Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

360°- My personal Health and Fitness Monitor

Documentation

10.02.2016

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

Table of Contents

Apart from the contents listed in above table,

a detailed code documentation is also available as .html file.

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

USE CASES

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

PRIMARY USE CASE

Analyze Datasets

SCOPE:	360° - Health and fitness application
LEVEL:	User goal, primary
ACTORS:	Athlete
STAKEHOLDERS AND INTERESTS:	Athlete: Wants a broad overview of training data, as well as detailed comparisons of related data records. By identifying correlations between different data types, the athlete wants to control training progress and draw conclusions for further training steps.
PRECONDITIONS:	• The athlete is logged in the system. • The athlete has tracked at least two different types of fitness or health data. • All needed datasets are already synchronized and accessible by the system.
DESCRIPTION:	The athlete opens the analysis page, selects an available data record and analyses special anomalies and dependencies between different data types. Like this, the athlete is able to evaluate training progress and draw conclusions for further activities.
POST CONDITIONS:	• The datasets are not changed and contain the same values as before. • Special anomalies and strong dependencies between two data types are correctly calculated and detected. • Athlete gets an impression of own training performances from the data record visualizations.
TRIGGER:	The athlete navigates to the data analysis page to evaluate the training progress.

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

MAIN SUCCESS SCENARIO:	1. Athlete navigates to data analysis page. 2. System provides an overview of available data records. 3. Athlete selects a data record for detailed information. 4. System presents the selected data record as distributed values over time. 5. Athlete chooses to highlight special anomalies like for example maxima, minima or other outliers. 6. System detects and emphasizes special anomalies. 7. Athlete selects to include other data types in order to evaluate correlations. 8. System provides a list of available data types, that are suitable for inclusion. 9. Athlete chooses to highlight strong dependencies between selected data types. 10. System computes and visualizes strong dependencies. 11. Athlete comes up with a conclusion for future training and leaves analysis page.
ALTERNATIVES:	*a. At any time, System fails: 1. During the reboot process, the system recovers the page presented right before the crash. 2. Athlete continues at the point the system crashed. 1a. The page recovery fails. 1. System presents a short failure indicator and shows the application's start page. 2. Athlete navigates to a different site or tries to perform the same analysis again. 2a. No data records are available to display. 1. System presents an empty dashboard and offers a suggestion to record and upload health or fitness data first. 2. Athlete performs the suggested data upload operation or leaves the application. 4a. The selected data record is temporarily unavailable.

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

	1. System shows an error message and navigates back to the data records overview. 2. Athlete tries to access the data record again or chooses another one. 6a. No special anomalies are detected. 1. System alerts, that no anomalies can be found and keeps on displaying the selected data record as values over time. 2. Athlete continues with 7. 8a. No suitable data types are available. 1. System indicates, that there are currently no appropriate data types available. 2. Athlete tries again or navigates back to the data records overview. 10a. No strong dependencies are detected. 1. System alerts, that no strong dependencies are detected for the selected data types and continues. 2. Athlete selects different data types or leaves the application
FREQUENCY OF OCCURRENCE:	High frequency, up to nearly continuous

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

SECONDARY USE CASE

Discover trends

SCOPE:	360° - Health and fitness application
LEVEL:	User goal
ACTORS:	Athlete
STAKEHOLDERS AND INTERESTS:	Athlete: Wants to track and access his health and physical performance data over a period of time. Tracking-Device-Manufacturer: Is interested in popular applications based on his data as their popularity is likely to promote his products as well.
PRECONDITIONS:	Authentication was applied and the user is allowed to use the intended profile in the application. Data to be analyzed and to be shown is read in.
DESCRIPTION:	The user is able to draw conclusions about his performance using the provided data analyzation mechanisms. Focusing on a certain attribute over a longer period of time enables him to adapt his behavior and define new performance goals.
POST CONDITIONS:	The user experienced an easy to use and all the same informative presentation of his performance data which enabled him to draw conclusions about his physical activity and health.
MAIN SUCCESS SCENARIO:	1. The user enters the trend analyzation mode 2. The user creates a new trend analyzation instance 3. The system provided possible datasets, that are available for a user defined timespan. In addition, experienced users are offered the opportunity to select an attribute and trend length themselves. 4. The system shows the result of the performed trend analyzation 5. The user can repeat steps 3 and 4 to vary the time period

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

	6. The user selects the provided automatization-mode, that is possible for certain attributes. The system presents an automatically created evaluation of his tracked performance and health. 7. Experienced users are invited to prompt metrics they want the analysis to focus on. Advancing, the system applies these metrics to the analyzation. 8. The user enters comparison-mode. He selects a previous trend analyzation and the system compares that with the new analyzation. 9. The user can define new goals and their deadlines. 10. The user saves the trend analyzation for future usage.
ALTERNATIVES:	1-2. The user selects a previous trend analyzation instead of creating a new one. 2-2. At all time, during selection of modes or datasets: 2-1. The system informs the user about a problem concerning that dataset and provides detailed information about the criticality and handling of the error. 2-2. The user aborts the process and can not perform analyzations. 2-3. The user is offered the possibility to report an internal application error to the product support.
SPECIAL REQUIREMENTS:	1. Easy to use frontend 2. The system is required to perform the analyzation algorithms within 5 seconds.
TECHNOLOGY AND DATA VARIATIONS LIST:	1. The system shall be compatible with most popular devices in 2015 (e.g. AppleWatch, Garmin, Android Wear etc.) 2. The application shall be accessible from a browser.
FREQUENCY OF OCCURRENCE:	Medium frequency

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

OPEN ISSUES:	1. Complexity of handling different devices and therefore varying data formats. 2. Quality of the analyzation and dimension of technical aspects in the analyzation

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

SECONDARY USE CASE

Read data from device

SCOPE:	360° - Health and fitness application
LEVEL:	User goal, secondary
ACTORS:	General user
STAKEHOLDERS AND INTERESTS:	User: wants to import data from his tracking device
PRECONDITIONS:	• Datasets are stored on the device • Authentication was done and user is granted access to the application.
POST CONDITIONS:	• Datasets from the devide are now stored in the application
TRIGGER:	User wants to add new data to the application in order to analyze them
MAIN SUCCESS SCENARIO:	1. User connects device with the system- 2. User selects which datasets to import. Optionally, he can add metainformation like descriptions, consumed food etc. 3. User submits his input 4. System saves the data and shows a success message.
ALTERNATIVES:	*a. At any time, System fails: 1. During the reboot process, the system recovers the page presented right before the crash. 2. Athlete continues at the point the system crashed.

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

	1a. No connection between device and system could be established. The process is aborted and the user is informed about the error and possible solutions. 4a. The system is not able to read or store the datasets. The process is aborted and the user is informed about the error and possible solutions.
SPECIAL REQUIREMENTS:	- Data formats of most popular most popular devices in 2015 (e.g. AppleWatch, Garmin, Android Wear etc.) shall be supported. - The system shall be compatible with most popular devices in 2015 (e.g. AppleWatch, Garmin, Android Wear etc.)
FREQUENCY OF OCCURRENCE:	Medium frequency
OPEN ISSUES:	- Connection of Device and System - Which data and data formats shall be supported? - Which format is used for internal storage of the data? - What additional metainformation shall be stored?

