ON RAIN INFORMATION AS MAP FEATURES FOR CAR NAVIGATION SYSTEMS

Martin Gebert, Thomas Berroth and J.-Emeterio Navarro-B.*

MBition GmbH, Daimler AG, Dovestrasse 1, 10587 Berlin, Germany
(martin.gebert, thomas.berroth, jesus_emeterio.navarro-barrientos)@daimler.com

KEY WORDS: rain radar, weather, microservices, cloud computing, digital mapping, map features.

ABSTRACT:

In this paper we present a design concept, architecture and implementation of a microservice to process and integrate rain information into a car navigation system in the form of rain map features. Two different input data sources are considered: QuadTile JSON format and GeoTIFF images. Our system converts this input data into an ouput GeoJSON format with only the most relevant information for the map overlay system in the navigation system of the car. We discuss different options for the cloud appearance, like color, shape and transparency. We present our microservices architecture together with data pipelines and implementation. Our approach allows for low latency and spare computing resources, which are especially needed in embedded systems. Finally, we discuss the advantages and disadvantages of our approach as well as further work.

1. INTRODUCTION

Rain is one of the main causes of car accidents in the world. Not only it impedes the clear visibility of the road but it also makes the road slippery, the brakes wet and in general the driving conditions unpredictable, leading to high probability of traffic incidents. Large amounts of rain often lead to severe flooding in urban areas, which cause also traffic congestion, pollution and in general unpleasant situations (especially for inexperienced drivers) that could be avoided if the driver could be advised of any upcoming rainfall on the road (Kyte et al., 2000).

The main goal of this paper is to present a backend service architecture that displays rain information on a digital map in the form of a semi-transparent overlay consisting of a number of polygons with specific color-coding (e.g. blue, yellow, red, etc.) according to the intensity of the rain. The most typical rain measurement is the rate of precipitation expressed in millimeters per hour, where one millimeter of rainfall is the equivalent of one liter of water per square meter (Krajeswki and Smith, 2001). Note that the term *rainfall* is sometimes used to include not only amounts of rain, but also snow and hail. For simplicity, we use in the rest of the paper, the term *precipitation*.

Several rain radar services map their data to color in different manners. Most of them map the intensity of precipitation to a color gradient. Either a rainbow gradient, a limited hue gradient (for example from blue to yellow), a brightness gradient, or the alpha value of a color. For this, some guidelines are available for practical usage of colors in maps (Stauffer et al., 2015). Using these guidelines and the web portal *ColorBrewer*[1] for map designers, it is possible to choose certain color schemes to display rain data in a readable way (Brewer et al., 2003).

Besides the color, the geometrical representation of the rain clouds is of high importance. Typically, the visualization of rain radar information is left pixelized. This has the advantage that the measurements are shown just as close as they can be to the real rain information, i.e. no information is falsified. The disadvantage is that the form of the clouds is not visually appealing and does not look professional. Some researchers have proposed the use of algorithms to convert radar information into polygons for nicer visualization. Of special interest are the strategies to overlap radar polygons with other map features like points and lines (Hu, 2014).

It is also very important to review the driver experience with systems displaying precipitation information. It is common knowledge that extreme weather affects driver behavior (Kilpeläinen and Summala, 2007). And it can be that unexpected routing decisions taken by the intelligent routing algorithms may confuse the driver, who will question the correctness of the routing algorithm (Curzon et al., 2002). Recently, some studies presented the view of some drivers on including weather forecasts in the optimal routing of navigation systems (Kisters et al., 2019). In these investigations, most of the participants would accept to drive following routing recommendations based on weather forecasts. The authors discuss also the importance of finding the best way to communicate weather-related route changes to the users. Another study proposes the use of a route-guidance system to help drivers avoid heavy rainfalls (Ito, Sadanori and Koji, Zettsu, 2020). In this study, some participants were given a driving simulator to test four alert methods, three route options, and four levels of possible risk avoidance. The authors report that such a system has a 85.63% social acceptance, demonstrating the usefulness of such systems. Our approach in this paper, improves acceptance by the user, as it would provide extra context visual information to the driver to understand, validate and trust the routing computed by the car navigation system.

Nowadays, several companies provide weather data but focus mainly on global products. These are sometimes difficult to customize and have a limited compatibility especially for the purpose of displaying rain data in a mobile device or a car. To improve this situation, some work has been done to provide for example satellite rainfall estimates in different formats using Python-based web service and Android applications (Mantas et al., 2015). On the other hand, some researchers have focused their studies on providing data frameworks for managing weather data, for example WeatherBench (Rasp et al., 2020). Some other studies focus on precipitation only, for example,

* Corresponding author
[1] https://colorbrewer2.org/

RainBench, a dataset for studying global precipitation forecasting from satellite imagery. And PyRain, a framework to release user-friendly rain datasets, using pipelines to enable efficient processing of data by any modeling framework (Tong et al., 2020). Another example is *THOR*[2] (Tool for High-resolution Observation Review), which is a rainfall satellite image dataviewer that started as a desktop application and is now available also on a web browser (Kelley, 2013). The authors report that the simplicity of their approach can be useful for easy adapting and deploying applications to analyse and test new algorithms and products for precipitation visualization.

Finally, note that the novelty of this paper is to present an approach and implementation to provide precipitation information into car navigation systems in the form of a rain map feature. In Section 2, we discuss two data input sources, QuadTile JSON format and GeoTIFF image format; and the data preparation and data output suitable for car navigation systems. In Section 4 and Section 5, we present our architecture and implementation, resp. Finally, in Section 6 we present our conclusions and further work. To our knowledge, no previous research work has focused on describing concept, design and implementation of microservices to bring rain information into digital maps for car navigation systems.

