ISSN 2464-4617 (print)
ISSN 2464-4625 (CD)
Computer Science Research Notes
CSRN 2802
Short Papers Proceedings
http://www.WSCG.eu

Service-based Processing and Provisioning of Image-Abstraction Techniques

M. Richter[1] marvin.richter@hpi.de
M. Söchting[1] maximilian.soechting@student.hpi.de
A. Semmo[1] amir.semmo@hpi.de
J. Döllner[1] doellner@hpi.de
M. Trapp[1] trapp@hpi.de
[1]Hasso Plattner Institute, Faculty of Digital Engineering, University of Potsdam, Germany

ABSTRACT

Digital images and image streams represent two major categories of media captured, delivered, and shared on the Web. Techniques for their analysis, classification, and processing are fundamental building blocks in today's digital media applications ranging from mobile image transformation apps to professional digital production suites. To efficiently process such digital media (1) independent of hardware requirements, (2) at different data complexity scales, while (3) yielding high-quality results, poses several challenges for software frameworks and hardware systems, in particular for mobile devices. With respect to these aspects, using service-based architectures is a common approach to strive for. However, unlike geodata, there is currently no standard approach for service definition, implementation, and orchestration in the domain of digital images and videos. This paper presents an approach for service-based image-processing and provisioning of processing techniques by the example of image-abstraction techniques. The generality and feasibility of the proposed system is demonstrated by different client applications that have been implemented for the Android operating system, for Google's G-Suite Software-as-a-Service infrastructure, as well as for desktop systems. The performance of the system is discussed at the example of complex, resource-intensive image-abstraction techniques, such as watercolor rendering.

Keywords

Image-processing techniques, service-based processing, service-based provisioning

1 INTRODUCTION

For many years, digital images have been usually captured by means of mobile devices, such as smartphones and digital cameras, and have been widely distributed by a number of different media channels. Especially on the Web, with the emergence of social media and image platforms (e.g., Instagram), the amount of digital images increases dramatically. In these contexts, techniques for image analysis, classification, and processing are required, e.g., to optimize image search engines, to develop image transformation software for the application of image-abstraction techniques [12], and to produce artistic effects, such as style transfers [18].

With respect to image transformations on mobile devices, two technical alternatives are predominant: *on-device* and *off-device* processing. On-device processing utilizes the device hardware (e.g., CPU and GPU) to transform images and, thus, does not depend on a remote processing server [2], resulting in three significant benefits: (1) processing is usually faster, since there is no round-trip-time to a server (*performance*), (2) if images contain personal information, they are not intended for sharing via the network (*privacy*), and (3) although network communication has become prevalent, it is not guaranteed to establish network connections, which especially applies to mobile devices using mobile networks (*reliability*).

However, on-device image-processing has also a number of limitations. The design, implementation, testing, deployment, and maintenance of hardware-accelerated software for on-device processing is a cost-intensive process in terms of development-time and resources due to the high diversity of devices. A multitude of platforms (e.g., iOS, Android, Windows, macOS, Linux) with different graphics Application Programming Interfaces (APIs) (e.g., OpenGL, OpenCL, Direct3D, CUDA) in different versions has to be supported.

In particular, performing image-processing tasks on mobile devices have the following implications: on the mobile Android market a huge device heterogeneity is ubiquitous, often implying different capabilities and hardware specifications. Technical evaluations of an on-device image-processing approach on Android devices revealed huge performance differences between

Figure 1: Application of an image-abstraction technique (oil paint) to the same image on different platforms: Web-based Google Office Suite (A), mobile Android app (B), and desktop client (C).

devices with similar hardware specifications [2]. Also, mobile devices are limited with respect to memory available, computational power, and battery life, which is a crucial limitation, since image-processing is usually an intense resource consuming process. Further, software developer have to take various inputs via device sensors as well as device orientation and display resolutions into account. Moreover, small displays often lead to screen real estate problems. Furthermore, Digital Rights Management (DRM) for on-device solutions cannot be guaranteed, since processing relevant data (e.g., textures, shaders) is accessible via API-call interceptions and, thus, the control and protection of intellectual property gets lost.

Service-based Image-Processing.

