Lehrstuhl für Systeme und Informationsverwaltung

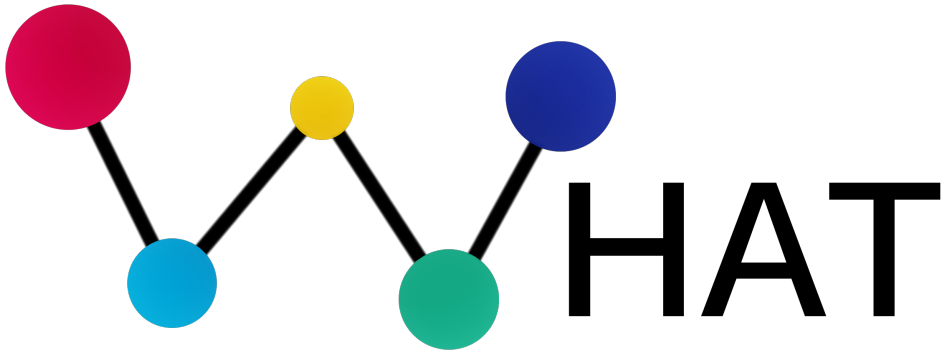
Prof.Dr.Ing. Klemens Böhm
Hoang Vu Nguyen
Marco Neumann

Karlsruher Institut für Technologie

Praxis der Softwareentwicklung (PSE)
WS 2012/2013

Visualizing and Statistically Analyzing Access Behavior to Scientific Databases

Design



WareHouse Analyzing Tool

December 14, 2012

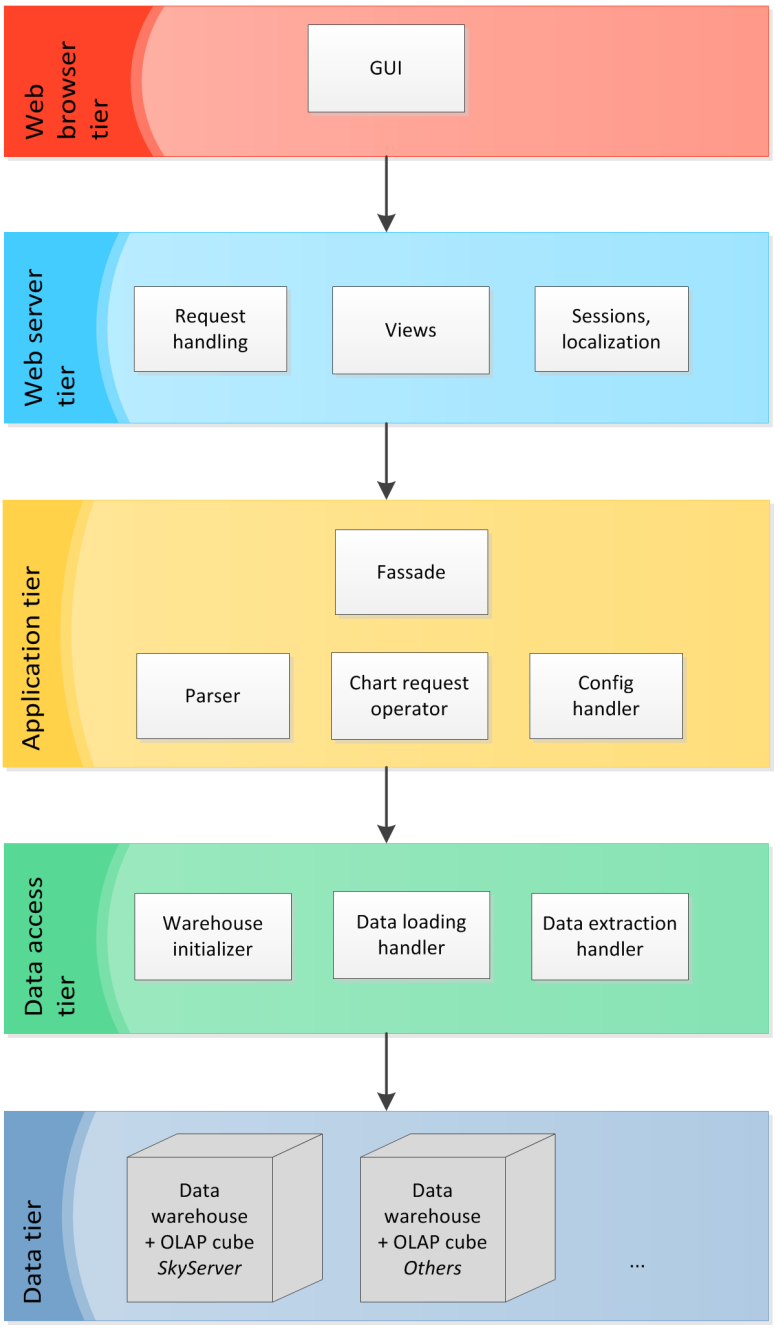
Functional Specification **Alexander Noe**
Design **Jonathan Klawitter**
Implementation **Anas Saber**
QA / Testing **Nikolaos Alexandros Kurt Moraitakis**
Final **Lukas Ehnle**

E-M@il: pse10-group14-ws12@ira.uni-karlsruhe.de

Contents

1 Architecture

WHAT follows an intransparent multitier architecture. The GUI, presentation logic, application processing, data accessing and storing data are logically and also partly locally separated. Intransparent means that communication between tiers just happens between adjacent ones. The different tiers are described below.



1.1 Web browser tier

The web browser tier represents the GUI of the client, which will be displayed as a web page. In the following context GUI and web page will be used interchangeably, as they are they same thing. The webpage utilizes javascript. It handles part of the user interaction with the program.

1.2 Web server tier

The web server provides the presentation logic. This includes both the serving of static files as well as the dynamic integration of content in the html pages. Other tasks are session handling, language localization and most importantly request handling.

It handles the rest of the user interaction with the webpage, while invoking the application tier when needed. This tier uses and is tightly integrated with java Play Framework.

1.3 Application tier

The application tier's function is twofold. On the one hand, it handles the parsing process and the management of configuration files. On the other hand, it acts as a sort of middleman between the web server tier and the data access tier.

