Mathematics and Physics, 2019.

[6] Manta Tools s.r.o. About us MANTA [online]. ©2020, [cited 2020-5-3]. Available from: `https://getmanta.com/about-us/`

[7] Knight, M. Data Lineage Demystified: The What, Why, and How. *Dataversity [online]*, April 2017, [cited 2020-5-3]. Available from: `https://www.dataversity.net/data-lineage-demystified/`

[8] Sebastian-Coleman, L. *Measuring Data Quality for Ongoing Improvement*. Morgan Kaufmann, first edition, 2013, ISBN 9780123970336.

[9] Allen, Cervo. *Multi-Domain Master Data Management*. Morgan Kaufmann, 2015, ISBN 9780128008355.

[10] Košvanec, P. *Analýza datových toků v reportovacích nástrojích*. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

[11] Míček, D. *Analýza datových toků v Excelu*. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

[12] Kováč, J. [in person], 2020, MANTA Employee, Developer of Utils Team.

[13] Visual Paradigm. What is Use Case Diagram? [online]. ©2020, [cited 2020-5-5]. Available from: `https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/`

[14] AltexSoft. Functional and Nonfunctional Requirements: Specification and Types. *AltexSoft [online]*, May 2018, [cited 2020-5-3]. Available from: `https://www.altexsoft.com/blog/business/functional-and-non-functional-requirements-specification-and-types/`

[15] Ostic, E. [in person], 2019, MANTA Employee, Senior Vice President of Products.

[16] Hermann, L. [in person], 2019, MANTA Employee, Vice President of Development.

[17] Ulrych, J. How to Run Individual Connections within a Technology. *MANTA Confluence [online]*, April 2020, [cited 2020-5-10]. Available from: `https://mantatools.atlassian.net/wiki/spaces/MTKB/pages/811433991/How+to+Run+Individual+Connections+within+a+Technology`

[18] Ulrych, J. [in person], 2020, MANTA Employee, Vice President of Presales.

[19] Prometheus Authors. Overview - What is Prometheus? [online]. ©2020, [cited 2020-5-7]. Available from: `https://prometheus.io/docs/introduction/overview/`

[20] PushMon. Script, Job, App, Batch & Cron Job Monitoring – Push Monitoring. ©2020, [cited 2020-5-7]. Available from: `https://www.pushmon.com/`

[21] Activeeon. Activeeon Try Platform. ©2020, [cited 2020-5-7]. Available from: `https://try.activeeon.com/`

[22] The Activeeon team. ProActive Workflows & Scheduling - User Guide. ©2020, [cited 2020-5-7]. Available from: `https://try.activeeon.com/doc/user/ProActiveUserGuide.html`

[23] Oracle. Class ProcessBuilder. *Java Documentation [online]*, ©2020, [cited 2020-6-17]. Available from: https://docs.oracle.com/javase/8/docs/api/java/lang/ProcessBuilder.html

[24] Oracle. Class Process. *Java Documentation [online]*, ©2020, [cited 2020-6-17]. Available from: https://docs.oracle.com/javase/8/docs/api/java/lang/Process.html

[25] Wildcard. Unable to use an Array as environment variable. *Stack Exchange [online]*, 2018, [cited 2020-6-25]. Available from: https://unix.stackexchange.com/questions/393091/unable-to-use-an-array-as-environment-variable

[26] The Open Group. Environment Variables. *The Open Group Base Specifications Issue 7 [online]*, 2018, [cited 2020-6-25]. Available from: https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap08.html

[27] Kováč, J. *Design and prototype implementation of a logging framework for Manta Flow*. Czech Technical University in Prague, Faculty of Electrical Engineering, Department of Computer Science, first edition, 2020.

[28] Shmeltzer, S. Introduction to Liquibase and Managing Your Database Source Code. *DZone [online]*, October 2017, [cited 2020-7-19]. Available from: https://dzone.com/articles/introduction-to-liquibase-and-managing-your-databa

Appendix A

Acronyms

MANTA CLI extracts metadata, analyses scripts and sends it to MANTA Server. Optionally, it can export Dataflow Analysis to third party-tools.

MANTA Server is a Java web application running on Tomcat server which consumes the Analysis from MANTA CLI and visualises the Dataflow.

MANTA Utility Tool is a set of java web applications which helps to Update or Configure MANTA Software. This set runs on different Tomcat server.

MANTA Admin UI is a new work-in-progress name for MANTA Utility Tool containing Updater, Configurator, Log Viewer and Orchestration & Monitoring tool.