With respect to the on-device image-processing challenges stated above, an off-device service-based image-processing approach offers a practical alternative. For this purpose a Service-Oriented Architecture (SOA) and its predominant design patterns is used [3]. It facilitates development of reusable components for processing images without the downsides of dealing with a wide range of devices. This allows a simple integration of those components into existing services. Thus, various heterogeneous clients can be attached to provide hardware independent cross-platform solutions via a common Representational State Transfer (REST) interface [6]. For example, Figure 1 shows the application of a complex oilpaint effect [19], a typical resource-intensive artistic stylization technique, on different platforms, such as web-based Google Office Suite add-ons, mobile Android apps, and desktop applications that interact with the proposed service-based image-processing server.

The approach of service-based processing offloads computation-intensive image-processing tasks (e.g., processing of stylization effects that rely on multi-stage processes) to a server infrastructure, equipped with more resources in terms of memory, computation power, and energy supply. Thus, this approach dramatically reduces energy consumption especially for mobile devices. Further, an important requirement for future image transformation processes is the capability to deal with continuously increasing *spatial* and *temporal* resolution of visual media (>20 megapixel at 120 frames-per-second) and to guarantee high quality output images. A service-based approach tackles the complexity of scalable and high-resolution image space. The applicability of the service-based approach and its performance highly relies on the bandwidth of the used network for data transmission. However, due to constantly evolving network infrastructure in terms of increasing bandwidth, it is an acceptable constraint.

Contributions. To summarize, this paper makes the following contributions:

1. It presents a concept for service-based off-device image-processing, following the orchestration pattern for service composition. Exemplarily, the creation of a high-level service based on atomic low-level services is shown as an example.

2. A platform-independent representation of image-processing effects is formulated for highly customizable configuration of the image-processing. Furthermore, a approach for storing and provisioning of these effect representations is presented.

3. It demonstrates the approach using different application examples, such as image manipulation on Android and desktop systems, and the integration into the Google Office Suite for various image-abstraction techniques (e.g., oilpaint, watercolor).

The remainder of this paper is structured as follows. Section 2 reviews related work on service-based approaches to image-processing and applications. Section 3 presents our concept for service-based provisioning of image-abstraction techniques and explains implementation details. Section 4 shows common application examples and Section 5 discusses our approach and states potential future research. Finally, Section 6 concludes this paper.

ISBN 978-80-86943-41-1

2 RELATED WORK

This section covers related work in the field of service-based image-processing and approaches for image-abstraction techniques.

Service-based Image-Processing. Several software architectural patterns are feasible for implementing service-based image-processing. One prominent style of building a web-based processing system for any data is the service-oriented architecture [22]. This approach allows server developers to set up a multitude of processing endpoints, each providing a specific functionality and covering a different use case. These endpoints appear as a single entity to the client, i.e. the implementation stays hidden for the requesting clients, but can be realised through an arbitrary number of self-contained services. This work follows the service-oriented architecture as described in Section 3.

Since web services are usually designed to maximize their reusability, their functionality should be simple and atomic. Therefore, the composition of services is critical for fulfilling more complex use cases [14]. The two most promiment patterns for realising this composition are choreography and orchestration. The choreography pattern describes decentralized collaboration directly between modules without a central component. The orchestration pattern describes collaboration through a central module, which triggers the different web services and passes the intermediate results between them. In this work, the orchestration pattern is implemented as described in Section 3.

In the field of image analysis, Wursch et al. [26] developed a web-based tool that enables users to perform different image analysis methods, such as text line extraction, binarization, and layout analysis. The tool is realised through a set of REST web services. Application examples in that work include multiple web-based applications for different use cases.

The viability of implementing image-processing web services using REST has been demonstrated by Winkler et al. [24], including the ease of combination of endpoints. Another example for service-based image-processing is Leadtools (https://www.leadtools.com), which provides a fixed set of around 200 image-processing functions with a fixed configuration set via a web API. In this work, however, a similar approach using REST is chosen, although with a different focus in terms of granularity of services. While Winkler and Leadtools focus on fixed endpoints for a selected number of image-processing effects, this work aims for a general-purpose image-processing system based on an platform-indepedent effect format (Subsection 3.1).

In the field of geodata, the Open Geospatial Consortium (OGC) set standards for a complete server-client ecosystem. As part of this specification, different web services for geodata are introduced [15]. Each web service is defined through specific input and output data and the ability to self-describe its functionality. In contrast, in the domain of general image-processing there is no such standardization yet. However, it is possible to transfer concepts from the OGC standard, such as unified data models. These data models are realised through a platform-independent effect format. In the future, it is possible to transfer even more concepts set by the OGC to the general image-processing domain, such as the standardized self-description of services.