2. DATA

In this paper, we propose to use precipitation data about the current precipitation situation from external weather data providers. The precipitation information is converted to a format that can be digested by the map overlay system in the car, adapted for optimal visualization, and delivered to all vehicles that are clients of the precipitation service.

2.1 Data Sources

Our implementation of the precipitation service is connected to two distinct, external data providers that deliver weather data. Due to privacy concerns we cannot disclose the names of these external companies. The flexibility of our proposed solution enables the usage of two providers using totally different data formats: Raster tiles JSON files and GeoTIFF images. This also allows us to compare the data quality of these two data sources. We can obtain further weather data from our external providers, such as air temperature, humidity, precipitation, and the like. However, for our rain map feature, we focus only on the *intensity of precipitation* value, the *iop* value. From our external data providers we obtain nowcasting data every 15 minutes, i.e. short-range forecast based on hourly weather stations data, modern techniques and algorithms. Therefore, we obtain *iop* values which are measured in mm / 15 minutes. Note that this has a minimum value of $iop = 0.01$, which means a small drizzle, whereas $iop > 2.0$ would mean a heavy rain storm. Fig. 1 shows an example of raster tile data for Europe. The raster tiles we obtain from our provider use *QuadTiles*[3] which has become a standard tiling method and is being used by OpenStreetMap, BingMaps, among other web maps.

Fig. 2 shows a zoomed-in example of rain data in QuadTile format for Germany and East Frisia. Note the rounding effects in the raster rain information. These effects can be caused by algorithms that extrapolate data from different sources every

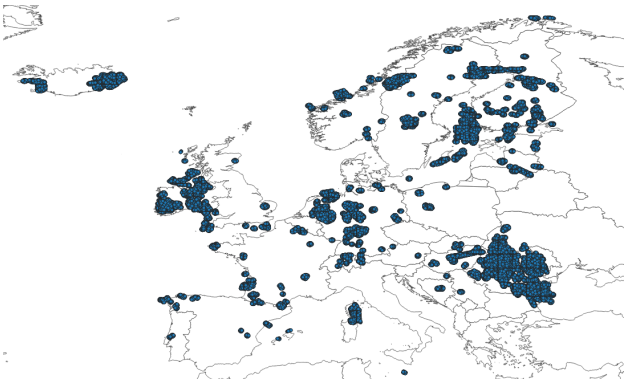

Figure 1. Example of rain data in QuadTile format for Europe.

15 minutes from hourly weather stations data. As clouds are moving over time, this produces an estimation with the shape of a probability distribution. In other words, rounded cloud effects may appear due to the mix of hourly weather station measurements plus algorithmic estimated rain probabilities every 15 minutes.

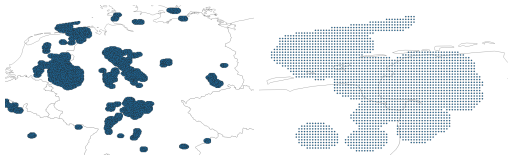

Figure 2. Zoomed-in examples of rain data in QuadTile format for Germany (left) and East Frisia (right).

In contrast, Fig. 3 shows rain information in a *GeoTIFF*[4] file color-coded with different tint shade blue gradients for different intensity precipitation levels. The GeoTIFF format is used by many GIS applications, where the GeoTIFF file is actually an ordinary TIFF image file with extra geographic metadata added in the header of the TIFF file (Ritter and Ruth, 1997). For comparison purposes, the size of the GeoTIFF file shown in Fig. 3 with rain data for Europe is of approx. 1.5 MB, whereas the size of the QuadTile JSON file used in Fig. 1 is of approx. 15 MB (zipped JSON file). Moreover, using QGIS 3.0, it was much faster to open and handle the data in GeoTIFF than in GeoJSON. This shows an advantage of using GeoTIFF image format instead of JSON format, especially for embedded systems. While GeoTIFF format is widely used for geographic information systems, QuadTile JSON is also frequently used, because of the general advantages of using JSON, like parsing customization, portability, easy integration and readability, among others. Furthermore, note that tools like *Rasterio*[5], provide diverse functionality to read and write GeoTIFF files and to use Python with N-dimensional arrays and GeoJSON. We considered both QuadTile JSON and GeoTIFF formats because we want to keep both easy handling and small size, resp., as well as the possibility to evaluate two different providers.

2.2 Data Output

Our precipitation service supports a number of different output data formats. The most relevant output format is the data format used by the map overlay system in the car. This map overlay system is a pre-existing software component built into

[2] https://arthurhou.pps.eosdis.nasa.gov/thorrelease.html
[3] https://wiki.openstreetmap.org/wiki/QuadTiles
[4] https://trac.osgeo.org/geotiff
[5] https://rasterio.readthedocs.io