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

SECONDARY USE CASE

Insert non-device data

SCOPE:	360° - Health and fitness application
LEVEL:	User goal, secondary
ACTORS:	General User
STAKEHOLDERS AND INTERESTS:	User: wants to store manually created datasets
PRECONDITIONS:	• Authentication was done and user is granted access to the application.
POST CONDITIONS:	• Data was stored successfully in the device
TRIGGER:	User enters mode to store manually created data
MAIN SUCCESS SCENARIO:	1. User inserts information into a form provide by the system 2. User classifies the data using classifications provided by the system. 3. System stores the data and informs the user about the successful storage
ALTERNATIVES:	*a. At any time, System fails: The process is aborted and no data is added.
FREQUENCY OF OCCURRENCE:	Low frequency

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

OPEN ISSUES:	Define which datasets are relevant (e.g. consumed food or protein etc.)

SECONDARY USE CASE

Use goals to manage work life balance

SCOPE:	360° - Health and fitness application
LEVEL:	User goal
ACTORS:	Permanent carrier of a tracking device
STAKEHOLDERS AND INTERESTS:	Permanent carrier of a tracking device
PRECONDITIONS:	Data from the tracking device are already read into the system.
POST CONDITIONS:	• A Visualization was presented to the user. • A comparison of goals and actually achieved values was done. • New goals were defined. • The user configured the system to remind him of the defined goals or not.
TRIGGER:	User select everyday-mode
MAIN SUCCESS SCENARIO:	1. The user selects a timespan and the system presents available datasets. 2. Having selected a dataset, the user selects additional dataset to be presented. If aggregation

	methods are available for the selected datasets, the user can decide whether to have a combined presentations or separate visualizations. 3. Previously defined goals are loaded automatically and their degree of fulfillment is shown together with the dataset. 4. The user is offered the possibility to except certain periods of time from analysis. This forces an adaption of the prior visualization. 5. The user can define new goals for the single measurements. He can define whether the defined value shall be treated as the lower bound of the tracked data or as an upper threshold. Based on previously defined goals, the system provides supports and makes proposals. Eventually, the goals are stored in the system. 6. The user can configure whether he wants the system to remind him of the goals and whether he wants positive feedback after the fulfillment. 7. The user closes the application.
ALTERNATIVES:	- **1a** If there are no datasets available, the system informs the user and falls back to its previous state. - **3a** If there are no previously defined goals stored in the system, the visualization is not changed. - **5a.** Reminders and positive feedback can be implemented by the devide, given a suitable interface is provided by the device. - **5b**. Using reminders, the user is reminded of the unfulfilled goals at every start of the application.
SPECIAL REQUIREMENTS:	- The tracking device is required to provide suitable datasets - Inconsistent datasets where detected and are not shown to the user.

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

TECHNOLOGY AND DATA VARIATIONS LIST:	- Desktop-Application using datasets that are read into the system from beyond the systems scope
FREQUENCY OF OCCURRENCE:	High frequency

OPTIONAL USE CASE

Compare own results with other users

SCOPE:	360° - Health and fitness application
LEVEL:	User goal, secondary
ACTORS:	general User
STAKEHOLDERS AND INTERESTS:	User: Wants to compare his tracked performance and health values with values from other users, in order to gain a better understanding of his performance.
PRECONDITIONS:	• Authentication was done and user is granted access to the application.
POST CONDITIONS:	• The user was shown an comparison of two datasets, one from him and one from a seconds person.
TRIGGER:	User enters comparison-mode
MAIN SUCCESS SCENARIO:	1. User selects a person he wants to be compared to.

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

	2. User selects a dataset from his own values and gets comparison with corresponding dataset of selected person shown.
ALTERNATIVES:	*a. At any time, System fails: 1. During the reboot process, the system recovers the page presented right before the crash. 2. User continues at the point the system crashed. 1a. The select comparison person did not agree to share any data. The process is aborted and the user is informed about the error. 2a. The select comparison person agreed to share his data but excluded specific datasets. The process is aborted and the user is informed about the error. 2b. No corresponding dataset could be found. The process is aborted and the user is informed about the error.
SPECIAL REQUIREMENTS:	- Follow legal privacy protection policy
FREQUENCY OF OCCURRENCE:	Medium frequency
OPEN ISSUES:	- Agreement to share data or exclude datasets from comparison

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

OPTIONAL USE CASE

Search similar persons

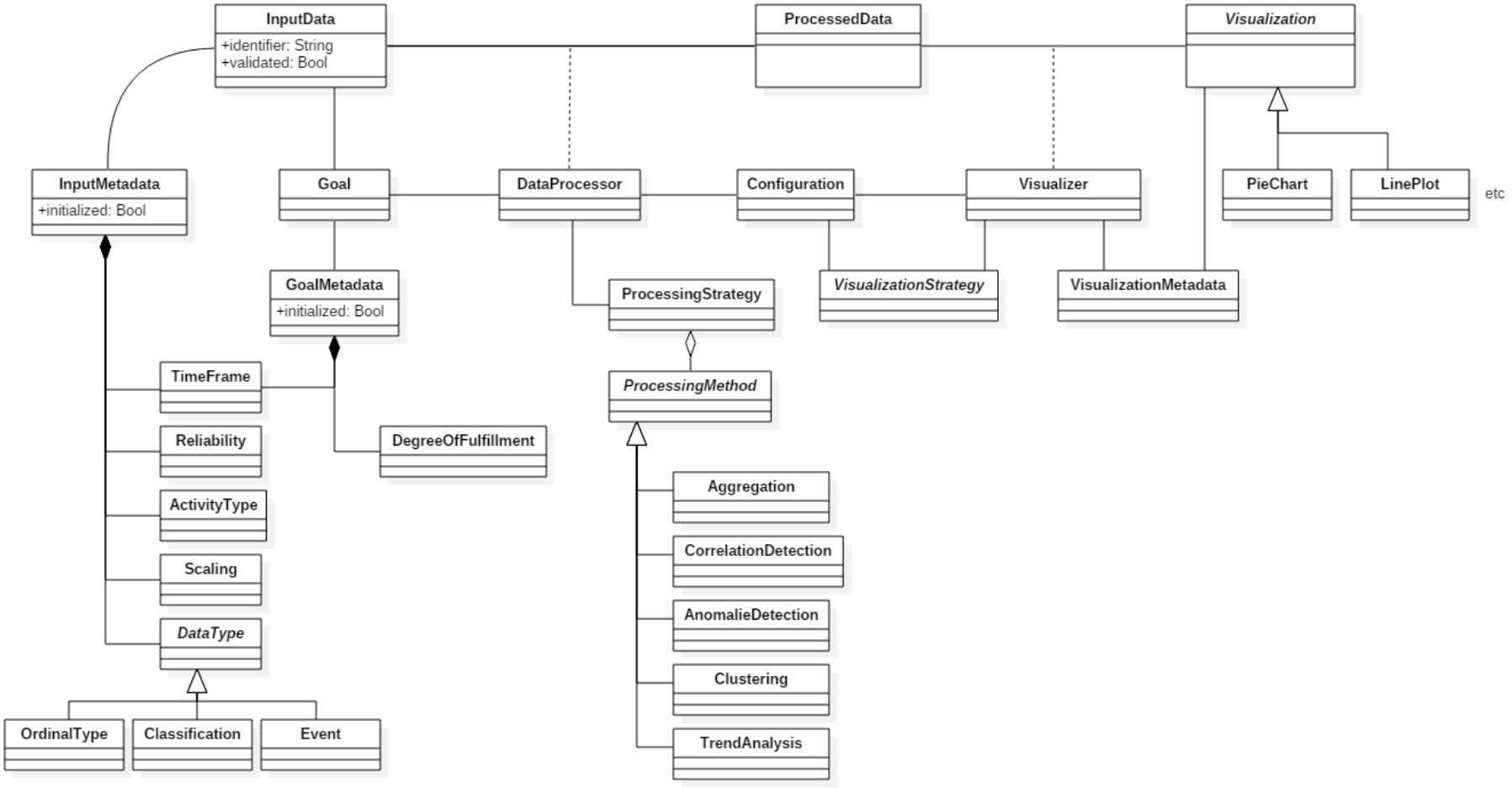
SCOPE:	360° - Health and fitness application
LEVEL:	User goal, secondary
ACTORS:	Athlete
STAKEHOLDERS AND INTERESTS:	The athlete user wants to find other athletes with similar performance
PRECONDITIONS:	• Authentication was done and user is granted access to the application. • The datasets corresponding to the user are available in the system
POST CONDITIONS:	• The user was shown a list of other users with similar performance values within his periphery.
TRIGGER:	User enters FindPartner-mode
MAIN SUCCESS SCENARIO:	1. The user defines the kind of training he is most interested in and defines requirements that a potential partner has to meet as well as the maximum distance between both. 2. The user can add additional, optional requirements like the gender of his partner. 3. The user is shown a list of persons that meet his requirements
ALTERNATIVES:	*a. At any time, System fails: 1. During the reboot process, the system recovers the page presented right before the crash.