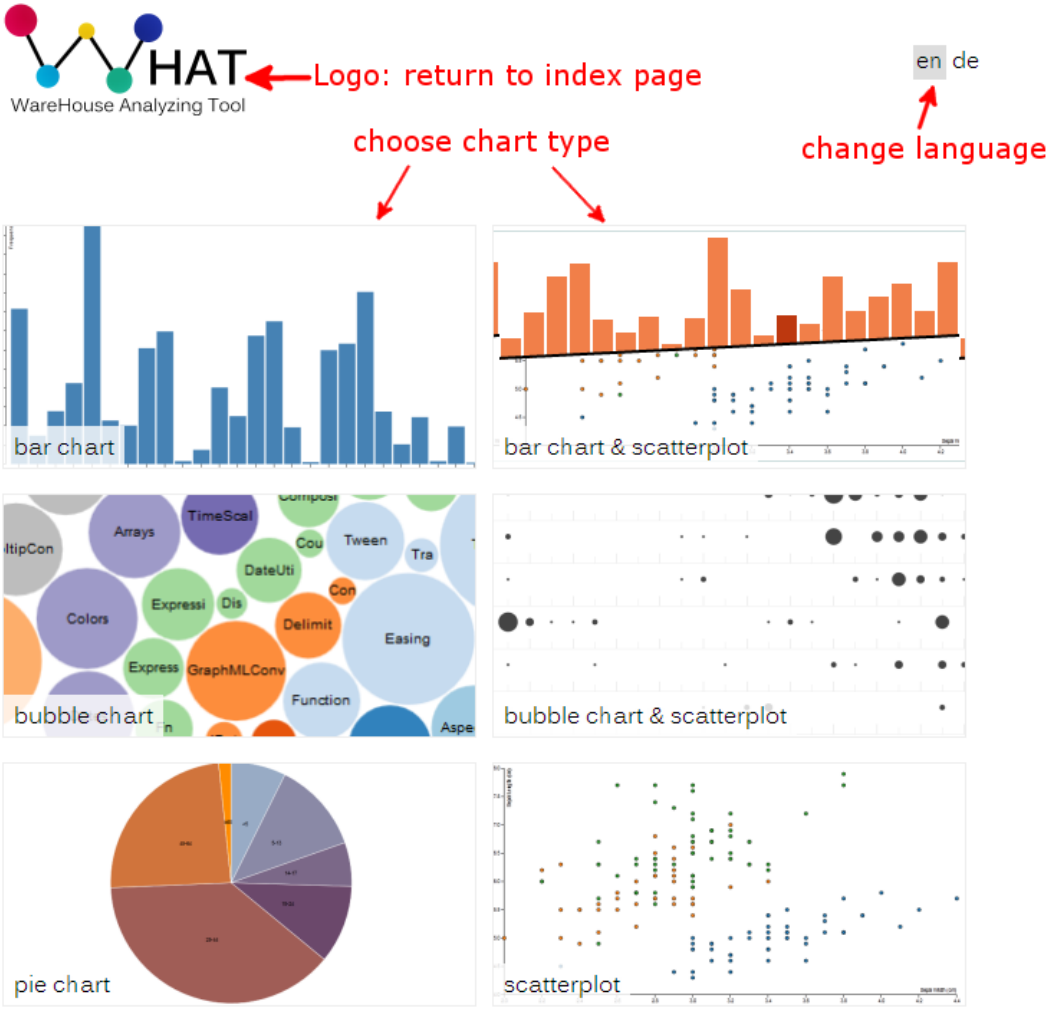
1.4 Data access tier

This tier manages all requests of loading and extracting data from the data tier. So its main task is to build a bridge from the application in Java to the SQL language of the Oracle warehouses and OLAP-Cubes. If there is enough time to implement this optional function, it will also handle the automatic initialization of new data warehouses.

1.5 Data tier

In the data tier the data warehouses and their OLAP-Cubes are stored. This will be done with the Oracle software.

2 Web page design

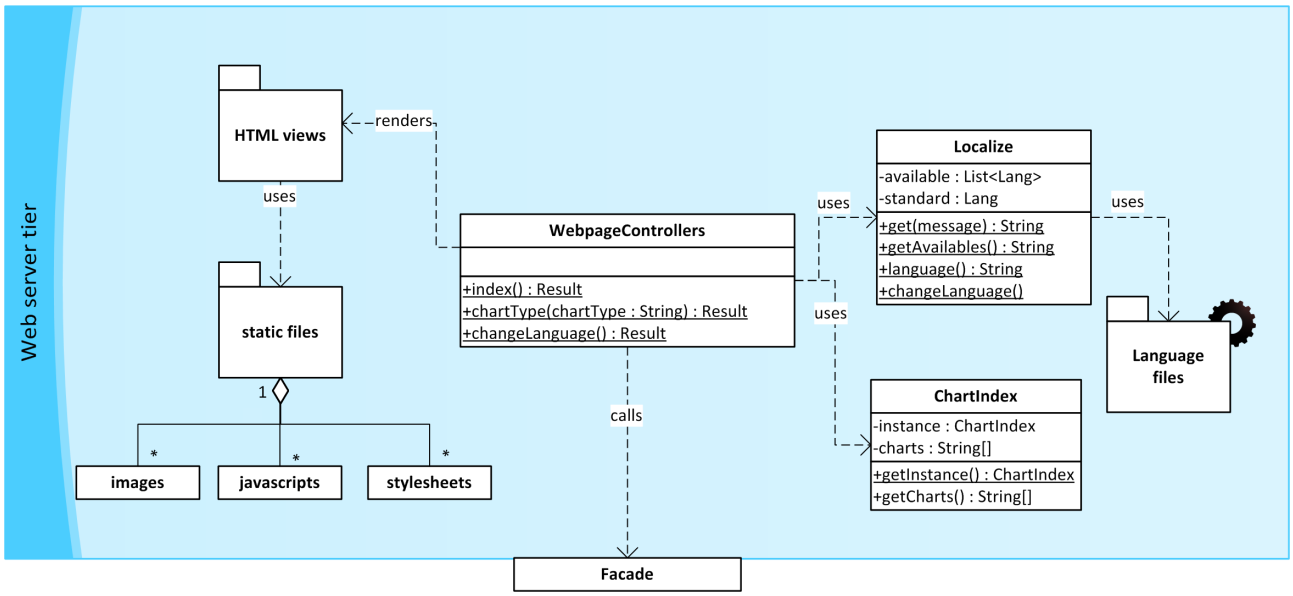


This is a prototype webpage, to show what the GUI could look like. It is not yet complete. There will be an admin login to access a password protected page where you can alter some settings of the GUI, but its main purpose is to allow the admin to initialize the data warehouse with logfiles.

3 Classes

3.1 Server tier class diagram

This diagram shows the web server tier of WHAT. It controls the web server and it can only access the facade of the tier below.



WebpageControllers

All valid HTTP requests are mapped to a method from WebpageControllers. WebpageControllers either uses other helper classes to handle requests itself or makes calls to the facade of the application tier. The last step done by WebpageControllers is to create a valid HTTP response.

HTML views

The HTML views contain the main template for, and all other HTML content of the web page. They do not have to be static, but can also dynamically integrate some content, e.g. localized strings.

Static files

Static files contains all files of the web page, which are not changed during runtime. This includes, but is not limited, to images of the web page, javascript files and stylesheets.