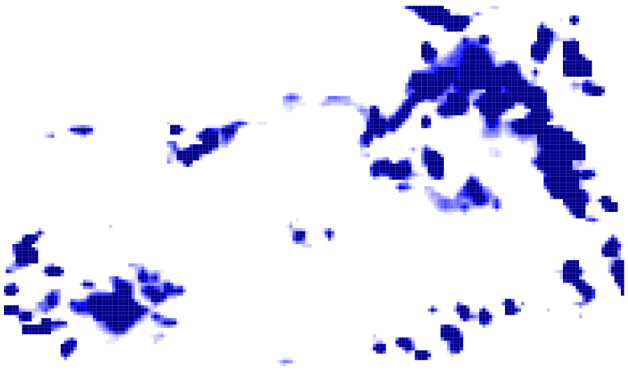

Figure 3. Example of rain data in GeoTIFF format for Europe

a large number of cars all over the world. Due to privacy reasons, we cannot disclose complete information about this overlay system. However, we can mention that it allows developers to add arbitrary objects, like images or geometric shapes, on top of the navigation map displayed on the screen. These overlay objects need to be defined in a JSON document with a specific structure. Our precipitation service can generate such map overlay JSONs, in particular, we focused on the following JSON formats: *LeafletJSON*, *GeoJSON* and *MOSJSON*. This latter is our special *"Map Overlay System"* JSON, which contains only the most relevant information for the car map system. While car navigation maps are our main target, we also enabled our solution to deliver precipitation data via apps and websites. For this purpose we defined a second data format, again in JSON, but specific to our precipitation use case. In comparison to the map overlay data format, which was designed to support a wide variety of map features, the clear focus in the second data format results in a much more concise - and also easy to understand - structure, which in turn decreases the amount of data that is needed to encode a given precipitation situation. As a third output format, the precipitation service also supports *GeoJSON*[6], an open standard for encoding geographical information. This allows our service to exchange data with a wide range of applications, for example to export and render precipitation data. In addition to these data formats meant for delivering data to other software systems, the precipitation service is capable of encoding precipitation data in *Leaflet*[7] format for faster validation using interactive web mapping, as well as, in binary formats that are used internally to exchange and cache information.

2.3 Data Preparation

Our precipitation system allows to modify precipitation data in different manners. When creating map overlay JSONs, one modification step that always needs to be done is splitting up precipitation areas that have holes in them. It might happen, for example, that it is raining in large parts of a given region, but there are few small areas within that region where there is no rain at all. In this case our system deals with a large precipitation area that has holes inside. While this case is handled properly by our system, the map overlay system in the car may not support shapes with holes. Therefore we split for each hole the surrounding shape into two parts, with the cutting line around the hole, so that we end up with two shapes that do not have a hole anymore, but together still exactly resemble the original precipitation area. Another modification step that we regularly apply for all output formats is binning. From our data sources we receive *iop* values on a fine-grained scale. However, for our users we want to distinguish only between few levels of rain. For this, we use the following classification for *iop* specified in mm per 15 min: (i) Light rain: *iop* is less than 0.5 mm; (ii) Normal rain: *iop* between 0.5 mm and 2 mm; and (iii) Heavy rain: *iop* is larger than 2 mm (MetOffice, 2012).

When dealing with precipitation data for large areas, like whole continents or even multiple continents at the same time, we observed performance issues, both in our precipitation service and in the clients rendering the output data. Therefore we introduced a number of optional modification steps that allow to reduce the amount of data. The first of these options restricts all data handling to a requested bounding box. This way we could set up several instances of the precipitation service in parallel, with each of them being responsible only for a certain area of the world, for example one service per country. Additionally, clients usually only show a small part of the world in their map viewer at a time. Thus it makes sense for them to dynamically request precipitation data only for the specific bounding box they are currently rendering. The second data reduction option is to define the outline of each precipitation area on a more coarse-grained level, omitting some details of its exact shape. This is particularly useful for clients which display precipitation for very large areas. In such cases, fine-grained details are usually not needed. Finally, we also included the option to set a hard limit on the output data size. With this option, all precipitation areas which would make the output bigger than allowed are simply discarded instead of being written to the output.

3. CLOUD APPEARANCE

In this section we review how to properly present different levels of precipitation. We discuss different options for the cloud appearance like color, shape and opacity, taking into account that the precipitation information in the navigation system should be visible, intuitive and useful to the driver.

3.1 Color Scheme

In infographics design, it has been shown that a rainbow gradient or heatmap which uses the full color spectrum may not be intuitive for the user. For example, it may not be clear for the user if a purple is a higher value than a deep red. Such situation may not be desirable for car navigation systems because it would need extra legends to explain the color meaning. Data mapped onto an alpha gradient of the same color would simplify interpreting values because more opaque areas will be perceived as stronger than more transparent areas. However, using transparent rain clouds may be problematic due to the underlying map mixing color with the rain clouds. This would result in brighter and darker areas with the same rain intensity. On the other hand, brighter colors are perceived as less intense than darker ones. Thus, we recommend using a simple brightness gradient to map the precipitation intensities to the rain clouds. To make the appearance of the rain clouds more pleasant, we added a limited hue gradient to the different intensities.

In this paper, we focus on precipitation only, however, note that to differentiate between rain and snow, we suggest to use diverging color gradients which will provide a complementary contrast. While the gradient for the rain goes from lighter blue to a dark blue to harmonize with the rest of the UI, the intensity of the snow could be mapped from light yellow to a dark red as it

[6] https://geojson.io/
[7] https://leafletjs.com/

is shown in Fig. 4. Moreover, hail could be shown with a pink color. Note also that in some countries there are conventions and customs to visualize rain information, while in Europe typically rain would be shown in blue, in the US for example, rain is typically visualized with yellow red color gradients.

Figure 4. Color scheme for rain (blue), snow (yellow-red) and hail (pink).

3.2 Cloud Shape

The main idea of polygon smoothing for rain clouds is to beautify the shapes of the polygons. Although it is possible to render complex cloud formations, the shape of the polygons should be as simple as possible for fast-processing and easy interpretation in car navigation systems. In general, whenever we process any cloud shape there can be loss of information. For example, the more smoothness we apply to a cloud, the more information will be lost from the original pixelized cloud. For this, Fig. 5 (top) shows different possible cloud shapes for displaying in a car-navigation system. We suggest to apply some smoothness but just enough to avoid pixelized clouds such that the minor loss of information does not affect the overall accuracy. Another advantage of using a more rounded and smoothed appearance for the clouds is, that they will stay in higher contrast to the mostly straight lines of the roads, ensuring their readability.