	2. Athlete continues at the point the system crashed. *b. No suitable user was found. The athlete is informed about the result and is invited to repeat step 1 with different parameters.
SPECIAL REQUIREMENTS:	- Follow legal privacy protection policy - The result must not contain persons who did not agree to usage of their data for this purpose.
FREQUENCY OF OCCURRENCE:	Low frequency
OPEN ISSUES:	- Which requirements shall be supported? - How to decide if requirements are matching?

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

CONCEPTUAL MODEL

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle



Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

InputData	Represents all raw data as stored in some form of persistence, e.g. a database)
InputMetadata	Metadata describing InputData with all kinds of parameters.
Goal	User-added goal for one or multiple datasets.
GoalMetadata	Metadata describing the goal, e.g. desired timeframe to achieve it.
Configuration	User submitted configuration on how to process the data and how to visualize it.
DataProcessor	Association class managing the processing strategy, the config and how data is processed.
ProcessingStrategy	Composition of multiple steps in the processing pipeline
ProcessingMethod	Individual methods of data processing.
ProcessedData	Result of processing methods.
Visualizer	Association class managing the configuration and the creation of the visualization.
VisualizationStrategy	Way of creating a visualization from input ProcessedData
Visualization	Final result of the pipeline.
VisualizationMetadata	Metadata describing the visualizations, their requirements, preferences, parameters etc.

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

CONTRACTS

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

Contracts

INPUTDATA MANAGER

getData

OPERATION	1 `getData : (DataID) InputData` 2 `getData :: DataID -> InputData` 3 $getData : DataID \rightarrow InputData \quad mit\ DataID \neq NULL$
TYPE	System Operation
DESCRIPTION	Searches for data : InputData with data.Id = inputId : DataId and returnes it.
PRE-CONDITION	A data : InputData object with data.Id = inputId : DataId exists.
POST-CONDITION	data : InputData was found and returned.
RESULT	Data : InputData with the given InputId : DataId
EXCEPTIONS	Data : InputData is not validated. Call of validateData before return of data : InputData
OUTPUT	[data.validateData] Data : InputData
REFERENCES	Conceptual model (Association class**:** Data Processor), Processing Methods / Strategies Other Contracts: validateData.
COMMENTS	The Datatype DataId is not defined in the conceptual modell. It has to be replaced as soon as the decision was made, what datatype will be used for ids.

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

getMetadata

OPERATION	1 getMetadata : InputMetaData 2 getMetadata :: InputMetaData 3 *getMetadata* :→ InputMetaData
TYPE	System Operation
DESCRIPTION	Returns md : InputMetadata of a data : InputData
PRE-CONDITION	Data : InputData is validated Md : InputMetadata is connected to data : InputData, from which it is called
POST-CONDITION	Md : InputMetadata was returned.
RESULT	Md : InputMetadata of data : Inputdata
EXCEPTIONS	-
OUTPUT	md: InputMetaData
REFERENCES	Conceptual model of Get-Method. Header-Data: MetaData containing structural information of the data
COMMENTS	Getter-Operation called from data : InputData

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

validateData

OPERATION	`validateData : (InputData) InputData`
	`validateData :: InputData -> InputData`
	`validateData : InputData → InputData`

TYPE	System Operation
DESCRIPTION	Unvalidated uid : InputData is validated and validated data : inputData is returned.
PRE-CONDITION	Uid : InputData is unvalidated and not null.
POST-CONDITION	Data : InputData was created and set as validated. Data : InputData contains all data from uid : InputData plus interpolation results.
RESULT	The data of uid : InputData was validated.
EXCEPTIONS	1. Uid : InputData has missing information. Interpolation is applied to generate missing data. 2. Uid : InputData has datatype that cannot be read. The user is informed by an error message and no validated data is returned
OUTPUT	Data : InputData
REFERENCES	Conceptual model
COMMENTS	No explicit interpolation algorithms mentioned as they depend on the input data type.

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

INPUTMETADATA

getMetaData

OPERATION	`getMetaData : (DataID) InputMetaData`
	`getMetaData :: DataID ->  InputMetaData`
	$getMetaData : DataID \rightarrow InputMetaData \quad mit\ DataID \neq NULL$

TYPE	System Operation
DESCRIPTION	Searches for data : InputData with data.id = InputId : DataId, returns imd : InputMetaData connected with data : InputData
PRE-CONDITION	inputId : DataId is not null and data : InputData with data.id = InputId : DataId exists.
POST-CONDITION	Data : InputData with InputId : DataId was found. Data : InputData was validated Imd : InputMetadata was initialized Imd : InputMetadata was returned
RESULT	Imd : InputMetadata connected to data : InputData, which was specified by given inputId : DataId is returned.
EXCEPTIONS	1. Data : InputData is not validated. Data.validateData is called to validate data: InputData 2. Imd: InputMetadata is not initialized. imd.Deduce is called to deduce Metadata and initialize imd : InputMetadata
OUTPUT	[data.validateData] [imd.deduce] Imd : InputMetadata
REFERENCES	Concetual Model System operations: validateData, deduce
COMMENTS	Getter-Methode

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

> The Datatype DataId is not defined in the conceptual modell. It has to be replaced as soon as the decision was made, what datatype will be used for ids.