Localize

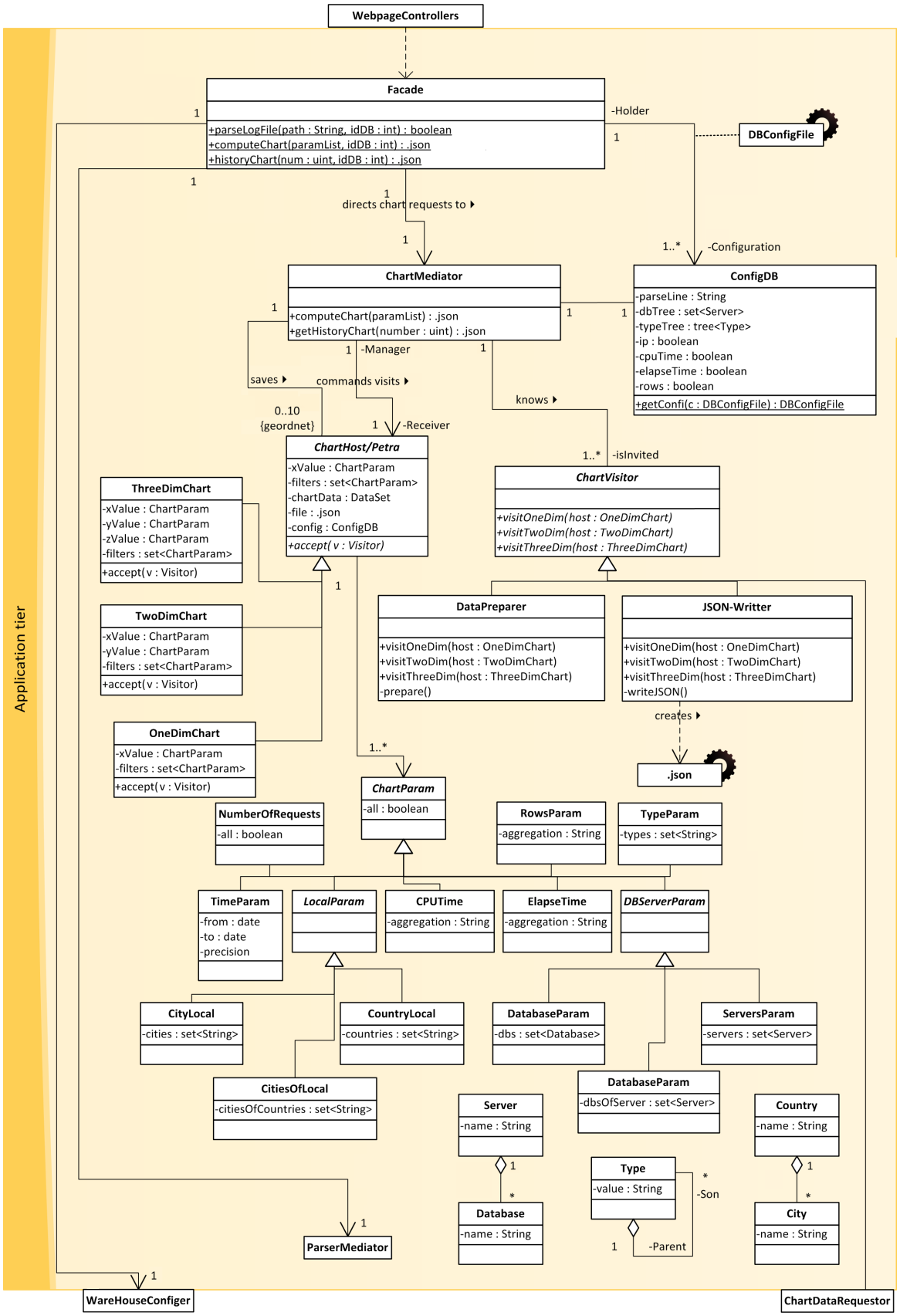
Localize is a static helper class handling all the language related things. It has a method to get localized Strings using the language files to change the language of the web page. It also has methods determining all available languages, allowing integration in the webpage if they are present.

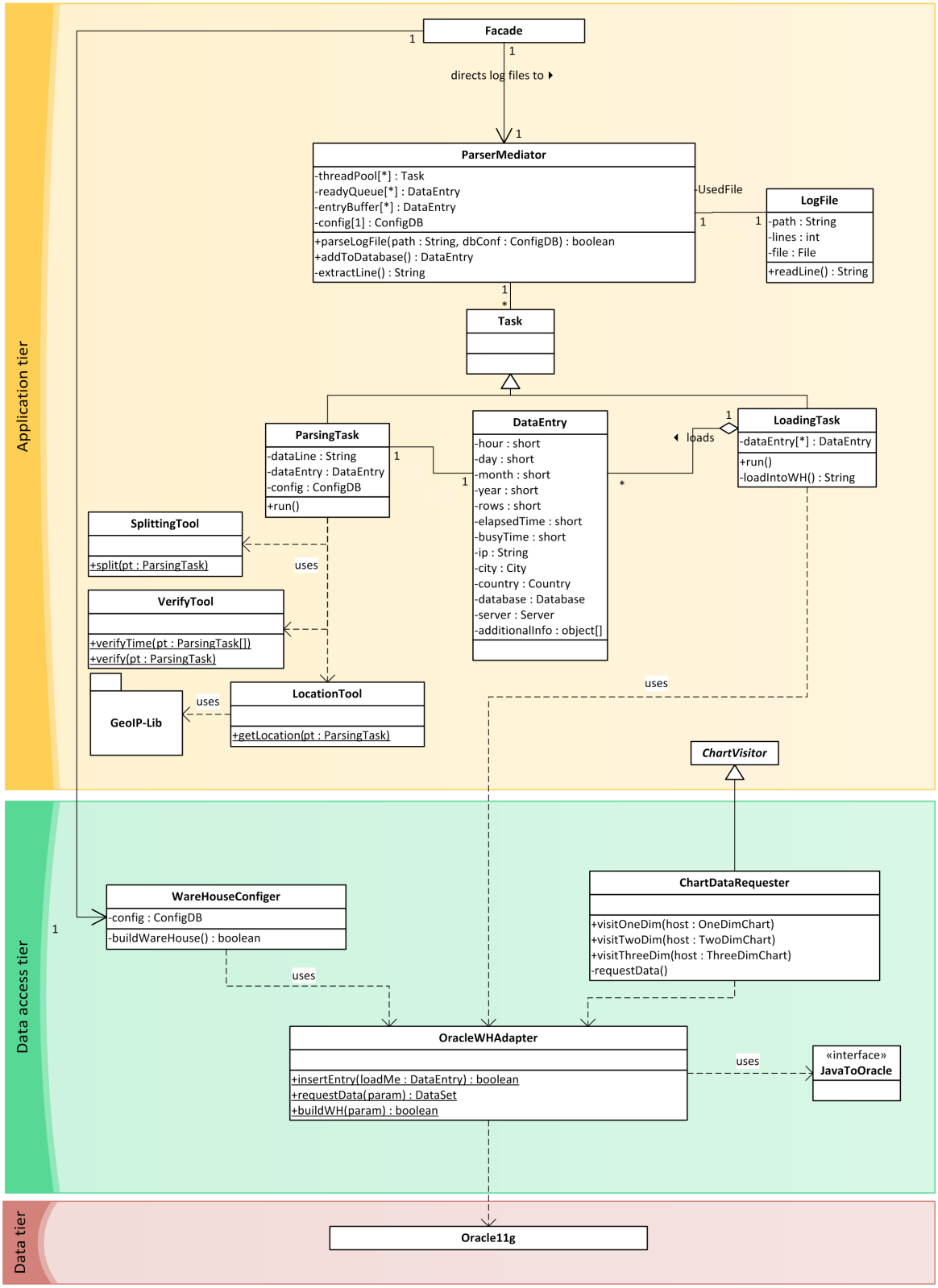
ChartIndex

ChartIndex implements the singleton pattern. On instantiation it scans which chart types are available and saves them internally. This is used by the index page to dynamically display all available chart types, even if one is deleted or added.

3.2 Application & Data access tier class diagram

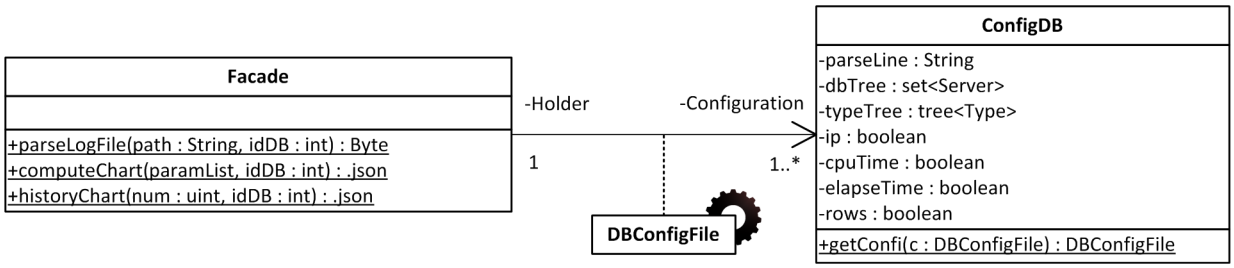
On the next pages the class diagram of the application and data access tier are displayed.