Image-Abstraction Techniques. In this work, we focus on edge-aware and content-preserving image-processing as a fundamental tool in computational photography and non-photorealistic rendering for abstraction and artistic stylization. Typical approaches that operate in the spatial domain for abstraction use a kind of anisotropic diffusion [17, 23] and are designed for parallel execution, such as approximated by the bilateral filter [21] and guided filter [9]. A plentitude of stylization techniques exist using these filters as building blocks to simulate traditional painting media and effects [13], such as cartoon [25] and oil paint [20]. However, these may become computationally expensive when applied in an iterative multi-stage process. This particularly applies to techniques using global optimizations to separate detail from base information, e.g., based on weighted least squares [4] or locally weighted histograms [11], and recent techniques that separate style from content using neural networks [7]. Because of their global optimization scheme, they are typically not suited for real-time application, in particular not on mobile devices. To this end, we implemented a variety of these techniques using the proposed image-processing service including stylization, HDR tone mapping and compression, JPEG artifact removal and colorization, to demonstrate its versatile application.

3 SERVICE-BASED PROCESSING

Figure 2 shows a conceptual overview of the components of the image-processing system. It basically comprises the following components, which are described in the remainder of this section in greater detail:

Effect Service: This service component is responsible for storing all resources required by the image-processing service. It delivers platform-independent representations of image-processing effects based on an Extensible Markup Language (XML) format bundled with Graphics Processing Unit (GPU) shader programs and textures for dedicated target platforms, i.e., GPU hardware and API. This service can be utilized for different use cases in addition to the one described in this work, e.g., delivering platform-independent image-processing effects directly to user clients for on-device rendering [2].

ISBN 978-80-86943-41-1

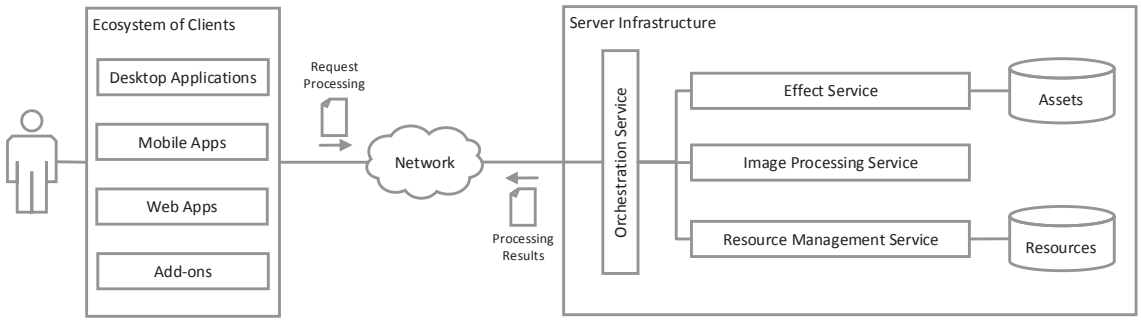

Figure 2: Components of the proposed service-based image-processing system. A diverse range of clients can request the image-processing functionality via the interface of the orchestration service, which coordinates services for effect provisioning (effect service), image-abstraction and stylization (image-processing service), and storing the respective data, e.g., images, (resource management service).

Image-Processing Service: This service performs the actual processing of images with respect to image analysis, abstraction, and stylization. It receives an input image, an effect reference, and a processing configuration (e.g., output file format, output quality etc.) and delivers a processing result, which, for example, can be a stylized image or an analysis result (e.g., dominant color, histogram).

Resource Management Service: The resource management service is responsible for storing and provision of data, such as images and its metadata. Along with the provision of resources this service also provides low-level image manipulation functionality, such as cropping, resizing, and rotating.

Orchestration Service: A service-based system consisting of various service components has to tackle the crucial challenge of service composition. The composition of services aims at creating a composite service that combines various atomic functional building blocks provided by the available services in order to satisfy a higher level use case. The composition is managed by the orchestration service. Further, the orchestration service provides the public service endpoints that can be accessed by the clients.