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

deduce

OPERATION	`deduce : (InputData) InputMetadata`
	`deduce :: InputData -> InputMetaData`
	$deduce : InputData \rightarrow InputMetadata$

TYPE	System Operation
DESCRIPTION	Deduces md : InputMetadata of given data : Input data.
PRE-CONDITION	Data : InputData is validated. Md : InputMetadata is not initialized.
POST-CONDITION	Md : InputMetadata is initialized. Tf : Timeframe is set in md : InputMetadata Rel : Reliability is set in md : InputMetadata At : ActivityType is set in md : InputMetadata Sc : Scaling is set in md : InputMetadata Dt : Datatype is set in md : InputMetadata
RESULT	Md : InputMetadata was initialized with values for each composition.
EXCEPTIONS	No md : InputMetadata exists for data : InputData. Md : InputMetadada is created, than deduced.
OUTPUT	md : InputMetadata
REFERENCES	Concetual Model
COMMENTS	

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

VISUALIZER

visualizeProcessedData

OPERATION	1 `visualizeProcessedData : (ProcessedData, Configuration) Visualization` 2 `visualizeProcessedData:: ProcessedData -> Configuration -> Visualization` 3 $visualizeProcessedData : ProcessedData \times Configuration \rightarrow Visualization$
TYPE	System operation
DESCRIPTION	Creates a vis : Visualization of pd : ProcessedData according to data from given cfg : configuration.
PRE-CONDITION	Pd : ProcessedData and cfg : configuration are not null.
POST-CONDITION	Cfg_vs : VisualizationStrategy was read form cfg: Configuration. Cfg_vs : VisualizationStrategy was applied on pd : ProcessedData creating a vis : Visualization. Vis : Visualization was returned
RESULT	Pd: ProcessedData has been visualized by application of cfg_vs : VisualizationStrategy.
EXCEPTIONS	1. Cfg : configuration does not define a cfg_vs : VisualizationStrategy. default_vs : VisualizationStrategy based on md : InputMetaData is created and used instead of cfg_vs : VisualizationStrategy. 2. Cfs_vs: VisaulizationStrategy cannot be applied to pd : ProcessedData. This is handled like no cfg_vs: VisualizationStrategy was defined by cfg: Configuration
OUTPUT	[getMetadata] Vis: Visualization
REFERENCES	Conceptual model (Association class: Visualization)

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

DATAPROCESSOR

processDatasets

OPERATION	`processDatasets : (List[InputData], Configuration) ProcessedData`
	`processDatasets :: [InputData] -> Configuration -> ProcessedData`
	$processDatasets \quad InputData^n \times Configuration \rightarrow ProcessedData \quad mit\ n \in \mathbb{N} \setminus \{0\}$

TYPE	System operation
DESCRIPTION	Creates pd : ProcessedDate from of idata : ListofInputData using pm : ProcessingMethod defined in cfg : Configuration
PRE-CONDITION	All data_i : InputData in idata : ListofInputData are validated Cfg : Configuration , idata : ListofInputData not null
POST-CONDITION	Pd : ProcessedData was created. Pm : ProcessingMethod was used to create pd : ProcessedData Pd:ProcessingData was returned.
RESULT	Pm : ProcessingMethod was applied to idata : ListofInputData and pd : ProcessedData was created.
EXCEPTIONS	1. Cfg: Configuration does not contain pm: ProcessingStrategy. a. Default_pm : ProcessingStrategy is created based on md_i : InputMetadata linked to data_i : InputData from list idata : ListofInputData. Default_pm: ProcessingStrategy is used instead of pm : ProcessingStrategy. 2. Pm : ProcessingMethod cannot be applied to data_i : Inputdate from idata : ListofInputData. This is handled as exception like no pm : ProcessingMethod was definded in cfg : Configuration 3. Idata : ListofInputData contains only one data_i : InputData a. If pm : ProcessingStrategy can handle single set data, it is still applied

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

	b. Else data_i : InputData is transformed to pd : ProcessData and returned.
OUTPUT	Pd : ProcessedData
REFERENCES	Conceptual model (Association class: Data Processor), Processing Methods / Strategies
COMMENTS	Listof… is not defined in the conceptual model, because it is a basic data structure in most languages

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

GOAL

create

OPERATION	`create : (TimeFrame, ActivityType, Scaling, DataType ) Goal`
	`create :: TimeFrame -> ActivityType -> Scaling -> DataType -> Goal`
	$create : Timeframe \times ActivityType \times Scaling \times Datatype \rightarrow Goal \rightarrow \text{mit n} \in \mathbb{N} \setminus \{0\}$

TYPE	System operation
DESCRIPTION	Creates g : Goal and and sets values tf : Timeframe, at : ActivityType, sc : Scaling, dt : DataType of gmd : GoalMetadata
PRE-CONDITION	No goal with identical values of In_tf : Timeframe, In_at : ActivityType, In_sc : Scaling, In_dt : DataType in the connected gmd' : GoalMetadata must exist. Input values are not null.
POST-CONDITION	G : Goal was created. Gmd: GoalMetadata was created and connected to g : Goal. tf : Timeframe of gmd : GoalMetadata was set to In_tf : Timeframe at : ActivityType of gmd : GoalMetadata was set to In_at : ActivityType sc : Scaling of gmd : GoalMetadata was set to In_sc : ActivityType dt : DataType of gmd : GoalMetadata was set to In_dt : DataType
RESULT	G: Goal has been created and values in gmd: GoalMetadata have been set
EXCEPTIONS	
OUTPUT	G : Goal

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

getActivityType

OPERATION	1 `getActivitType : ActivityType` 2 `getActivitType :: ActivityType` 3 $getActivityType: \rightarrow ActivityType$
TYPE	System operation
DESCRIPTION	Returns at : ActivityType of gmd: GoalMetadata
PRE-CONDITION	Gmd : GoalMetadata is connected to g : Goal
POST-CONDITION	-
OUTPUT	At: ActivityType

getMetadata

OPERATION	1 `getMetadata : (Goal) GoalMetadata` 2 `getDatasets :: Goal -> GoalMetadata` 3 $getDatasets: Goal \rightarrow GoalMetadata$
TYPE	System operation
DESCRIPTION	Returns gmd: GoalMetadata of g : Goal
PRE-CONDITION	Gmd : GoalMetadata is not null
POST-CONDITION	-
OUTPUT	Gmd : GoalMetadata

getTimeframe

OPERATION	1 `getTimeframe : (Goal) Timeframe` 2 `getTimeframe :: Goal -> Timeframe` 3 $getTimeframe: Goal \rightarrow Timeframe$
TYPE	System operation
DESCRIPTION	Returns tf : TimeFrame of gmd : GoalMetadata
PRE-CONDITION	Gmd : GoalMetadata is connected to g : Goal
POST-CONDITION	
OUTPUT	Tf : Timeframe