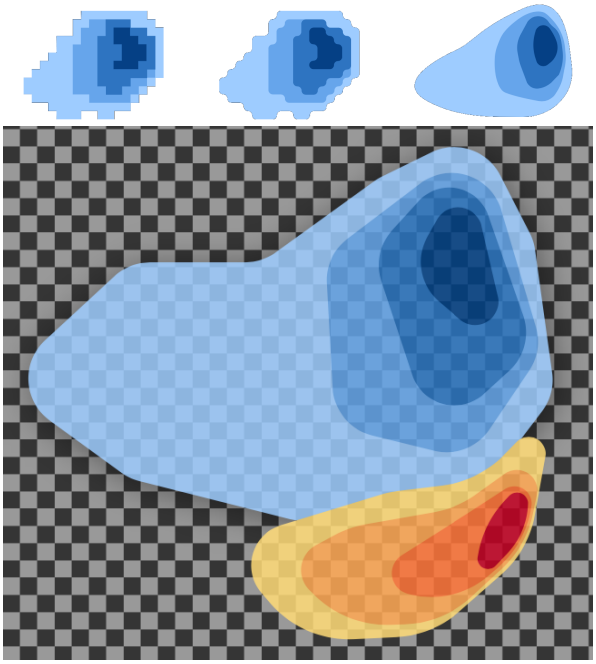

Figure 5. Top: cloud shapes for navigation systems: pixelated (left), rounded (middle) and smoothed (right). Bottom: example of rain cloud (blue) and snow (yellow red) shapes with 85% alpha transparency.

3.3 Cloud Opacity

We suggest to use clouds with a very low amount of transparency to reduce overloading the map. In this manner, the clouds will not be affected by the underlying map and they will be readable without distracting the driver. Note that the color of the clouds can be very close to important map elements like roads, lakes and rivers. To make these elements to appear, they should contain a border or drop shadow which should be active when the rain-map is active. We suggest to use 85% alpha as it is shown in Fig. 5 (bottom). Note that clouds are readable and information from background is still visible, where rain is represented in blue and snow in yellow red colors. If we decrease or increase the alpha value then clouds would be either not readable because the background would heavily influence them or readable but all further map information would be lost, respectively.

Fig. 6 shows an example of our design concept for rain cloud shapes for a car navigation system. Note the good readability of the information about the road map features, route navigation and precipitation. Moreover, blurring the clouds results in a overall reduced readability, as different intensities of rain can sometimes not be clearly distinguished. Therefore, we do not suggest to add blur nor borders to the rain clouds.

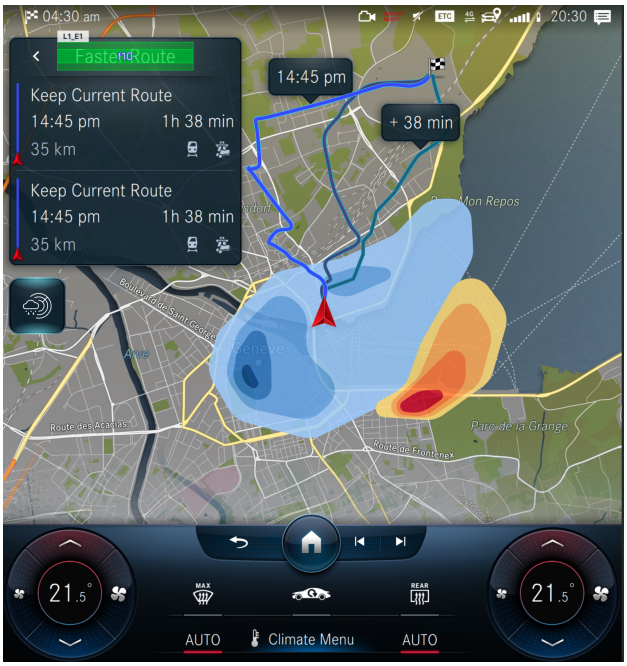

Figure 6. Example of rain cloud (blue) and snow (yellow red) shapes on a navigation system.

4. ARCHITECTURE

Fig. 7 shows the architecture of our system. On the right, the two precipitation data providers, for each one, there is a separate pipeline which imports and prepares the corresponding data from our providers. The first step of such a pipeline is an import process that is triggered every 15 minutes. It downloads the latest data from the respective data provider, converts it to our internal binary storage format and persists it, in a file named ”Provider Cache”. Once finished, the importer process triggers the output converter process, which reads the latest data from the provider cache and runs the corresponding data preparation steps described in Section 2.3. Specific parameters for the data preparation options are configurable when setting up a pipeline. After all configured intermediate transformations have been finished, the output converter process converts the precipitation data to the map overlay JSON format so that it can be consumed by the cars. The results are persisted in a file.

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-VIII-4-W1-2021-49-2021 | © Author(s) 2021. CC BY 4.0 License.