MANTA Log Viewer is a new bachelor thesis from Jakub Kováč (CTU FEL) which is being created simultaneously with this thesis. This product should gather logs from whole MANTA Ecosystem.

MANTA Updater is my bachelor thesis which helps a customer to merge changes in XML and properties files keeping their changes.

MANTA Configurator is a tool which helps a customer to configure MANTA to their environment.

MANTA Orchestration & Monitoring (sometimes referenced as the application or Orchestration & Monitoring Tool) is a product of this thesis.

MANTA Installer is a tool which installs MANTA to customer's device and helps to update MANTA Updater and MANTA Configurator.

Dataflow Analysis is data from MANTA CLI, which can be visualised as data processing, its movement and its transformation in MANTA Server or third party software. It also can be referenced as a Revision, and Revision can be distinguished to Major Revision and Minor Revision.

Workflow (Scope AS-IS MANTA CLI) is a MANTA CLI process composited from three steps in this order: Extract, Analyse, Export. It is also known as the Full Run. Technically it is a set of batch or shell scripts which define the execution order of Master Scenarios. Newly, when batch and shell is used no more, it is a payload definition for Orchestration & Monitoring tool.

Master Scenario is a Java bean which executes one or more Scenarios. Scenarios have a predefined order, and some of them can run parallelly in more threads.

Scenario (Scope AS-IS MANTA CLI) is a Java bean responsible for a step of MANTA Extract, Analyse and Export process.

Platform is a Java application with instantiates and executes a Master Scenario or Scenario. It is possible to run more Master Scenarios parallelly, that means there could be more Platform processes.

Connection ID (Scope AS-IS MANTA CLI) is a pair of Connection Name and Technology Name. This pair identifies a user-configured Connection to one of the user's systems in MANTA Configurator.

Workflow Scripts are actual workflows configured in .sh and .bat scripts in MANTA CLI.

Full Run is a Workflow containing all three steps Extract, Analyse, Export for some technologies. It creates a new revision of the Dataflow Analysis. It does not mean that all available technologies were analysed.

Incremental Run can add Dataflow Analysis of previously skipped technologies. Alternatively, it can update the Analysis for some Scenarios in specific cases.

Extract is a retrieving of the metadata from Databases and supported technologies. The corresponding Master Scenario is named ⟨technology⟩ExtractorMasterScenario. In this step, MANTA CLI communicates with the outer world.

Analyse is a workflow step which processes source files, links them with extracted metadata, and sends them to MANTA Server. The corresponding Master Scenario is named ⟨technology⟩DataflowMasterScenario. The first step of Analysis is Open Revision, and the last is Commit. In this step, MANTA CLI does not talk with the outer world.

Export is a workflow step which takes the Dataflow Analysis from MANTA Server and sends it to third party software. The corresponding Master Scenario is named ⟨technology⟩ExportRepositoryScenario.

Repository is a set of files and graph database containing commits of Analysis for Dataflow. MANTA Server uses it.

Major Revision contains a Dataflow Analysis from Full Run. Technologies which were not analysed cannot be seen here.

Minor Revision inherits Dataflow Analysis from Major Revision and rewrites or adds new technologies analysis.

Open Revision is the first step of Analysis. It prepares the MANTA Server for Analysis arrival.

Commit is a name of the last step of sending the Analysis to MANTA Server. It determines that all data were sent. Commit must be preceded by Open Revision step.

Workflow (Scope Orchestration & Monitoring) is a list of Phases, Technologies, Connections and Scenarios. It defines which Scenarios should be executed for which Phase, Technology and/or Connection, regardless of their Orchestration Constraints. The Workflow also contains metadata about itself, e.g. creation time and the creator or configurable environment variables.

Scenario (Scope Orchestration & Monitoring) is an execution unit for a Workflow. The Scenario may have Orchestration Constraints.

Job is a node in an execution plan. (Execution Job, Parallel Job, Barrier Job)

Phase is a step grouping list of Scenarios. Its value may be one of Extract, Analyse, Export.

Technology ID is the name of supported technology by MANTA. Together with Connection ID identifies a particular Connection for MANTA Configurator.

Connection ID (Scope Orchestration & Monitoring) is the name of a user-created Connection. This name is unique only for a particular technology; thus, two technologies can contain two different Connection with the same name. The system-wide unique identifier is a pair of Technology ID and Connection ID.