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

getScaling

OPERATION	`getScaling : (Goal) Scaling`
	`getScaling :: Goal -> Scaling`
	$getScaling: Goal \rightarrow Scaling$

TYPE	System operation
DESCRIPTION	Returns sc : Scaling of gmd : GoalMetadata
PRE-CONDITION	Gmd : GoalMetadata is connected to g : Goal
POST-CONDITION	
OUTPUT	Sc : Scaling

getDataType

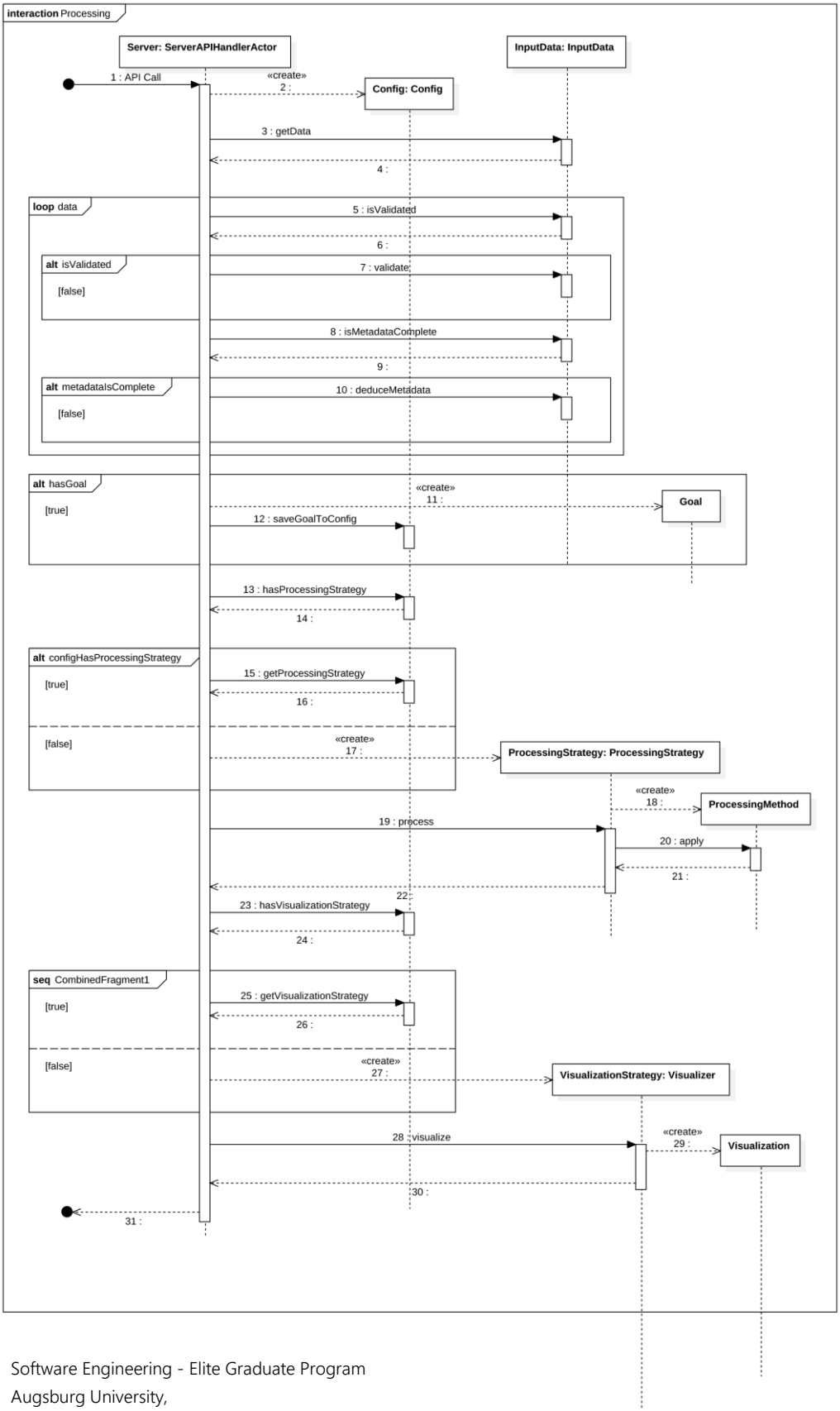
OPERATION	`getDataType : (Goal) DataType`
	`getDataType:: Goal -> DataType`
	$getDataType: Goal \rightarrow DataType$

TYPE	System operation
DESCRIPTION	Returns dt : DataType of gmd : GoalMetadata
PRE-CONDITION	Gmd : GoalMetadata is connected to g : Goal
POST-CONDITION	
OUTPUT	Dt : DataType

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

SEQUENCE DIAGRAMS

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle



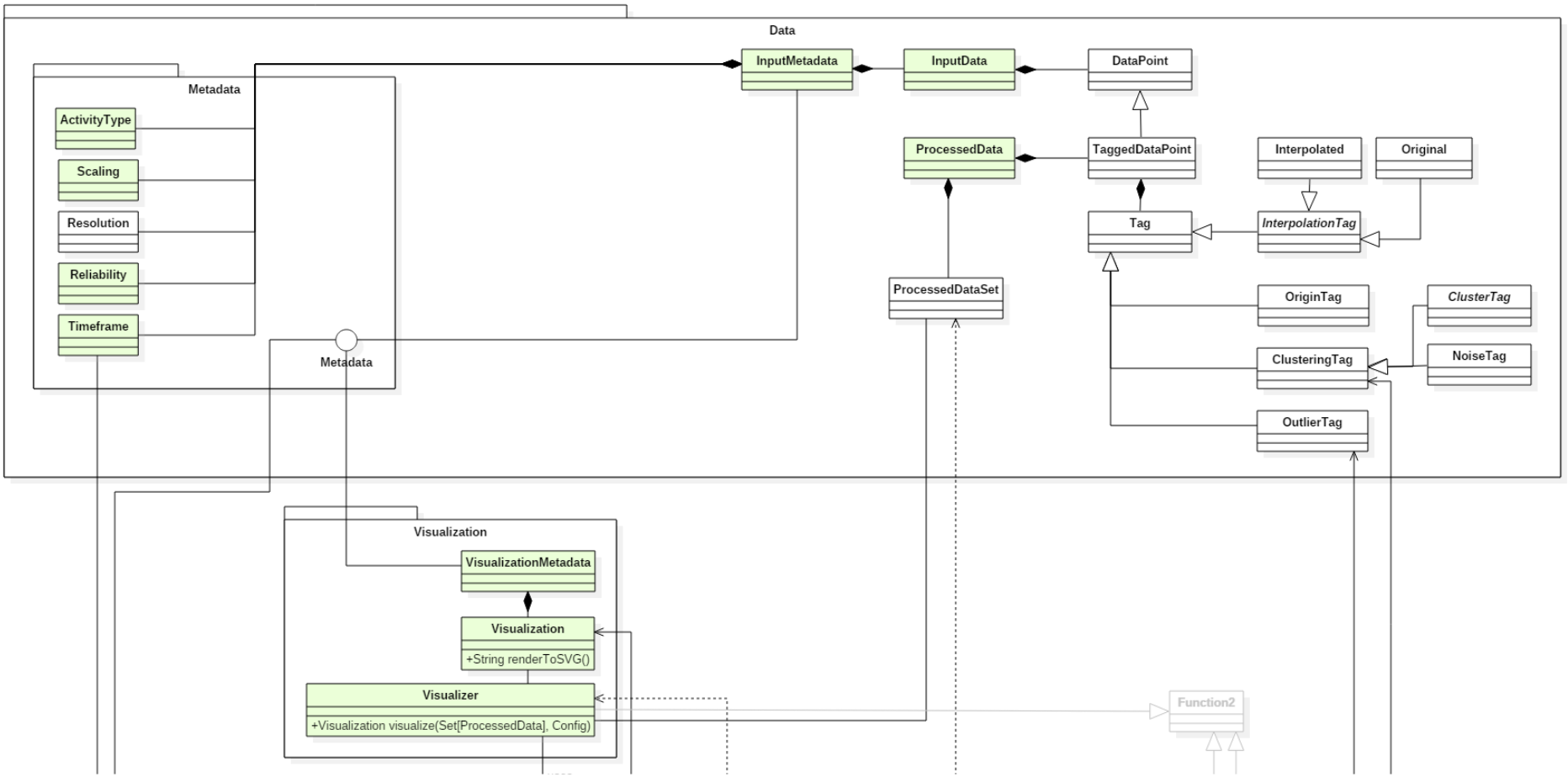
Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