WebpageControllers
Facade
+parseLogFile(path : String, idDB : int) : boolean
+computeChart(paramList, idDB : int) : .json
+historyChart(num : uint, idDB : int) : .json
-Holder
DBConfigFile
directs chart requests to
-Configuration
ChartMediator
+computeChart(paramList) : .json
+getHistoryChart(number : uint) : .json
-Manager
ConfigDB
-parseLine : String
-dbTree : set<Server>
-typeTree : tree<Type>
-ip : boolean
-cpuTime : boolean
-elapseTime : boolean
-rows : boolean
+getConfi(c : DBConfigFile) : DBConfigFile
saves
commands visits
knows
0..10
{geordnet}
-Receiver
ChartHost/Petra
-xValue : ChartParam
-filters : set<ChartParam>
-chartData : DataSet
-file : .json
-config : ConfigDB
+accept(v : Visitor)
-isInvited
ChartVisitor
+visitOneDim(host : OneDimChart)
+visitTwoDim(host : TwoDimChart)
+visitThreeDim(host : ThreeDimChart)
ThreeDimChart
-xValue : ChartParam
-yValue : ChartParam
-zValue : ChartParam
-filters : set<ChartParam>
+accept(v : Visitor)
TwoDimChart
-xValue : ChartParam
-yValue : ChartParam
-filters : set<ChartParam>
+accept(v : Visitor)
OneDimChart
-xValue : ChartParam
-filters : set<ChartParam>
+accept(v : Visitor)
DataPreparer
+visitOneDim(host : OneDimChart)
+visitTwoDim(host : TwoDimChart)
+visitThreeDim(host : ThreeDimChart)
-prepare()
JSON-Writter
+visitOneDim(host : OneDimChart)
+visitTwoDim(host : TwoDimChart)
+visitThreeDim(host : ThreeDimChart)
-writeJSON()
creates
.json
ChartParam
-all : boolean
NumberOfRequests
-all : boolean
RowsParam
-aggregation : String
TypeParam
-types : set<String>
TimeParam
-from : date
-to : date
-precision
LocalParam
CPUTime
-aggregation : String
ElapseTime
-aggregation : String
DBServerParam
CityLocal
-cities : set<String>
CountryLocal
-countries : set<String>
DatabaseParam
-dbs : set<Database>
ServersParam
-servers : set<Server>
CitiesOfLocal
-citiesOfCountries : set<String>
DatabaseParam
-dbsOfServer : set<Server>
Server
-name : String
Database
-name : String
Type
-value : String
-Son
-Parent
Country
-name : String
City
-name : String
ParserMediator
WareHouseConfiger
ChartDataRequestor
Application tier


Application tier
Facade
directs log files to
ParserMediator
-threadPool[*] : Task
-readyQueue[*] : DataEntry
-entryBuffer[*] : DataEntry
-config[1] : ConfigDB
+parseLogFile(path : String, dbConf : ConfigDB) : boolean
+addToDatabase() : DataEntry
-extractLine() : String
-UsedFile
LogFile
-path : String
-lines : int
-file : File
+readLine() : String
Task
ParsingTask
-dataLine : String
-dataEntry : DataEntry
-config : ConfigDB
+run()
DataEntry
-hour : short
-day : short
-month : short
-year : short
-rows : short
-elapsedTime : short
-busyTime : short
-ip : String
-city : City
-country : Country
-database : Database
-server : Server
-additionalInfo : object[]
loads
LoadingTask
-dataEntry[*] : DataEntry
+run()
-loadIntoWH() : String
SplittingTool
+split(pt : ParsingTask)
uses
VerifyTool
+verifyTime(pt : ParsingTask[])
+verify(pt : ParsingTask)
GeoIP-Lib
uses
LocationTool
+getLocation(pt : ParsingTask)
uses
ChartVisitor
Data access tier
WareHouseConfiger
-config : ConfigDB
-buildWareHouse() : boolean
uses
ChartDataRequester
+visitOneDim(host : OneDimChart)
+visitTwoDim(host : TwoDimChart)
+visitThreeDim(host : ThreeDimChart)
-requestData()
OracleWHAdapter
+insertEntry(loadMe : DataEntry) : boolean
+requestData(param) : DataSet
+buildWH(param) : boolean
uses
«interface»
JavaToOracle
Data tier
Oracle11g

3.3 Facade + Configurations

The facade of the application tier and configuration files in this design fulfill an optional feature. According to the database on which is operated - if there are more than SkyServer - just a configuration file for this specific one, has to be created and given to the program. This may give the application the power to manage all functions dynamically according to the database. Facade and ConfigDB handle this configurations.



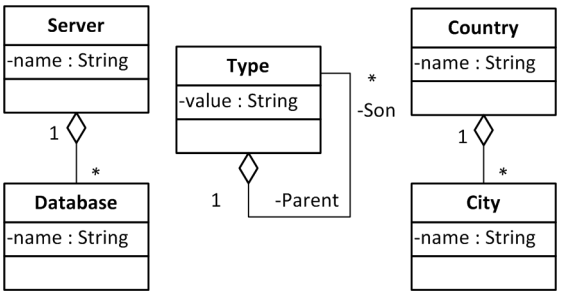
Facade

The Facade is, as you might have suspected, the facade of the application tier. This means the only way to communicate with this tier is via this class. One of its tasks is to get the right ConfigDB for any call. With this it forwards them to the corresponding mediators or workers. So new log files are passed to the ParserMediator and chart requests to the ChartMediator. Some other optional calls, like the history of charts or the configuration of a new warehouse are just adumbrated.

ConfigDB

The ConfigDB class provides a static factory for creating itself out of a configuration file, which may use caching. Objects of ConfigDB store the information about the database they describe and will be used by all the dynamic methods, which are hooked on specific databases.

One thing stored in a ConfigDB is a String indicating the form of a line in the log files. Other attributes are booleans for specific measures or dimensions, as well as trees of Strings for dimensions like Database > Server or the Type.

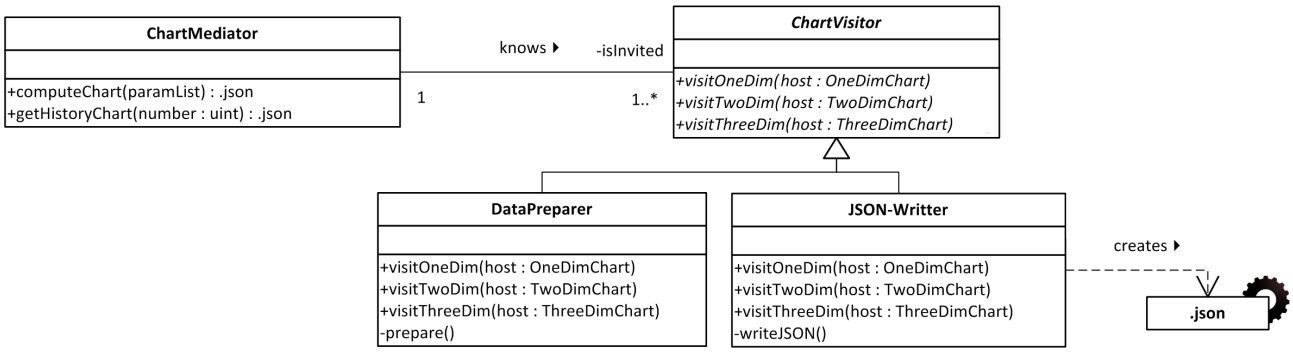


Server, Database, Type, Country, City

All this classes are Stringwrappers and also build as trees. They stand exemplary for possible things stored in a ConfigDB. The implementation of them may be way more dynamic. Whereas the content of Server & Database and Type are depended on the configuration, the Country & City tree is mostly hooked to the GeoIP library.

3.4 Chart request operator

The task of the following classes is to handle a chart request. For their design the visitor pattern is used.



ChartMediator

The ChartMediator is the mediator of the whole chart request process. For a incoming request it will create a new ChartHost firstly. Then it triggers the visits of the three visitors.