3.1 Effect Representation and Service

The effect service is responsible for delivering platform-independent descriptions of image-processing effects to requesting clients, such as the image-processing service. The delivered image-processing effects are composed of multiple assets, which depend on each other. Each asset is described in an asset format. This asset format and the dependency structure is based on the work of Duerschmid et al. [2]. Examples for assets are implementation-specific files such as shaders and textures. The effect service is realised through a Node.js JavaScript application, which provides the web service interface, in conjunction with PostgreSQL, which is used for storing asset meta data and asset files.

Assets are sets of files that, once composed, define a platform-independent, executable description of an image-processing effect. Assets are designed to strictly separate between platform-independent parts and platform-dependent parts. The composition of assets is realised through inter-asset dependencies. Once an image-processing operation is requested, the effect service resolves the dependencies and bundles all assets together, resulting in an executable asset bundle. This minimizes the required client effort to download a given image-processing effect.

3.2 Image-Processing Service

The image-processing service is responsible for executing image-processing effects. The service implementation uses a cross-platform C++ image and video processing framework designed for desktop and server systems. To enable efficient processing, graphic acceleration supporting multiple modern graphic APIs (e.g., OpenGL, Vulkan) is used. Per request, the image-processing service takes the reference to an image as input and can be configured with respect to the following configurations:

Effect File: Reference to the parameters of the desired effect. The effect's XML file is parsed and a processing pipeline is instantiated based on the respective effect configuration.

Preset Identifier: The effect can be configured using so-called presets. Setting a preset identifier applies a predefined parameter configuration that is basically a list of parameter name and value tuples.

Output File Format: To reduce streaming bandwidth and the size of transmitted data, the image-processing service can generate output images in

 ISBN 978-80-86943-41-1

compressed, lossy formats (e.g., JPEG) and lossless formats (e.g., PNG). The compression setting can be configured by the client according to the specific use case.

Output File Quality: The quality parameter configures the JPEG quality and the PNG compression rate, respectively. The quality factor of the output image must be in the range 0 to 100. As for PNG export, specify 0 to obtain small compressed files and 100 for large uncompressed files. The format and the respective quality configurations results from a trade-off between output quality and transmission time. Those are crucial adjustments to achieve a responsive user interaction within the client application.

Return Type: The return type determines, if an *image token* or the processed *image* is returned by the service. Choosing an image token reduces bandwidth while applying multiple effects and/or different effect presets, since the image is kept on the server and does not need to be transmitted for every single request.

3.3 Resource Management Service

Two types of resources are associated with the resource management service: *image* and *image-related* resources. Supported image file formats are the ubiquitous MIME types JPEG and PNG. The support for further file formats can be easily integrated. Every image resource can be identified using a Universal Resource Identifier (URI). The image information includes both technical properties about the image (e.g., image resolution) and low-level analysis results such as pixel edge-count, dominant colors, or a color histogram. In addition to the management of resources and their metadata, the resource management service provides low-level image manipulation functionality (e.g., rotating, cropping, resizing), which can be used by image-viewer clients that require zoom, pan, or rotate functionality.

3.4 Orchestration Service

The orchestration service follows the facade design pattern—in the context of service-based architectures also denoted as API gateway—and provides the public service-endpoints that handles requests sent by the clients. Requests that require atomic functionality, which is provided by a single service, are dispatched to the respective service (e.g., provision of an image, update of an effect). More complex requests (e.g., processing of an image with a specific effect) that depend on several services to fulfill a high-level use cases are managed by the orchestration service. The orchestration service calls and coordinates required services, manages the execution flow, and assembles information required for the response. A workflow example for processing an image that uses the orchestration of the three main services, is outlined in the following:

1. Fetch image from resource management service.
2. Fetch the requested effect from the effect service.
3. Request the processing of the fetched image with the requested effect at the image-processing service.
4. Gather and deliver relevant status/error information and the processing results.

Further, the orchestration service delivers its capabilities using the OpenAPI specification (https://www.openapis.org/). Thus, the service endpoints are both human and machine readable and can be comprehended without reading source code or documentations.