DESIGN MODEL

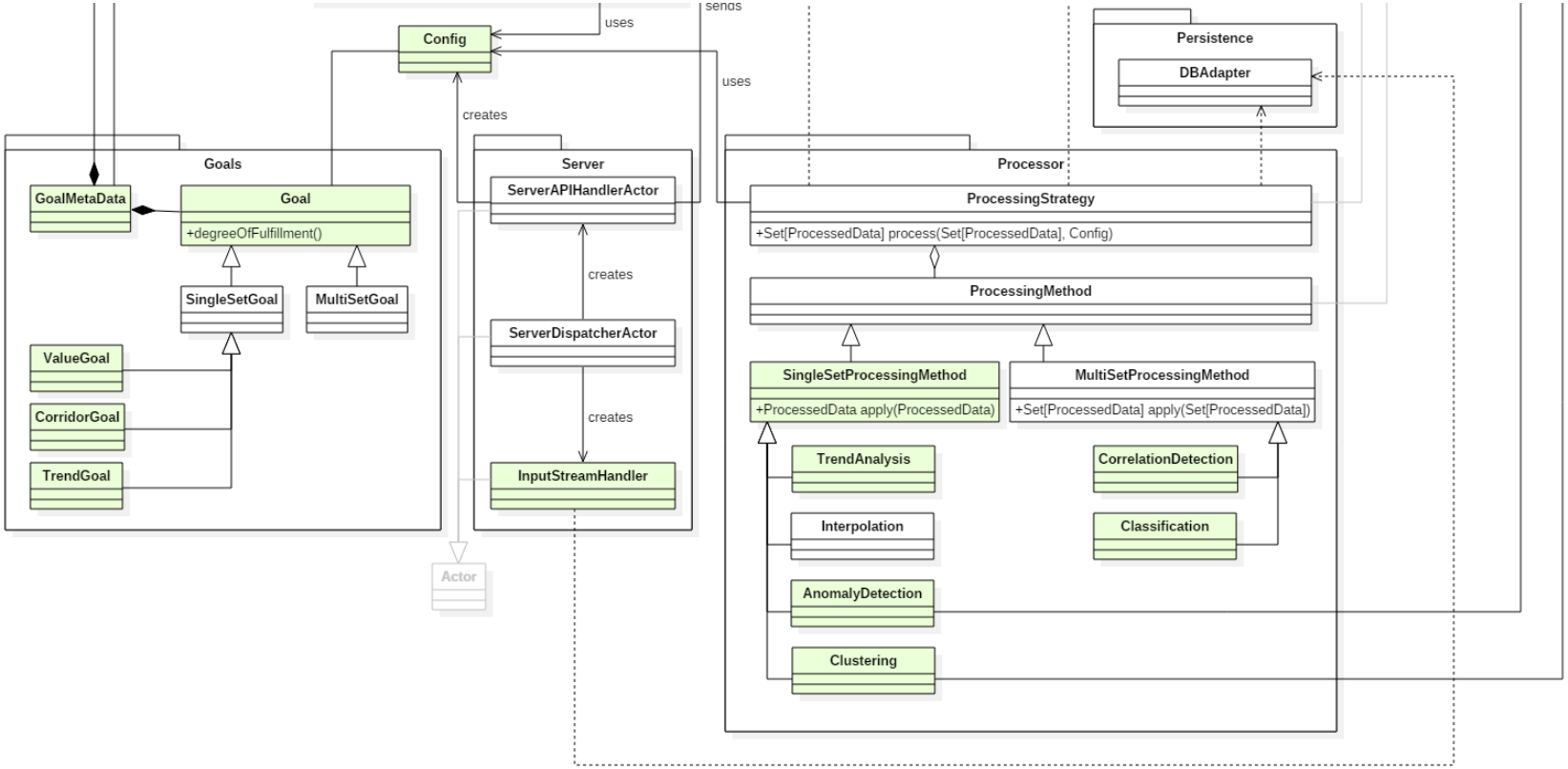
360° - Visualization Engine

Project for the Software Engineering Lecture

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle



Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle



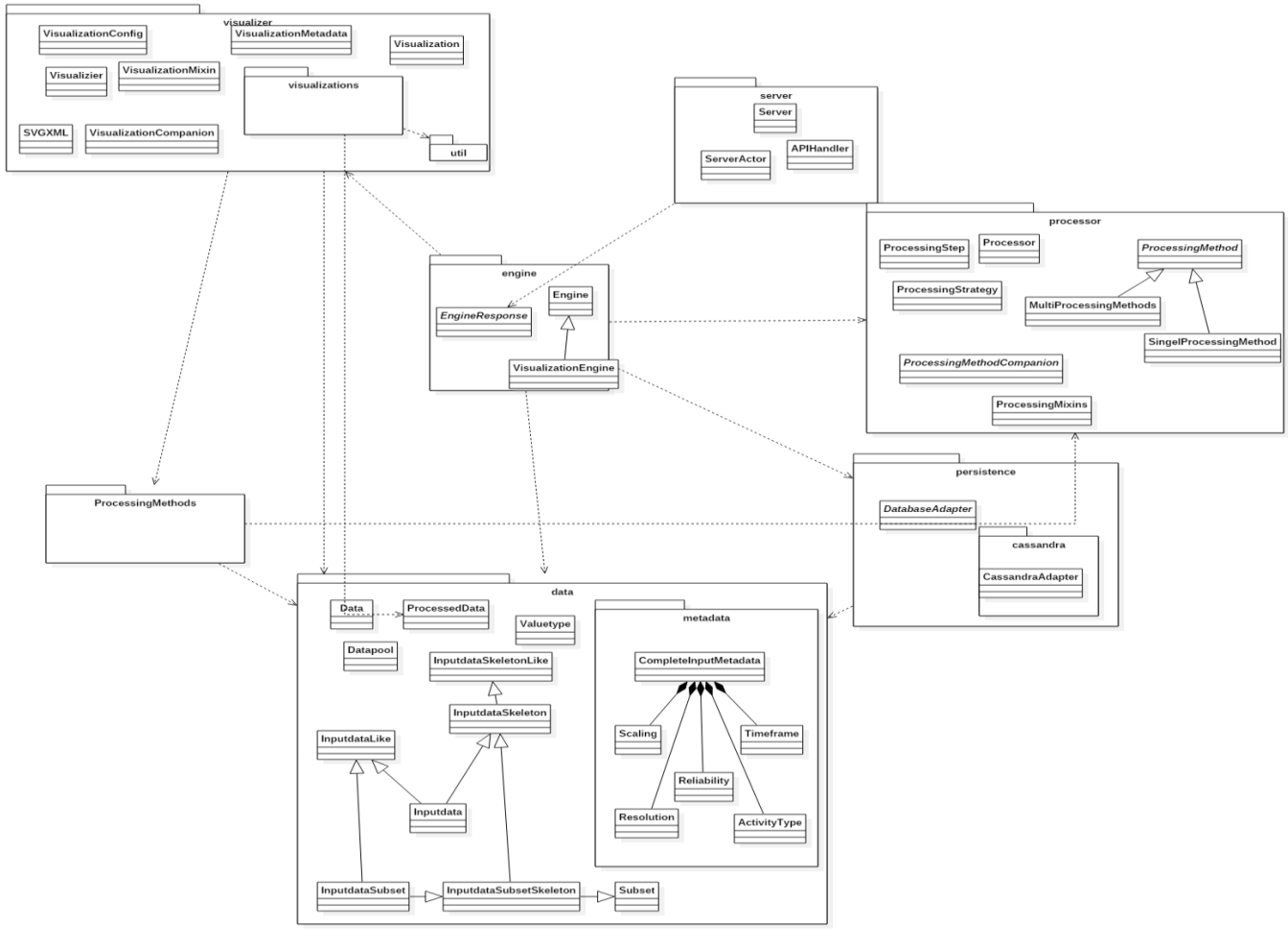
Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

ARCHITECTURE

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

Architecture

This section shows the implemented package structure. Dependencies are shown only between packages and not between classes. For facility of inspection reasons, please note, that some rather unimportant classes are not part of this model.



Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

TECHNOLGICAL OVERVIEW

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

Technological Overview

In this part, we would like to give an overview on the tools and technologies used during implementation of the system. On one side, the purpose of this section is to present some technologies, which came up recently and might not be well known in the community yet. On the other side, we present advantages of our chosen frameworks as well as a short description of alternatives and their disadvantages.

Database

Cassandra is an open source distributed database management system. Regarding to a study performed by the University of Toronto in 2012, Cassandra is the clear winner in terms of scalability, having a maximum data throughput. According to the requirements defined together with our product owner SAP, the system is demanded to handle big amounts of high frequency data leading to BigData requests on the server.

Having this requirement in mind, it quickly becomes clear that traditional databases like Microsoft SQL Server, PostgreSQL or other relational systems can not meet those prerequisites. Hence, the given data structure forces the use of a cluster-able NoSQL database like Cassandra. Popular alternatives are MongoDB and Hbase, but those do not provide write operations with the same speed as Cassandra. In addition, Cassandra's query language called CQL (Cassandra Query Language) is in its style similar to SQL and makes administration and querying simple, even for unexperienced users. Since 2010 Cassandra has been seen as one of the Top-Level projects of the Apache Software foundation and its monthly release cycles help to keep its state-of-the-art status. Naming a reference, Twitter is using Cassandra as well.

The Toolkit Phantom provides an elegant interface to query Cassandra from Scala-Applications.

Engine

Being the core of the system, the implementation of the engine is one of the most critical points. After weighting the alternatives, we decided to use Scala for the following reasons. At first, Scala provides full support for functional programming, which is likely to facilitate

the necessary data manipulations. In addition, the static type system is a powerful tool for the development of an extensible software architecture and design.

With this extensibility of the system, SAP can easily add new visualizations or processing methods. Apart from few exceptions, we developed with respect to the Open/Closed principle. Although other SOLID principles where respected, too, Open/Closed was our main focus.