ChartVisitor

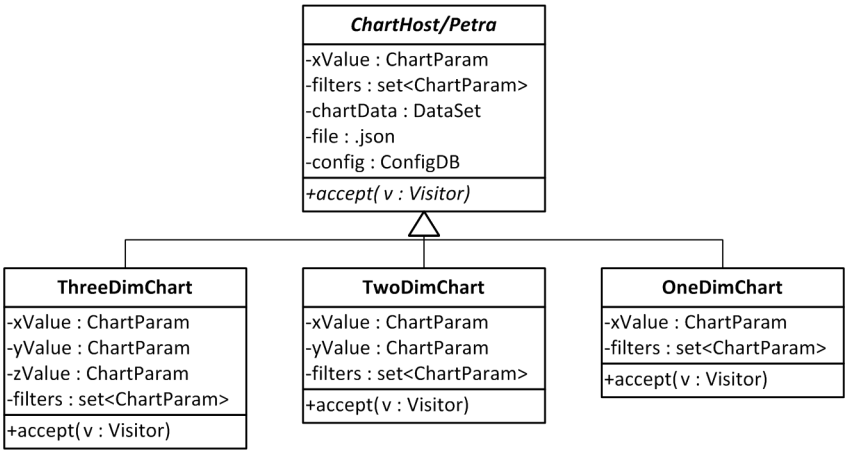
ChartVisitors work on a ChartHost. What the visitors do depends on which specific ChartHost is visited and of course on their type. One visitor, the ChartDataRequester, is part of the data access tier.

DataPreparer

The DataPreparer is the second visitor. It prepares the raw data stored in a ChartHost, so that the JSON-Writter can do its work easily.

JSON-Writter

The JSON-Writter is the last visitor. According to the data stored in the visited ChartHost he creates a .json file which contains all information needed by D3 to display the charts.



ChartHost/Petra

ChartHosts are objects hosting one chart request. Once initialized, they store the chart parameteres requested for the axes and the filter options. During the visits they first store raw data and later the .json file needed for the chart. The other name Petra for ChartHost is the intern name used for this class. As it had no official name it was and still is known by Petra. So because when mentioning Petra all current members know what is meant the name is also given in this design document.

OneDimChart

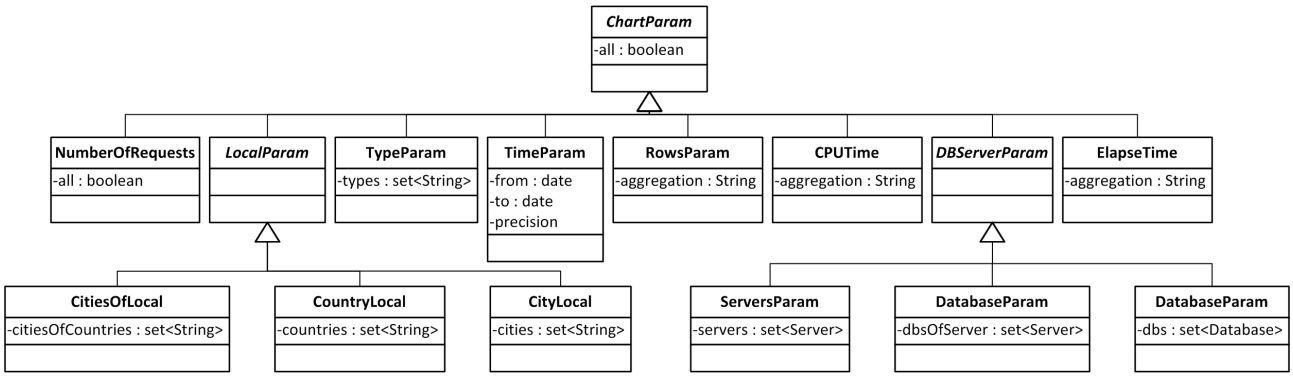
A OneDimChart - the name standing for one dimension chart - stores the ChartParam for the x axes or for example just the measure/radius needed in a bubble chart.

TwoDimChart

More than the OneDimeChart the TwoDimChart also stores the ChartParam of a second axis.

ThreeDimChart

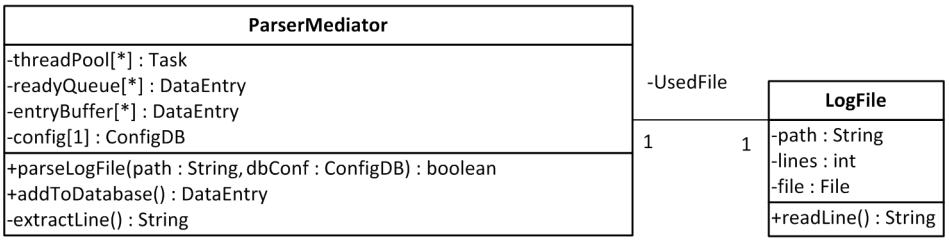
The ThreeDimChart stores ChartParams for three axes.



ChartParam

ChartParams represent on the one hand the type of a dimension or measure, like time or number of rows. On the other hand they store the requested interval or the aggregation type, e.g. sum or maximum, by which it should be filtered. With a boolean set to true it is indicated, that no filtering is required on this parameter, or, if implemented an other way, they may be just left out in that case. This ChartParam stand just representative for possible chart parameter and may be implemented way more dynamically.

3.5 Parser



ParserMediator

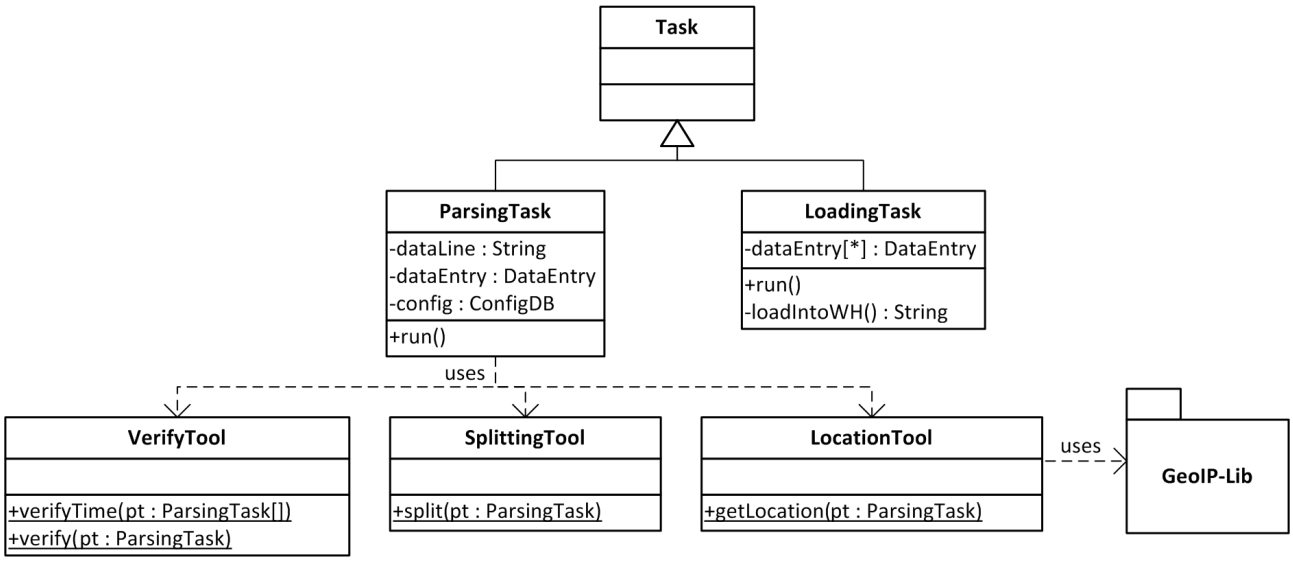
ParserMediator is the 'main'-Class of the Parser. It creates and administrates a thread-pool, which contains several tasks. The ParserMediator also contains the entryBuffer for finished DataEntries, the stringBuffer for strings, which were extracted from the log file and it saves which log file and configuration file is used.