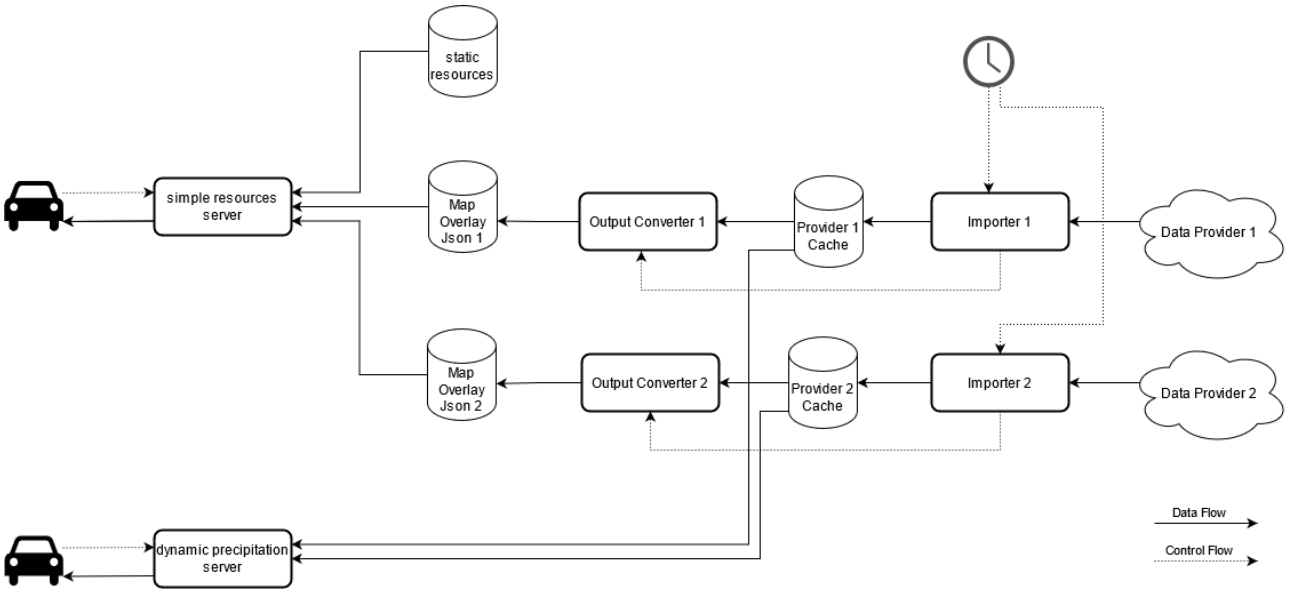

Figure 7. Microservice architecture of the rain map feature system.

Cars request the latest precipitation data via HTTP and a standard web server is used to serve the respective files. Since precipitation data is stored in simple files, common web servers can be used for this purposes. On the car, we let the user choose which data to display and this will be requested accordingly. Due to the design of the map overlay system, the car also needs to request some additional information. This data is independent of the actual precipitation data and contains style definitions, like the color of displayed polygons, borders, etc. These style definitions are static, so by putting them in a file, they can easily be served by the described web server.

The architecture described so far is built around the idea of precomputing the precipitation data that is sent to our cars. This approach allows for low latency and uses only little computing resources. However, it does not allow us to adapt precipitation data to particular requirements of each car. As outlined in Section 2.3, especially problematic is the large amount of data and rendering resources needed to update the full precipitation status on every car. For this reason, our architecture in Figure 7 also allows cars to request individually adapted precipitation data from an alternative server, called ”dynamic precipitation server”. This server reads precipitation data from the provider caches, adapts it as requested by its clients, and serves it to them. Supported adaptions include the conversion to any of the JSON formats mentioned in Section 2.2, selecting data for a given bounding box and discarding all data beyond a chosen output data limit as described in Section 2.3. In addition, the dynamic precipitation server allows the modification of the style definitions of the displayed precipitation areas dynamically. With both the standard web server and the dynamic precipitation server in place, we can compare both approaches and data providers with regards to latency, output data size, required computing resources, flexibility, etc.

5. IMPLEMENTATION

Fig. 8 shows the internal structure of our implemented application. There are two entry points to our application, *CommandLineMain* and *RainWebServer*. *CommandLineMain* class, this is the entry point for running the whole pipeline which: (i) imports precipitation data from a provider; (ii) performs any number of configured intermediate operations on it; and (iii) outputs it in a chosen format. While the data processing itself is handled in separate classes, *CommandLineMain* is responsible for handling command line parameters and setting up the data processing classes and their configuration accordingly.

Data processing is split as follows: importing data, buffering data, limiting data and exporting data. These steps are handled by a *Source* class, the *Buffer* class and a *Sink* class, respectively. *Source* and *Sink* have several sub-classes. They manage import and export settings, respectively, and implement common functionality, but leave the handling of specific input and output data formats to the sub-classes.

Source has a sub-class for each of our two data providers, capable of importing data in the specific format used by the respective provider. *Provider1Source* for example has the optional capability to import data for a specific bounding box. A third *Source*, *FileSource* is used for experimentation purposes. Our application and its various data processing options can easily be tested, for example, we added functionality to export/import precipitation data directly to/from a file, respectively. The importing from such a serialized file is the task of *FileSource*, and it supports several different serialization formats.

Source classes generate precipitation data for one location at a time. *Sink* classes on the other hand need a collection of all existing precipitation locations before they can start generating any output. For this reason, we added the *Buffer* class in between. It collects all precipitation data generated by a *Source* and forwards it to the - potentially multiple - connected *Sink*s only when the *Source* has finished. It is here where the option to impose a hard limit on the data size is implemented.

Sink has a sub-class for each supported output data format: *GeoJsonSink*, *LeafletJsonSink* and *MOSJsonSink* (where ”MOS” stands for ”Map Overlay System”). These classes contain logic for smoothing polygons, assigning colors to *iop* values, etc. There is also a *SerializedFileSink* for the serialization use case described above. In addition, there are several model