During Scala development, the build tool sbt was used, which is the Scala equivalent to Java Ant or Maven. The server and API functionalities of our system are realized using the Scala concurrency framework Akka. The Spray Toolkit is based upon Akka and enables our server to be fast, lightweight and asynchronous.

Visualizations

Our engine creates visualizations i.e. diagrams without using any framework support. The output of the engine are pure SVG graphics. Discussed alternatives were standard picture formats like JPG and PNG, or to return sourcecode that the frontend has to embed. In comparison with those alternatives, SVGs are easier to create and easier to be transferred via http than PNGs. Also, they are way easier to handle for the frontend applications than any sourcecode output.

Explicitly no framework was used to create or render the graphics. Hence, we have full control of every point and line in the charts. SVG is size-independent and preserves its components (retained mode). Thus manipulation on the client side and event listeners are possible. SVG is an image format that is supported on the major platforms.
Also, creating and rendering SVG ourselves is way faster than the popular tool D3. Our prototypical benchmark showed an advantage of factor 10 in rendering runtime, probably because of D3's large overhead.

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

Application

To demonstrate the use of the engine, we designed a frontend application. This app was developed with JavaScript, HTML and CSS. This means a browser is required to launch the application. The application can be found at location *app/index.html* in the installation folder and is optimized for Google Chrome.

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

DEPLOYMENT

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

Installation

The “360° - My Health and Fitness Monitor” consists mainly of three different components. Firstly, we have a Cassandra Database managing all the collected health data. The second part is an engine handling all necessary calculations, deductions and the creation of visualizations. To exemplify the use of this engine, we also developed a frontend application. This application enables the user to select several datasets and can be used in a browser. To enjoy the whole experience of our system, we recommend you to set up all three components. This document will show you how:

Cassandra Database

To set up your one local Cassandra database, please visit

https://docs.datastax.com/en/cassandra_win/2.2/cassandra/install/installWin.html

and follow the given instructions.

To demonstrate the use of our engine with a remote database, a server at University of Augsburg is hosting a Cassandra instance. Using the default settings of the engine, a connection to that server with our running default Database will be established. For this connection, a VPN connection to the University of Augsburg is necessary. Our recommended tool for this purpose is “Cisco VPN Client”. Please keep in mind that access right are necessary for a such connections.