In technical terms, communication between the services is based on RESTful HTTP [5] – which is one of the established standards, along with SOAP [8] and message oriented middleware [1] – for service-based architectures [16]. REST facilitates the communication within heterogeneous environments comprising services running on different machines or in different execution environments. The REST services accept HTTP requests to a URI with a specific HTTP method (PUT, GET, POST, DELETE). The URI for requesting the orchestration service complies the following template:

```
https://<IP>/<resource>/<id>/<action>
```

Here, the parameter `<IP>` refers to the location of the server, `<resource>` indicates the type of the resource (e.g., image), `<id>` specifies the identifier of the resource, and `<action>` declares the type of processing, (e.g., transform, analysis, or info). Additionally, HTTP methods are used to map CRUD (create, read, update, delete) operations to HTTP requests. Hence, a resource can be created or updated (POST), fetched (GET), and deleted (DELETE). To get or delete a resource, the action path parameter can be omitted. A request to retrieve the image information of a specific image with the identifier 12345 is the following GET request:

```
https://<IP>/image/12345/info
```

4 APPLICATION EXAMPLES

This section demonstrates the applicability of the presented concept to various application domains, such as (web-based) add-ons for office products (Subsection 4.1), image manipulation on mobile devices (Subsection 4.2), and desktop systems (Subsection 4.3).

 ISBN 978-80-86943-41-1

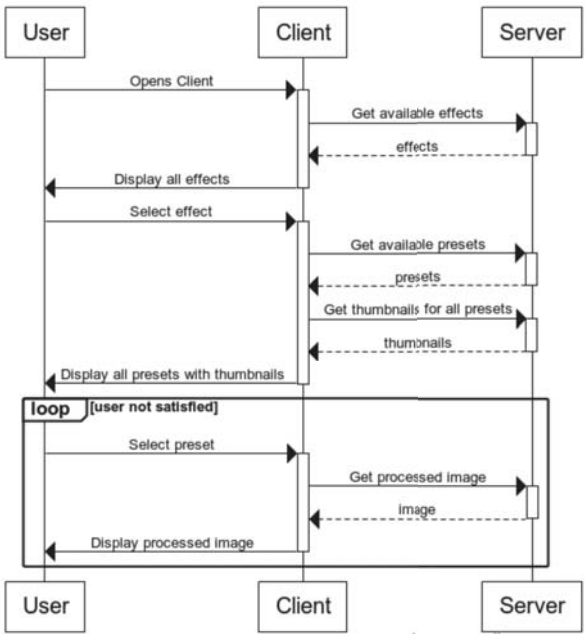

Figure 3: The common workflow of all clients communicating with the web services. From top to bottom: Initially, all effects are retrieved and displayed to the user. The user then has the option to try out different effect and preset combinations until he is satisfied.

4.1 Google Office Suite Integration

Google offers various web apps as part of its office suite (G-Suite): Sheets, Docs and Slides. Each of these Google Apps offers add-on integration through the Google Apps Script platform. As a demonstration for the integration of service-based image-abstraction techniques, add-ons for Google Sheets, Docs and Slides that utilize the presented image-processing web service of this work were developed (Figure 4).

The workflow of the add-on, which is common for all application examples of this chapter, is shown in Figure 3. First, the client requests a list of all currently available effects from the effect service. These are displayed to the user in a list. Once the user selects an effect, the currently selected image is cropped and processed multiple times to generate previews for the different effect presets. These dynamic previews are displayed below the effect list, enabling users to get a first impression of the visual impact of the effects and its respective presets. Subsequent to selecting one of the previews, a full-resolution image is processed on the server and displayed to the user. This full-resolution image can then be inserted into the Google Slides, Docs, or Sheets document.

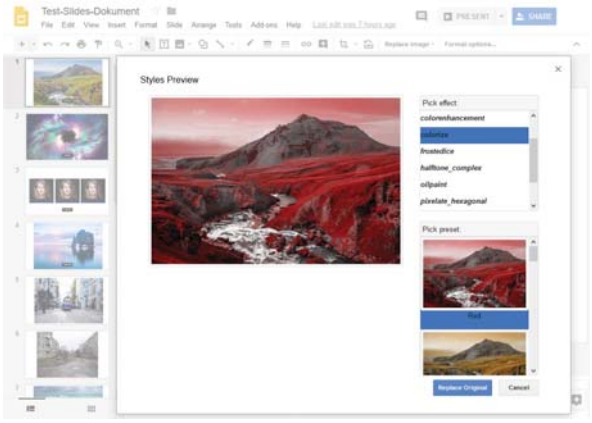

Figure 4: Example of the integration of service-based image-abstraction techniques into Google Slides via an Add-on using server-sided Google Script.