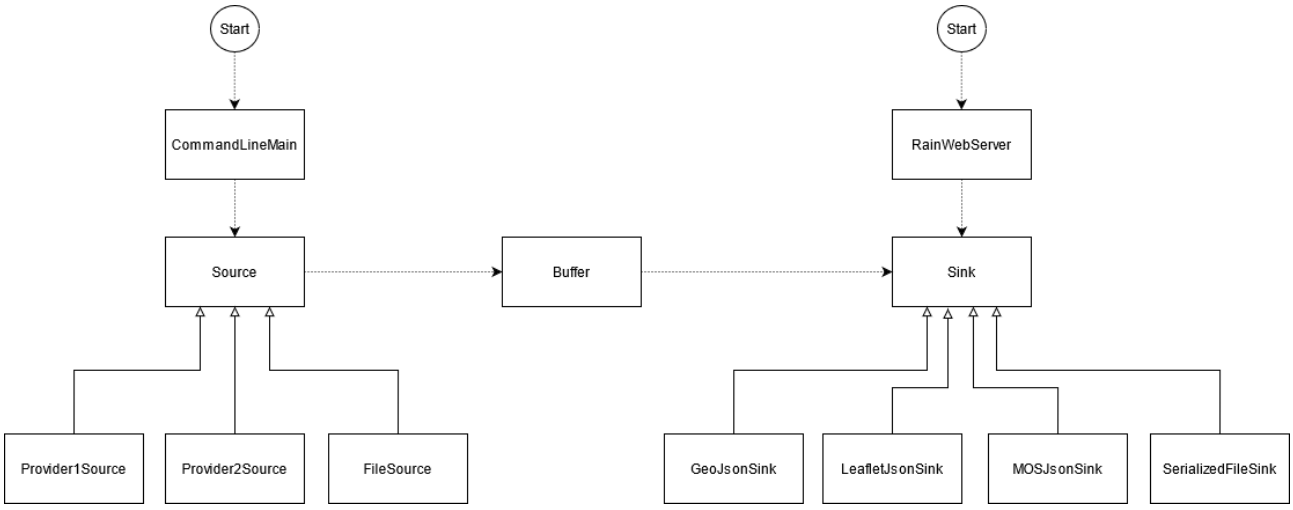

Figure 8. Internal structure of our implementation.

classes as well as helper classes for logging, serialization, etc. We rely on the *geotools*[8] library for some of the more advanced geospatial operations in our application.

The second entry point to our application is *RainWebServer*, a server that creates precipitation data on the fly as requested by its clients. To achieve low latency, this server does not make use of a full *Source* - *Buffer* - *Sink* chain. Instead, in terms of data input, it is specialized on reading serialized data from a file. In other words, the need for a *Buffer* is eliminated by reading the whole file before processing it. *RainWebServer* has its own, specialized functionality to limit data size and it also allows to limit output precipitation data to a configured or requested bounding box. For encoding the output, however, the server class makes use of some of the aforementioned classes, namely the three JSON-related *Sink* classes.

The system we implemented fetches precipitation data from external providers, whose names are not disclosed due to privacy concerns. The job of our backend components is to request this data periodically, convert it to polygons that can be digest by the navigation system in the motor vehicle and bring this polygons as a map overlay to the digital map. For this, we implemented the following steps: (i) fetch precipitation data; (ii) convert precipitation data to microservice format; and (iii) delivery of precipitation data to the car. To improve performance, note that the server should dynamically select precipitation data on demand for a requested bounding box only

Fig. 9 shows our current implementation of the rain map polygons in the car navigation system in 2D and 3D respectively. In our implementation we used Java together with GDAL libraries to read GeoTIFF files. Another option is to use *Rasterio* which provides a much developer friendly experience and performs just as fast as GDAL's Python bindings.

5.1 UI Integration

The rain map is a feature that should be easy and fast to activate by the car driver. Especially in regions of the world where weather can change very fast, it is needed that the user can quickly check if there will be heavy rain on the road ahead. Therefore, the rain map feature should be easy to access.

[8] https://geotools.org/

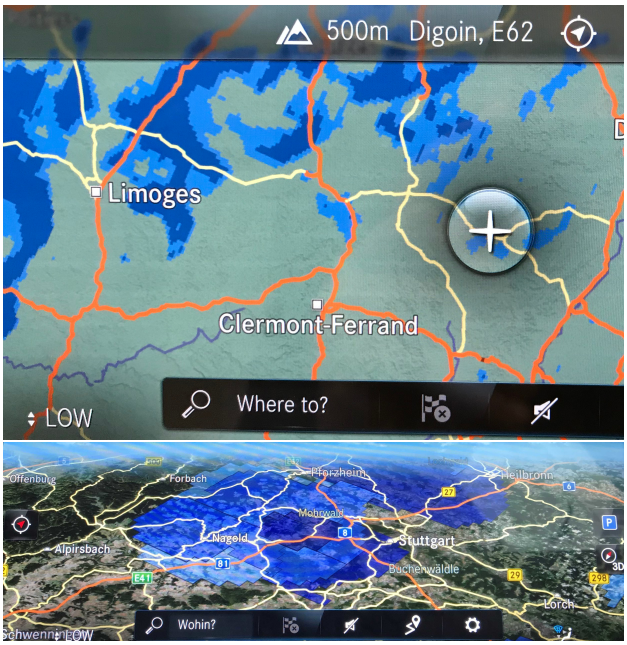

Figure 9. Implementation of our rain map feature on a car navigation system: (top) 2D and (bottom) 3D views.

Ideally it should be activated from the top level of the navigation interface, without the need to tap or scroll through menus. Therefore, a button to activate the rain map feature could be placed directly on the navigation interface. This button can be used to communicate the status of the feature too, i.e. if it is activated or not, if data is available or not, etc. For this, a glowing border and a brighter, more saturated color can tell the active state and a rotating throbber can indicate whenever the feature service needs to fetch data to update the rain map. If the feature is activated through a menu, an icon on the main interface should communicate an active state. In both cases, the feature should also be accessible using speech with some command like "show me the rain map" or "show me where it rains". If there is no service available, or an error occurs while fetching data, a second icon should be displayed communicating to the user, that the rain map is activated but not functional. Ideally, a user research study should be conducted to assess the acceptance of

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-VIII-4-W1-2021-49-2021 | © Author(s) 2021. CC BY 4.0 License.

the feature, i.e. to determine if a legend is necessary and to determine also the level of extra information detail needed by the user. Note that the appearance and location of the button has to be in line with UI design guidelines of the specific platform.