The server address is the following:

IP: 137.250.170.136 **Keyspace:** threesixty

In order to set up your own server or use your local database, you can modify the credentials in *config/application.conf* in the installation folder of the engine.

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

Health Engine

For easy and fast installation of the actual health engine, we provide an installer .msi file. Simply open the threesixty.msi and follow the instructions. It has been tested on Windows 10 (64bit). After successful installation, the engine listens on Port 8080 by default. You can change these settings in the configuration file *config/application.conf* where the very same can be done for the connection with the Database.

To create / recompile the installer use the sbt command depending on your target OS, e.g.

Windows:

```
sbt windows:packageBin
```

Ubuntu:

```
sbt debian:packageBin
```

Application

To demonstrate the use of the engine, we designed a frontend application. This app was developed with JavaScript, HTML and CSS. This means a browser is required to launch the application. The application can be found at location *app/index.html* in the installation folder and is optimized for Google Chrome.

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

API

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

Unless specified otherwise, the engine requires well-formed JSON for all its requests.

Visualization request

The most common call uses `"type": "visualization"`. It is used to request a visualization with certain parameters.

```
{
    "type": "visualization",
    "data": [ ],
    "visualization": { },
    "processor": [ ]
}
```

Parameters

- `data` *required*: A list of identifiers of datasets which will be used for the visualization. The are either simple Strings when the whole dataset should be processed (**not recommended** for large datasets) or JSON objects defining a start and end point { "id": "foo", "from": 0, "to": 1000} where `from` and `to` are timestamps and can be omitted for a partially bounded selection.
- `visualization` *optional*: Specifies the desired visualization format. Is deduced when omitted.
- `processor` *optional*: A list of processing steps applied to the input data. Is deduced when omitted.

`visualization`

JSON object with the two keys `type` and `args`.

```
{
    "type": "FooVisualization",
    "args": { }
}
```

- `type` *required*: Name of the visualization
- `args` *required*: Object containing visualization specific parameters.

`args` always requires a `width` and a `height`.

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

For specific details on `name` and `args` of individual visualizations, see their usage info.

`processor`

List of JSON objects, each describing a single step in the processing pipeline.

A definition of a processing step contains the `method`, i.e. its name, and `args`.

```
{
    "method": "FooProcessingMethod",
    "args": { }
}
```

`args` always requires an `idMapping`. And ID-mapping specifies what datasets are processed and the IDs assigned to them after they are processed.

```
{
    "idMapping": {
        "Foo": "FooProcessed",
        "Bar": "Bar"
    }
}
```

In the above example, the two sets with IDs `"Foo"` and `"Bar"` are processed.

The result of processing `"Foo"` is stored as `"FooProcessed"` and can be accessed that way later on. Additionally, the original "Foo" can also still be accessed.

`"Bar"` on the other hand is overwritten with the result of the processing method. Other processing methods - or the visualization - accessing `"Bar"` later on, will receive the processed data. *The unprocessed data is no longer accessible*.

Additional parameters may be required depending on the processing method. See their specific usage info for details.

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

Data requests

Data requests can be used to send and retrieve raw data. The use `"type": "data"`

The action parameter defines which one it is:

- `"action": "insert"`
- `"action": "get"`

insert

To insert data, pass the data using the `"data"` key

```
{
    "type": "data",
    "action": "insert",
    "data": {

    }
}
```

Data follows the following format:

```
{
    "id": "Foo",
    "measurement": "Bar",
    "dataPoints": [ ],
    "metadata": { }
}
```

- `id` is the ID of the dataset
- `measurement` is an arbitrary String describing the data
- `dataPoints` is a list of datapoints
- `metadata` is a metadata object

dataPoints

Each datapoint must follow the format:

```
{
    "timestamp": 1234,
    "value": {
        "type": "int | double",
        "value": 12.34
```

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

```
    }
}
```

- `timestamp` ist the timestap for the measurement taken
- `value` is an object with the optional `type` being
  either `"int"` or `"double"` to define, what type it is, and `value` the actual value (**Note** that `Double` values with `type` being `int` are converted to Integers; when no `type` is given, `double` is assumed.)

`metadata`

The metadata object can contains the following keys (*all are optional*)

```
{
    "timeframe": {
        "start": 1234,
        "end": 4567
    },
    "reliability": "Device | User | Unknown",
    "resolution": "High | Middle | Low"
    "scaling": "Nominal | Ordinal",
    "acitivityType": {
        "name": "Foo"
    }
}
```

`Get`

Retrieveing data requires an ID of the dataset.

```
{
   "type": "data",
   "action": "get",
   "id": "someID"
}
```

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

Help requests / usage info

Calling the engine with just `"type": "help"` returns a general usage message.

```
{
    "type": "help"
}
```

Additionally specifying `"for"` gives usage messages for specific parts.

```
{
    "type": "help",
    "for": "visualizations"
}
```

`visualizations` lists available visualizations.

```
{
    "type": "help",
    "for": "processingmethods"
}
```

`processingmethods` lists available processing methods.

Using the name of a visualization or a processing method as returned by the two calls above, displays usage information for that specific visualization or processing method.

```
{
    "type": "help",
    "for": "data"
}
```

`data` lists help on how to get and insert data.

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

DATA & CORRELATION

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

Available Datasets

	AUTOMATIC ENTRY	MANUAL ENTRY
NOMINAL	• Brain Activity • GPS-Data	• Mood • Gender
ORDINAL	• Substances in blood stream • EDA (Skin conductance) • Sleep cycles	• Food consumed • Sustances consumed • Liquid consumed • Sexual activity
METRICAL (INTERVAL OR RATIO)	• Heart Rate • Weight • O_2-Saturation • Calories burned • Steps • Velocity • UV-Radiation • Altitude • Environmental air pressure • Testosterone in blood stream • Blood sugar • Alcohol • Blood Pressure • V_{O_2} (Breath Volume) • Body fat • Basal temperature • Environmental temperature • Accelerometer • Gyroscope	• Age • Size • BMI

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle

Potentially interesting correlations

Calories burned ↔ Body weight (Chart with trend line)

Velocity / Activity ↔ Heart rate / Blood pressure (Heatmap, colour-coded chart)

Altitude / Environmental temperature ↔ Velocity / V_{O_2} (Chart with secondary coding in thickness)

Training duration ↔ Heart rate / Blood sugar / Liquid consumed / Calories burned

Sleep cycles ↔ Mood / Heart rate / Basal temperature

Food consumed – Mood / Weight / Velocity / Blood sugar / Blood pressure

Steps ↔ Weight / Mood / Sleep cycles

Weight / Activity ↔ Body fat

UV / Environmental temperature / air pressure ↔ Workout duration / Mood / Calories burned / V_{O_2} / Velocity

Traning duration / Heart rate / Blood pressure ↔ EDA

EDA ↔ Mood / Heart rate (Clustermap, Heatmap)

Stefan Cimander
Thomas Engel
Markus Schnappinger
Thomas Weber
Jens Wöhrle