LogFile

The LogFile class is the gateway between parser and a log file - it contains the path of the logfile and an integer which saves how many lines have been read from this file. It can read single lines from the logfile and return them to the Parser.

DataEntry

The DataEntry stores the data which will be written into the warehouse. The given attributes are representative for possible data. The implementation of them may be way more dynamically.



ParsingTask

ParsingTasks are tasks created by the ParserMediator. They receive a line from the log file and use SplittingTool, VerifyTool and LocationTool to create a DataEntry.

SplittingTool

The SplittingTool splits the dataLine from its parsingTask and enters the splitted raw parts into the DataEntry.

VerifyTool

The VerifyTool checks the DataEntry for its correctness. If mistakes are found this line will be deleted and an internal exception will be thrown.

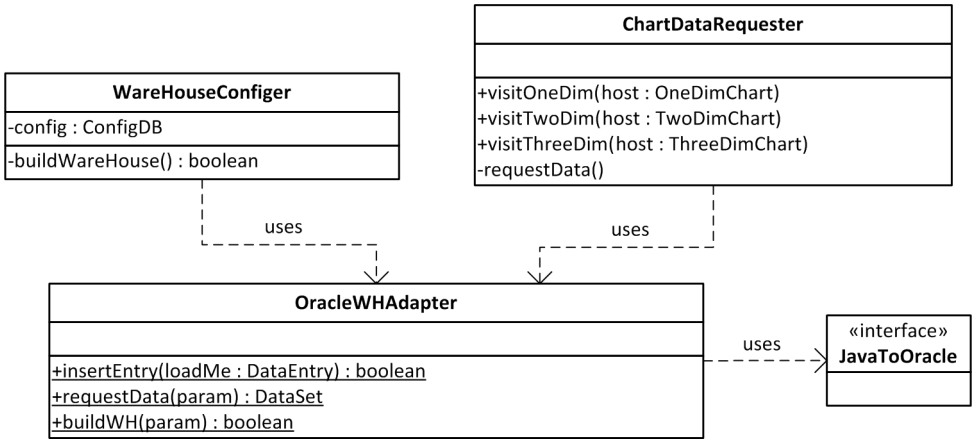
LocationTool

The LocationTool utilizes GeoIP-Libraries (7.3) to determine the city and country of the request using the IP.

LoadingTask

A LoadingTask is another possible task for the thread pool. It takes finished DataEntries from the buffer and sends them to the data access tier.

3.6 Data access tier



ChartDataRequester

The ChartDataRequester is the first of the three visitors of a ChartHost. Its task is to create the queries to gain the data needed for the charts. It takes the information about what it has to request from the hosted ChartHost.

OracleWHAdapter

Every query, whether loading, extracting or anything else, run against the Oracle server is made by this, and exclusively by this, class. So it represents the interface to the data.

JavaToOracle

Any library used for the connection and communication with the Oracle server, where the Warehouse is stored. It is only used by the OracleWHAdapter.

WareHouseConfiger

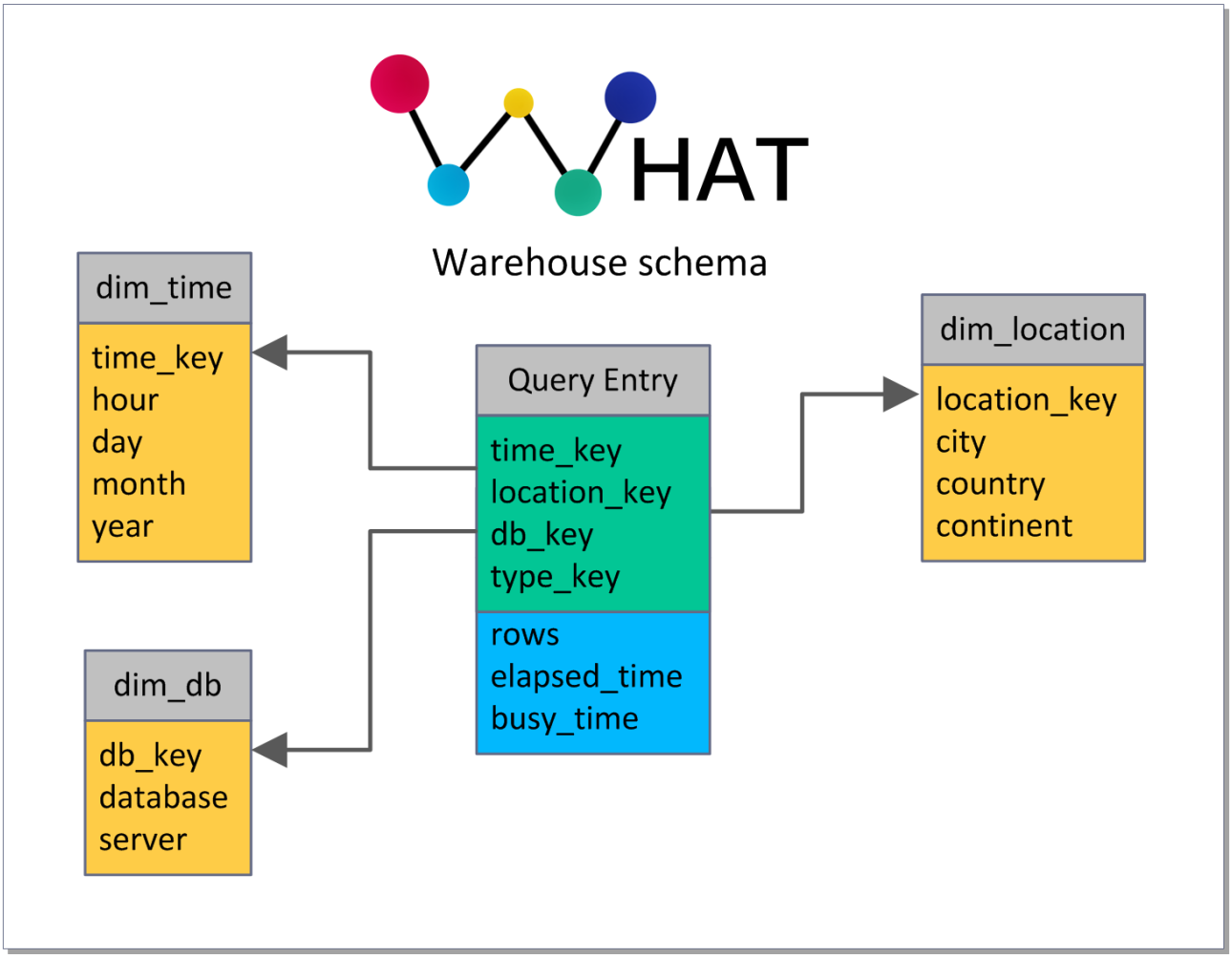
This class will only be needed if a very optional function will be implemented. It's task is to create a warehouse for one certain new database just with the information of its configuration file information.

4 Data warehouse design

The data warehouse stores all parsed information and creates OLAP-Cubes to achieve fast responses for even the most complex of queries.

4.1 Schema

The data may be stored in the following schema. If optional functions are implemented and the warehouses are configured dynamically, the schema may diverge from this.



4.2 Dimension descriptions

The only database independent dimensions are time and location. The time dimension is defined by us. The location dimension is given by the GeoIP-lib. (7.3).

BILD TIME DIMENSION

Other dimensions like database > server, type and maybe others, depend on the database operated on, e.g. SkyServer.