Scheduling is a process of finding the best execution order respecting runtime constraints.

Orchestration Constraints is a set of rules. It defines which Scenarios cannot run simultaneously, which has to be executed first or which one has to wait for the other one. These constraints are defined in the Scenario Metadata layer.

Workflow ID is a name and identifier of an existing Workflow. It is a String, and it must be unique. Can also be referenced as a Workflow name.

Workflow Execution ID is the identifier of a particular Workflow Execution. It is a Long, generated in the database. One Workflow ID may have multiple Workflow Execution IDs. Workflow Execution ID is linked with several Scenario Execution IDs.

Scenario ID is a name of a Scenario. It is a combination of Technology name and Operation name. It is used as an ID for a Scenario in a Workflow and as an ID for a Scenario in ScenarioMetadata. It is a unique String. Can also be referenced as a Scenario name.

Scenario Execution ID is an ID of execution of a particular Scenario. One Scenario ID may have multiple Scenario Execution IDs.

Scenario Metadata is information about a particular Scenario and is identified by Scenario ID. These Scenarios are hidden from users and stored in the database.

Customer, Client is a purchaser of MANTA software.

User is a person who uses the MANTA software.

Ordering Party is a group of MANTA employees represented by Jan Ulrych, Ernie Ostic.

Technical Supervisor is a group of MANTA employees represented by Lukáš Hermann and Jakub Moravec.

Appendix B

Contents of enclosed SD card

```
outputs/ ..... the directory with application output examples
readme.txt ..... the file with SD card contents description
src/ ..... the directory of source codes
    app/ ..... implementation sources
    diagrams/ ..... diagram sources used in the thesis
    thesis/ ..... the directory of LaTeX source codes of the thesis
text/ ..... the thesis text directory
    MT_Gondek_Petr_2020.pdf ..... the thesis text in PDF format
```