The add-ons are implemented using Google Script, a server-side scripting interface based on JavaScript, and templated HTML5 user interfaces. Figure 5 shows an overview of the basic add-on architecture and integration. Since every Google App has a different API for retrieving images from the document, each add-on requires to be adapted for the specific app. The core of all add-ons, the web-service connection library, encapsulates common functions required for calling the web services. Using this design, rapid development of add-ons for new and existing Google Apps is easily possible.

However, the add-on environment of Google Apps Script poses two limitations. First, the daily quota for regular users of HTTP requests is limited to 100 megabytes. A naive implementation that directly uploads and downloads processed images can reach this quota quickly. To counter-balance this, the add-on is designed to send links instead and exploit HTML image-embeddings whenever possible to circumvent the quota. A second limitation currently concerns the runtime performance of inserting new images into a document. In a Google Slides presentation example, inserting a 1.6 megapixel image takes 3.5 seconds. In contrast, uploading, processing and downloading the image takes only approx. 0.8 seconds in the same environment. Therefore, to achieve sufficient performance, the add-on is implemented to minimize these calls to Google APIs.

4.2 Android Image Manipulation App

Figure 6 shows screenshots from an Android mobile app that enables the application of various image-processing techniques to input images. This client offers similar features as ProsumerFX [2] but without on-device processing.

The workflow in this app is similar to the presented Google Office Suite add-on. At first, the client fetches a list of available image-processing techniques from the web services. Next, these effects are filtered and grouped based on delivered effect metadata, such as effect complexity and category. Once the user decides for an effect, dynamic previews for each predefined parameter configuration (preset) via thumbnail processing are

ISBN 978-80-86943-41-1

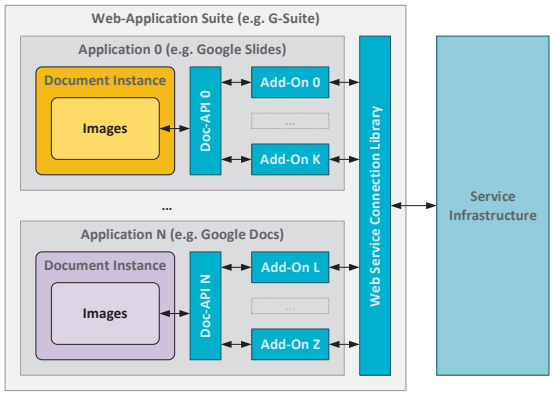

Figure 5: Architecture of the Google Office Suite add-ons. For every Google App, an arbitrary number of add-ons with a different set of available effects can be deployed. All add-ons use the web service connection library, which provides convenient access to the web services.

generated. This allows users to get a first impression of the visual impact of each preset and helps them in their decision making. When the user selects a preset, a high-quality version of the processed image will be generated on the server and then displayed in the client. The app also allows the combination of multiple effects, sequentially applying them to the image. The user can reorder these effects, taking control over the orchestration of the web services.

There are multiple advantages of the app in contrast to on-device processing. First, the processing is independent of the device graphics hardware since it is performed on a server machine. This allows weaker devices to apply complex image-processing effects to high-resolution images. Furthermore, the battery consumption is significantly lower than a comparable on-device rendering solution. The app can also be utilized in a business context as a white-label solution for different companies, i.e., multiple customized versions of the app with a specific brand, logo, and identity can be created and sold to companies.

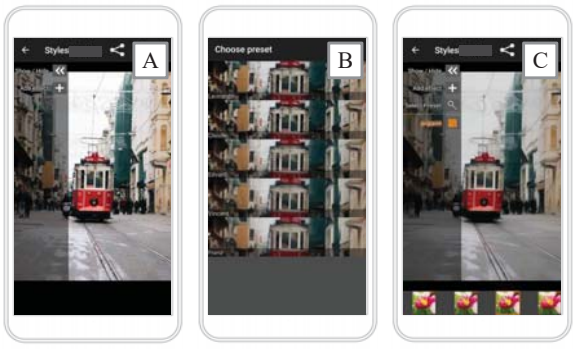

Figure 6: Workflow of the Android image manipulation app. After selecting an image (A) the user can choose an effect and one of the predefined parameter configurations (B) and apply them to the input image (C).