There may be a problem regarding the activation of the rain map on higher zoom levels. It may be possible, that a cloud will overlay the whole screen. Without a visible polygon shape, the blue region will not be recognized as a cloud as it would not provide any context to the user. Therefore, it is necessary to hide the cloud polygon in higher zoom levels and substitute this with an icon indicating the intensity of the rain. The intensity can be communicated with different types of icons, each one with different amount of rain drops and colored shape indicating the intensity of the rain according to the used color gradient. Another option is to just hide the cloud and to not show any further information, just as pedestrian and private roads are not shown at lower zoom levels.

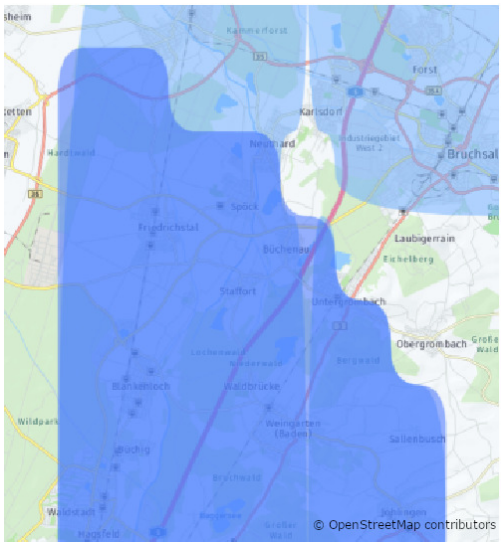

Figure 10. Smoothing problem when over-posing cloud layers.

6. CONCLUSIONS AND FURTHER WORK

In this paper, we propose a system to integrate rain information as a map feature for motor vehicle navigation systems. We discussed the usage of two different input data formats: QuadTile JSON files and GeoTIFF images. And the output data formats: GeoJSON, LeafletJSON and MOSJSON with the most relevant information for the map overlay system. We also discussed different important properties for the rain cloud appearance in a navigation system. We presented in detail our microservices architecture, and discussed some important issues when dealing with data pipelines and implementation of such systems in cloud and embedded systems.

Further work includes to improve the shape of the polygons. Our current polygons have perpendicular sides (pixelated). In Fig. 10, we show some problems we found when applying smoothing to our pixeled polygons. One problem occurs when placing polygons on top of each other, which leads to unrealistic cracks or fissures in the clouds. Further work includes further testing of algorithms to find the most suitable algorithms that generate polygons leading to smooth rain clouds for the navigation system. For this, several libraries can be found, for example *psimpl* or *boost: simplify*, both libraries for polyline simplification using Douglas-Peucker algorithm (Douglas and Peucker, 1973). A different promising approach is the use of *k-means* or *DBSCAN* clustering algorithms to locate clusters of points, i.e. rain clouds that belong together and as a second step a convex hull algorithm to find a coarse polygon that covers all points in the cloud (Ada et al., 2018).

In general, there is a trade-off on using large contour polygons over small size polygons for visualizing the rain clouds in motor vehicle navigation systems. The advantage to use large contour polygons is that this leads to short JSON file size, which means higher performance, and fast processing of the rain map feature in the navigation systems. The disadvantage, however, is that the shape of some polygons may be too generic, falsifying rain information, i.e. showing rain clouds where there is no rain at all, or on the contrary, showing no rain where it is actually raining. How to deal with these too generic polygons? We can either prevent their formation adding some extra logic, or we can fix them by adding some extra computations to dissect these polygons. If possible, the former would be a much better approach. On the other hand, when we use small size polygons, the advantage is that of simplicity, no extra computations are needed to determine the most proper rain cloud polygon shapes. Another advantage is that pixeled clouds are already familiar for the end users, as they already know the meaning of these rain clouds and shapes from mobile or desktop rain radar applications. Thus, drivers would immediately recognize them and understand their meaning when they see them for the first time in the car navigation system. However, the disadvantage is that the performance of the navigation system would decrease, the larger the JSON file, the larger the CPU computation in the embedded system, where resources are usually scarce.

Further work also includes presenting precipitation in 3D in a much convenient form as it is shown in Fig. 9 (bottom). One suggestion is to render the clouds in the 3D view on a common height above the horizon as shown in Fig. 11. Atmospheric perspective is used to minimize visual clutter due to clouds in the background and in order to give the user a reference to estimate the distance to the clouds. The shape of the clouds in Fig. 11 is the same as in the top-down view in Fig.6, they are only rendered as flat polygons. A polygon could be drawn down from the cloud to give a better understanding of the area where the rain is present. An alpha gradient can be applied to the clouds to avoid covering objects behind the polygons. It would be also useful to investigate how to differentiate intensities of blue in 3D. For this, some frameworks are available for web-based real-time 3D visualization of large-scale weather radar data using 3D tiles and WebGIS technology (Lu et al., 2021). This 3D tiles technology is really promising as it is an open specification for online streaming 3D geospatial datasets with high rendering performance and low memory consumption.

Further work is also to validate and integrate information from different sources, assessing the best accurate data to provide the most reliable information to the final user. This includes data that can also come directly from the fleet of vehicles that are clients of the service. This approach of harvesting precipitation data from client vehicles was not part of our current implementation and it is left as further work. Nevertheless, each individual vehicle on the street can sense for raining - by checking the status of the windshield wipers, wipers speed, etc. Therefore, some information could flow from the car to the backend to update the rain nowcasting. And since cars are also equipped with GPS, our precipitation service could be extended to com-

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-VIII-4-W1-2021-49-2021 | © Author(s) 2021. CC BY 4.0 License.