Appendix C

Attachments

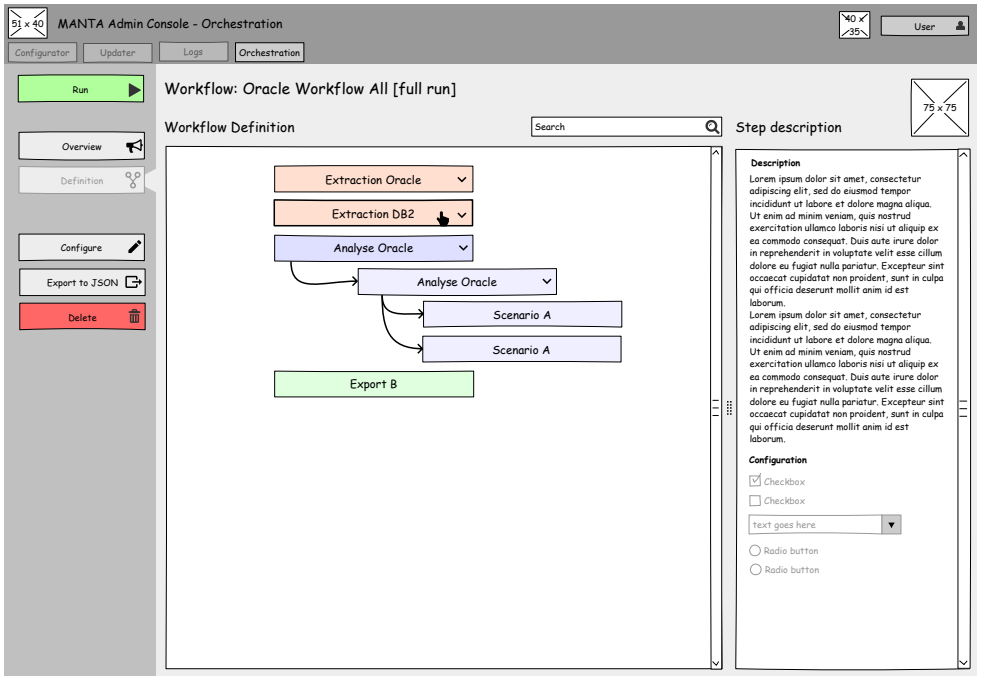

Figure C.1: Workflow Detail Screen - Definition

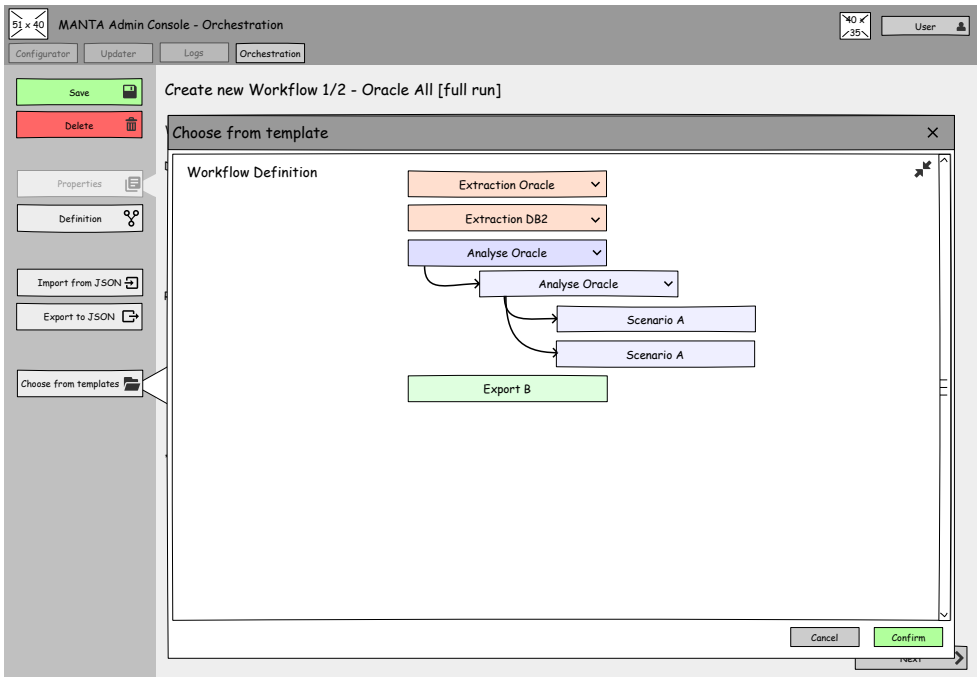

Figure C.2: Template Modal Window - Bigger preview for Workflow Definition

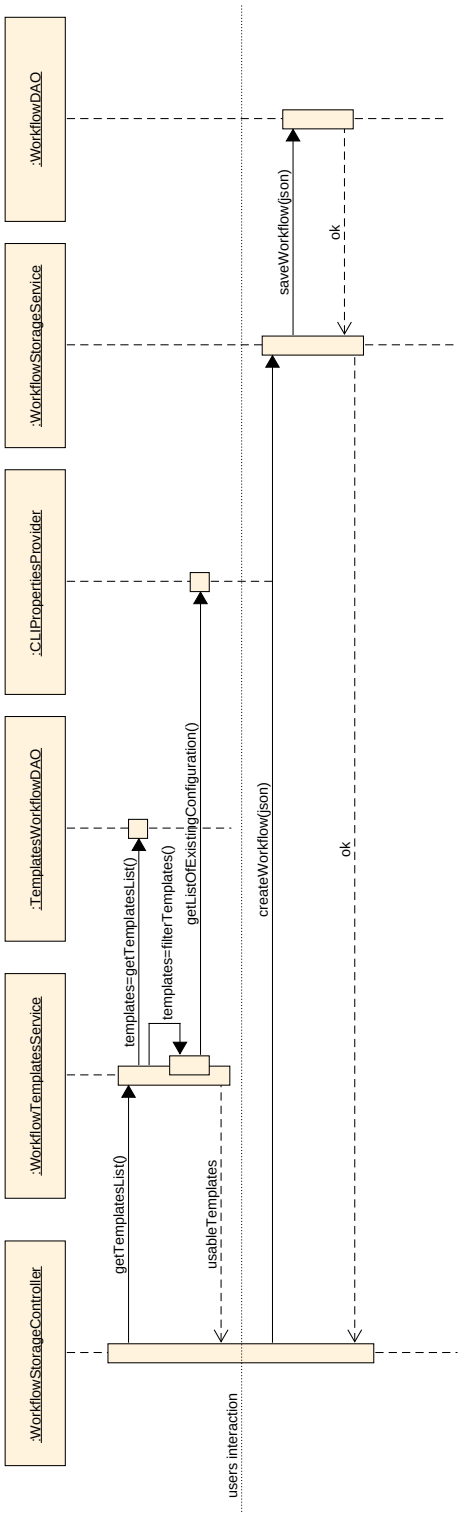

Figure C.3: Workflow Creation Sequence Diagram