4.3 Desktop Client

The desktop client is realised through a C++ framework, using the Qt application framework that communicates with the orchestration service to utilize the functionality of the server component and to fulfill high-level use cases. The presented approach is tested using a Command-Line Interface (CLI) and a Graphical User-Interface (GUI) application.

CLI Application. The CLI application is used as a rapid application development framework for the analysis and processing of images and image collections. Further, it provides statistics and reports of effects and additional metadata. The CLI can be easily utilized by developers or desktop applications to implement more complex functionality based on the provided effects. For instance, it can be used as a convenient tool to test effects and retrieve an overview of the effect status (e.g., average runtime, load, and bandwidth tests as well as errors). Further, it supports both basic low-level functionality (e.g., listing available effects and filter them) and more advanced features (e.g., batch processing images that reside in specific folders or whose filenames comply to a specified regular expression).

GUI Application. The GUI application demonstrates the applicability of a simple cross-platform image-processing app that applies selected effects with specified presets subsequently to an image. The application is implemented with the Qt GUI module that facilitates the development and deployment of cross-platform software for various desktop systems (Figure 1-C). Because of the minimalistic user interface, the application is well-suited for casual non-professional users, who want to import, process, and export a single image.

5 RESULTS AND DISCUSSION

This section discusses the results obtained by our application examples with respect to its runtime performance, current limitations of the presented approach, and future research questions to address.

5.1 Performance Observations

In general, two major factors affect the runtime performance of service-based architectures: *data transmission* and *data processing*. The transmission time of input and output images over a network highly depends on the image data size and the available network bandwidth. Timings of data transmission are not examined in this paper. The image-processing service output configurations via the format and quality parameters account

ISBN 978-80-86943-41-1

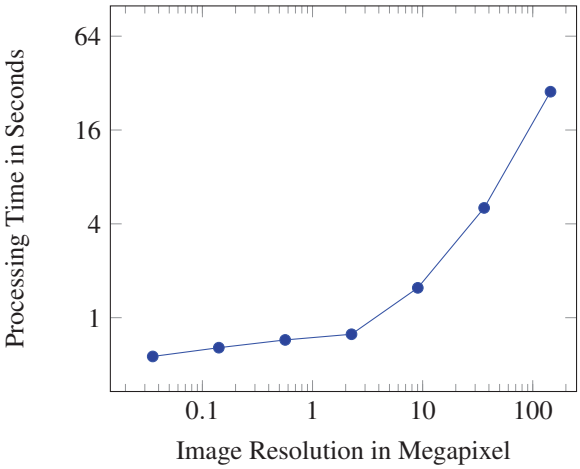

Figure 7: Performance of the image-processing service applying the watercolor effect on images with different resolutions (0.01 to 145 megapixels).

that. For example, within the Google Office Suite add-on, a low-quality image with high compression can be chosen for preview images. The machine hosting a single instance of the processing service has the following specification: CPU: Intel(R) Xeon(R) CPU E5-2637 v4 @ 3.50GHz; GPU: NVIDIA(R) Quadro M6000 24GB; RAM: 64.00 GB; Hard drive: 4 × SanDisk X400 2.5 512 GB in RAID 0; OS: Ubuntu 16.04.3 LTS 64-bit.

The performance measurements of the image-processing service (Figure 7) allow the following observations: (1) the runtime depends linearly on the resolution of the input image while for common resolutions the processing achieves acceptable performance (200-500ms), (2) high-resolution image data up to 16.384×16.384 pixels can be processed, but requires approx. 15 GB RAM; an amount that can hardly be managed during on-device processing on mobile devices, and (3) the prototypical implementation of the image-processing service is hardly real-time capable for common HD resolutions and, thus, is not suited for video-stream processing yet.

Table 1 shows the runtime performance of the image-processing service for different effects with respect to the major processing stages:

Effect Loading: Load the effect, which includes parsing of the XML representation and instantiation of the effect pipeline with the associated GPU objects.

Image Decoding: Load the image, allocate texture memory and data transfer for the image.

Image Processing: Execute the effect pipeline. The result will be stored in the pipeline output texture.

Image Encoding: Export the image to a specified format (e.g., JPEG and PNG) with a specified compression strategy and write it back to disk.

Table 1: Runtime performance of image-processing stages using effects of different complexity. The input is a 2 megapixel JPEG image.