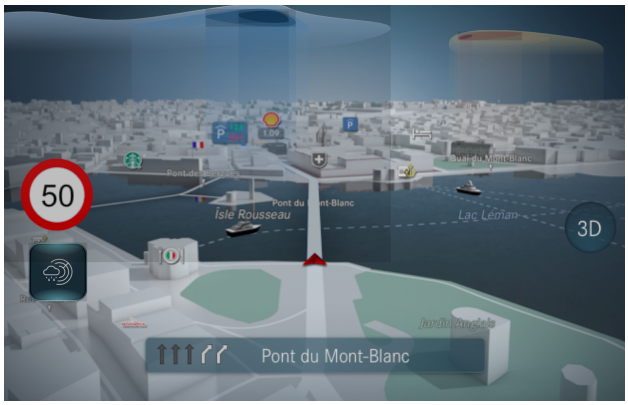

Figure 11. Design concept for rain map feature in 3D.

pute a precipitation map based on the collected data. Note also that some countries provide also free weather information, for example, in Germany the Deutscher Wetterdienst *(DWD)*[9], through the CDC (Climate Data Center) they offer free access to many climate data of the DWD. The service we propose in this paper is designed in such a flexible way that allows different data sources - as demonstrated already with the existing providers - so that integrating this computed map as another data source is already enabled by our concept.

Some extra features (for example to show information on average and variance of rain precipitation) can be also included in a navigation route. Recent approaches have shown the possibility to predict dangerous roads under certain weather conditions (Reichenbach and Navarro-B., 2021). This approach could be integrated in the system proposed in this paper to help warn the driver not only about coming rain storms and severe weather situations, but also roads and junctions that are particularly dangerous during heavy precipitation. Finally, it is feasible to include animated rain by showing in which direction the rain is moving. The main problem for an animated feature is safety, the driver could be distracted or annoyed by the animation.

ACKNOWLEDGMENT

Thanks to Frank Bielig (MBition GmbH) who sparked the initial idea for this project, his continuous support and feedback. Thanks also to Iris Dannenmann (Daimler AG) for her support during the last part of the project.

REFERENCES

Ada, N., Harsono, T., Basuki, A., 2018. Cloud satellite image segmentation using meng hee heng k-means and dbscan clustering. *2018 International Electronics Symposium on Knowledge Creation and Intelligent Computing (IES-KCIC)*, 367–371.

Brewer, C., Hatchard, G., Harrower, M., 2003. ColorBrewer in print: a catalog of color schemes for maps. *Cartography and geographic information science*, 30(1), 5–32.

Curzon, P., Blandford, A., Butterworth, R., Bhogal, R., 2002. Interaction design issues for car navigation systems. *Proceedings Volume 2 of the 16th British HCI Conference London, September*, Springer Verlag.

Douglas, D. H., Peucker, T. K., 1973. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The Int. Journal for Geographic Information and Geovisualization*, 10(2), 112-122.

Hu, H., 2014. An algorithm for converting weather radar data into GIS polygons and its application in severe weather warning systems. *International Journal of Geographical Information Science*, 28(9), 1765–1780.

Ito, Sadanori, Koji, Zettsu, 2020. Assessing a risk-avoidance navigation system based on localized torrential rain data. *MATEC Web Conf.*, 308(03006).

Kelley, O. A., 2013. Adapting an existing visualization application for browser-based deployment: A case study from the Tropical Rainfall Measuring Mission. *Computers & Geosciences*, 51, 228-237.

Kilpeläinen, M., Summala, H., 2007. Effects of weather and weather forecasts on driver behaviour. *Transportation Research. Part F: Traffic Psychology and Behaviour*, 10, 288–299.

Kisters, P., Bade, D., Wulk, J., 2019. Dynamic routing using precipitation data. *2019 Fourth International Conference on Fog and Mobile Edge Computing (FMEC)*, 312–317.

Krajeswki, W. F., Smith, J. A., 2001. Radar hydrology: rainfall estimation. *Advances in Water Resources*, 25, 1387–1394.

Kyte, M., Khatib, Z., Shannon, P., Kitchener, F., 2000. Effect of environmental factors on free-flow speed. *Fourth International Symposium on Highway Capacity*, Citeseer, 108–119.

Lu, M., Wang, X., Liu, X., Chen, M., et al, 2021. Web-based real-time visualization of large-scale weather radar data using 3D tiles. *Transactions in GIS*, 25(1), 25-43.

Mantas, V., Liu, Z., Pereira, A., 2015. A web service and android application for the distribution of rainfall estimates and Earth observation data. *Computers & Geosciences*, 77, 66-76.

MetOffice, 2012. Fact sheet 3 — water in the atmosphere. Technical report, MetOffice, UK. Crown copyright 2012 12/0151. Retrieved 2021-03-14.

Rasp, S., Dueben, P. D., Scher, S., Weyn, J. A., Mouatadid, S., Thuerey, N., 2020. WeatherBench: A Benchmark Data Set for Data-Driven Weather Forecasting. *Journal of Advances in Modeling Earth Systems*, 12(11), e2020MS002203. e2020MS002203 10.1029/2020MS002203.

Reichenbach, A., Navarro-B., J.-E., 2021. A model for traffic incident prediction using emergency braking data. *to appear in 32nd IEEE Intelligent Vehicles Symposium*.

Ritter, N., Ruth, M., 1997. The GeoTIFF data interchange standard for raster geographic images. *International Journal of Remote Sensing*, 18(7), 1637–1647.

Stauffer, R., Mayr, G., Dabernig, M., Zeileis, A., 2015. Somewhere over the rainbow: How to make effective use of colors in meteorological visualizations. *Bulletin of the American Meteorological Society*, 96(2), 203–216.

Tong, C., Schroeder de Witt, C., Zantedeschi, V., et al., 2020. Enabling data-driven precipitation forecasting on a global scale. *Tackling Climate Change with Machine Learning*, 1–10.

[9] https://www.dwd.de/