4.3 Measure descriptions

Measures are for example busy time (CPU time), elapse time and rows. The aggregations stored in the OLAP-Cubes for this measures are:

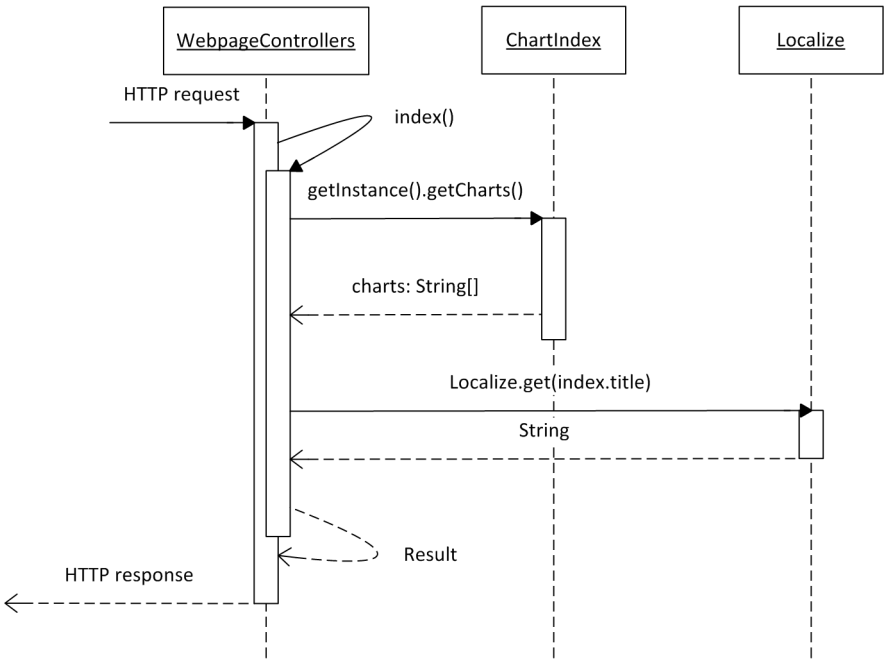
- sum of their value,
- maximum of the value overall in all levels below,
- maximum of the values stored for sum in the level below,
- number of entries in a level below.

5 Sequences

In this sections the important processes of the system are shown exemplarily. They include a request from the web page handled in the web server tier, a call on the facade of the application tier, a parsing process and a chart request.

5.1 HTTP request

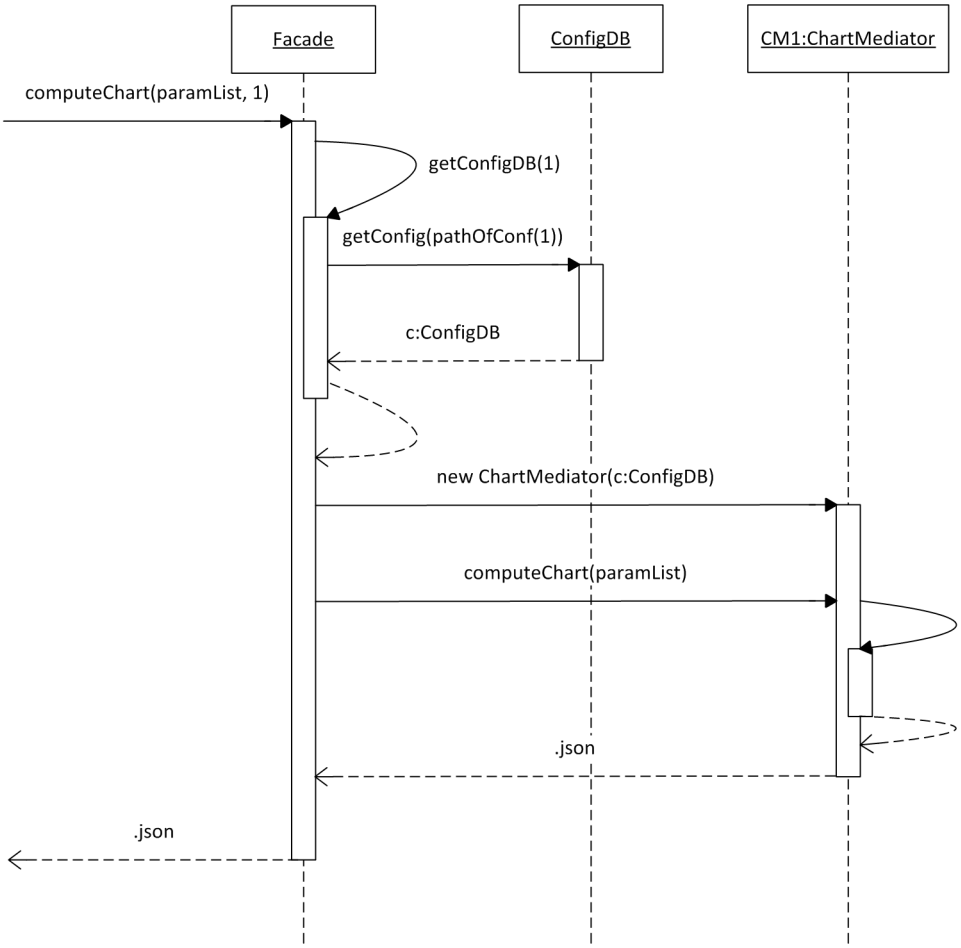
A HTTP request is triggered by the user on the web page.



»»»»»»» Noch dazu schreiben was für ein request das ist vll

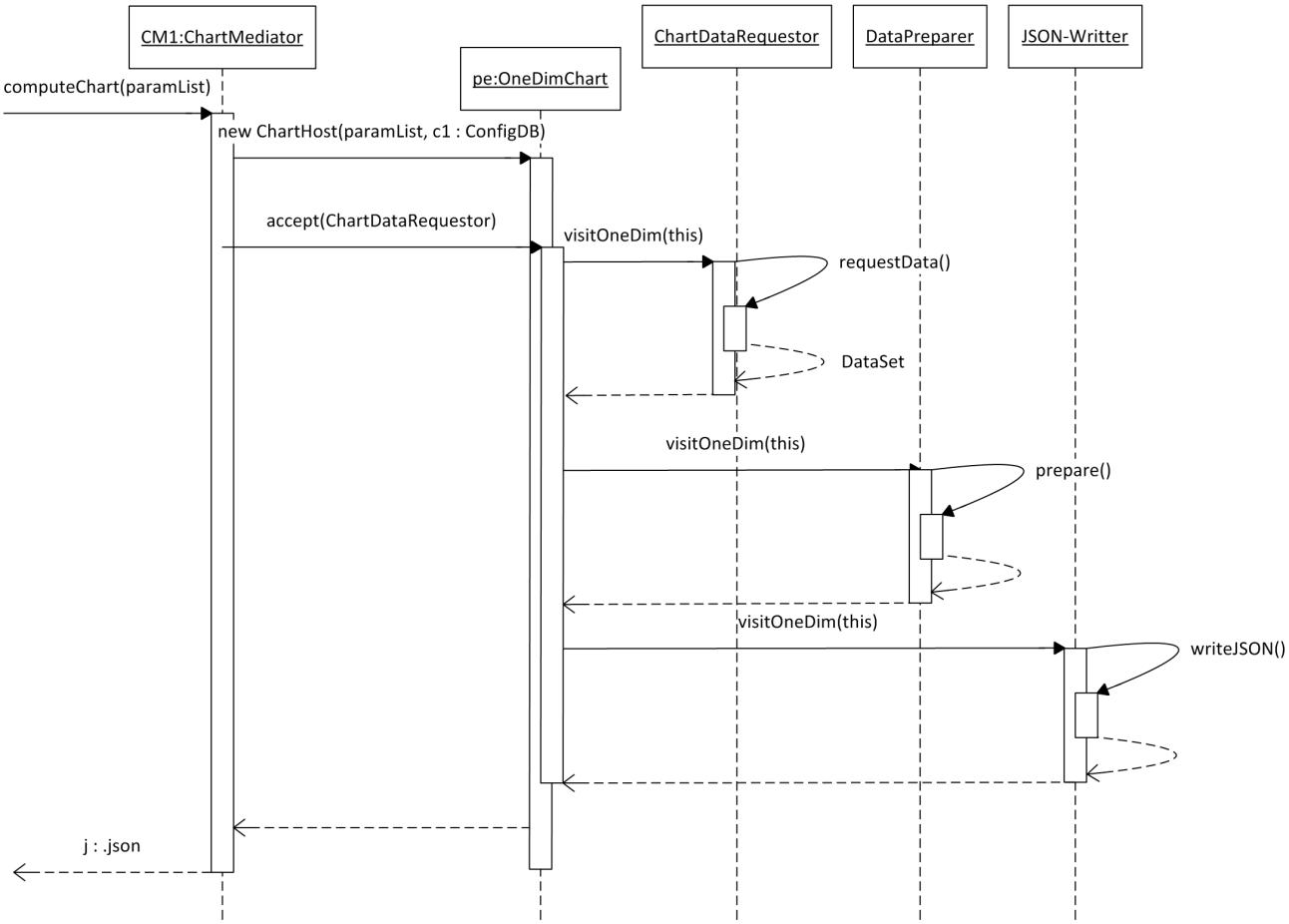
5.2 Facade call

In the following sequence diagram a call on the application tier facade is shown on the example of a chart request with `computeChart(..)`.



5.3 Chart request

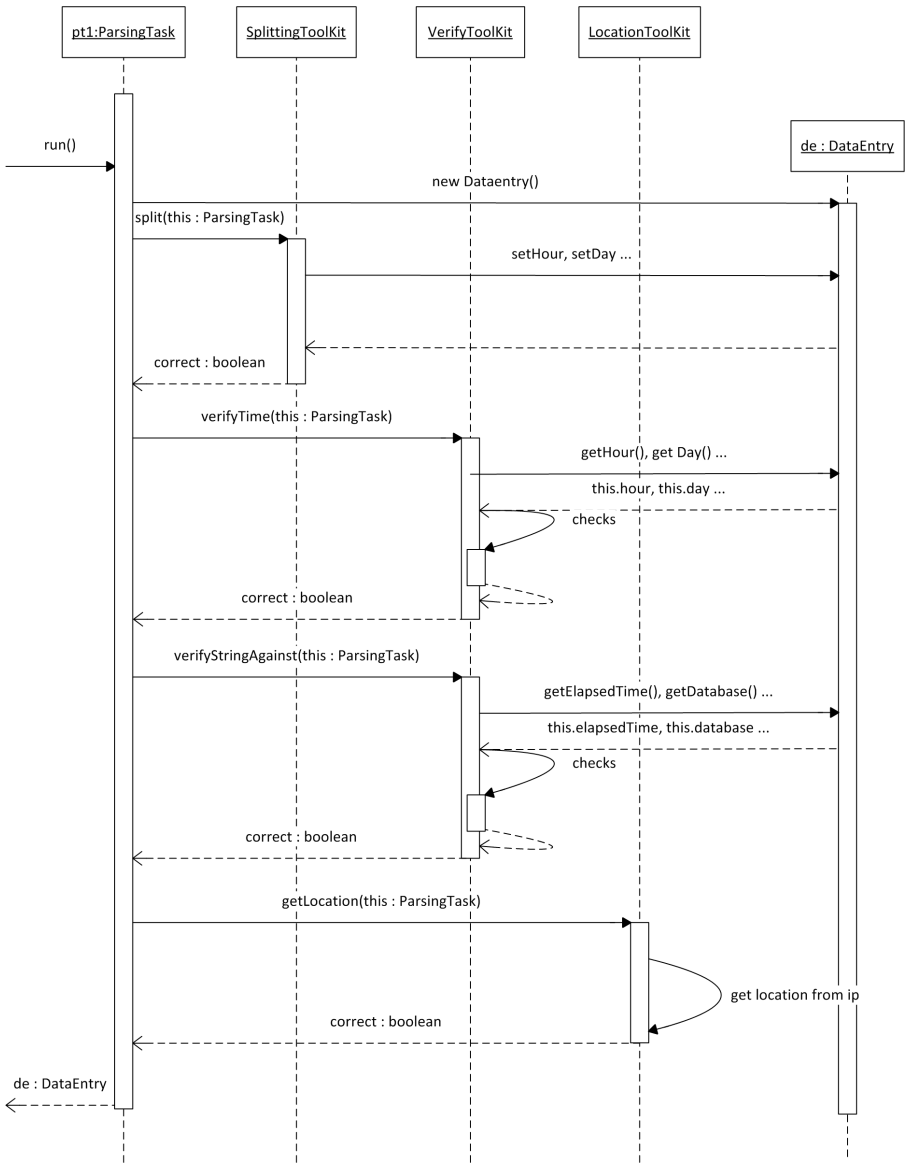
Sequence diagram of a `computeChart(..)` call, a chart request, on the ChartMediator.



5.4 Parsing

This activity diagram shows the generall work flow of the parsing process of a log file. The sequence diagram underneath describes this process with the point of view on class interaction.

AKTIVITY-PIC



6 Data

The program along with its libraries gets compiled to an executable .jar file. However, it also uses a number of external data.

6.1 Log files

Log files are arguably the most important data files our program has to deal with, since it simply needs them to function properly.

They have to be valid .csv files as well as be consistent with the configuration files specified. For more information on configuration files, read on.

6.2 Configuration files

This fulfills an optional requirement.

The program may work with more database log files than just with these from the skyserver without any modifications to its source code. However, to achieve this functionality configuration files have to be read. A configuration file needs to be written per database.

The configuration file contains, among others, the database schema, the dimensions and measures used as well as their types.

Additionally, we will provide a sample configuration file for the skyserver, as well as documentation on creating one.

6.3 Visualization files

The program dynamically recognizes javascript visualization files during its startup, provided they are created with our specifications in mind and are either placed in the appropriate directory structure, or are uploaded through the admin panel. That said, this allows new chart types, or different visualizations to be created by a user simply by providing the javascript file. Javascript files have a .js ending.

6.4 Language files

Our program keeps language files separate to enable a smooth translation and maintenance of the graphical user interface.

6.5 Misc

Favicons, logos, thumbnails for the chart types are also used.

7 Libraries

The program uses a number of external libraries to achieve its functionality. This section describes which they are, how they interact with the program, as well as licensing information.

7.1 Java play

Version: 2.0.4

Link: http://www.playframework.org/

Usage: Java play is primarily a web framework, but it includes a webserver too. We use it to create a modern, dynamic and interactive website, instead of a conventional java Swing GUI. Our web server tier is just a modification of appropriate java play classes.

License: Apache 2 License

7.2 D3.js

Version: v2

Link: http://d3js.org/

Usage: D3.js is a javascript visualization library for the browser. It allows the creation of beautiful, interactive visualizations. It is used, as one would imagine, in our browser tier.

License: BSD License, Clause 3

7.3 GeoIP

Version: 1.2.8

Link: https://www.maxmind.com/download/geoip/api/java/

Usage: We use the Maxmind GeoIP Lite library to map user IPs to locations. This allows queries based on location, as well as preserving anonymity. Our parser uses the GeoIP library. **License:** LGPL

7.4 Oracle Data Warehouse & OLAP software

Version: 11g Release 2

Link: Oracle Data Warehousing

Usage: Data will be stored with an Oracle warehouse and Oracle OLAP-Cubes. We will use the software over the KIT.

License: Oracle license