Effect	Load Effect	Decoding	Processing	Encoding	Total
Blur	111ms	169ms	20ms	17ms	**317ms**
Emboss	157ms	174ms	24ms	25ms	**380ms**
LUT	196ms	164ms	25ms	23ms	**408ms**
Oilpaint	242ms	170ms	47ms	25ms	**484ms**
Watercolor	523ms	163ms	67ms	30ms	**783ms**

On complex effects (e.g., oilpaint, watercolor) loading of effects and processing requires more time than on simple effects (e.g., emboss and blur filter). Compared to the on-device equivalent the effect loading stage, which takes 1-3 seconds on device, is significantly faster [2]. However, input and output operations (effect fetching, decoding, encoding) poses the bottleneck of the processing component. Caching mechanisms for the effect fetching stage can result in a considerable speedup, while on high-resolution images the decoding and encoding stages become the major time consuming parts.

5.2 Limitations

The prototypical implementation of the presented approach exhibits some limitations. Since the reusability of web services is maximised, each single service often only provides simple and atomic functionality. Therefore, to enable more complex use cases, the composition of services is a critical question to address. Previous research has shown that composition can be performed automatically, once the web service ecosystem and the desired use cases are strictly formalized [10]. In this work, the composition of services was implemented using a meta service, i.e., the orchestration service described in Subsection 3.4, and client-side, i.e., the thumbnail generation shown in Subsection 4.1 and the effect composition described in Subsection 4.2. As described in the according sections, these approaches come with their own limitations respectively.

Furthermore, the current prototypical implementation assumes a monolithic image-processing service. With the assumption of this service only being handled by one physical machine, this could turn out to be a limiting factor for the scalability of the complete system. A load-balancing system in addition to multiple server-instances would be a potential improvement to overcome this limitation. In addition thereto, each server-instance is still limited with respect to processing capabilities such as maximum texture sizes, number of shader cores, or memory bandwidth.

5.3 Future Research Directions

On the basis of the presented results, various future research directions are possible. Extending the web

ISBN 978-80-86943-41-1

service processing approach with capabilities for video processing is planned for future work. Here, a possible implementation for efficient real-time video processing could involve using live streaming protocols, e.g., Real-time Streaming Protocol (RTSP), and exploiting compression methods. Another aspect to improve the performance of the presented system is scalability. In addition to introducing load balancing in front of the image-processing server(s), the image-processor can be extended to support tiled rendering. This would allow the system to process even higher resolutions than 16.384×16.384 pixels, which might facilitate more application fields for this work, such as the geodata domain.

Furthermore, support of an extended effect parameterization beyond choosing presets might be desirable to some users or application integrations. Allowing fine-grained control over every effect parameter increases the creative freedom but might make it harder for users to achieve desirable results. Featuring a service-oriented architecture, the parameters could be exposed via self-description of services. Furthermore, allowing users to share their own parameterizations as new effects on a platform as demonstrated in [2] is imaginable.

6 CONCLUSIONS

This paper presents a novel concept for service-based processing and provisioning of image-processing techniques with respect to extensibility and applicability of image effects to further domains and current software and hardware systems. The approach is based on the design of atomic services that are orchestrated to higher level services to fulfil sophisticated use cases, such as applying configurable image-abstraction effects to images. Since the image-processing is performed on the server, clients are not responsible for resource-intensive processing tasks and the image-processing can be implemented and optimized only once for a known GPU environment. The presented approach enables cross-platform interoperability with a diverse range of heterogeneous clients. The applicability of the approach is demonstrated to various application domains, such as mobile and desktop applications. The image-abstraction effects are described via a platform-independent representation that enables a highly customizable configuration. These effects and their dependent assets are stored on the server, which provides a high degree of protection for sensitive data (e.g., intellectual property). As part of the technical evaluation performance measurements showed the general applicability of the approach, i.e., image-processing can be performed in a reasonable amount of time. Also, the processing and output of high quality images with resolutions up to 145 megapixels have been shown.

ACKNOWLEDGMENTS

We thank Markus Brand, Merlin de la Haye, Phaedra Goudoulaki, Erik Griese, Moritz Hilscher, Alexander Riese, and Hendrik Tjabben for their contributions to the design and implementation of the presented system. This work was partly funded by the Federal Ministry of Education and Research (BMBF), Germany, for the AVA project 01IS15041B. We further thank Digital Masterpieces GmbH for providing data sets.

REFERENCES

[1] Edward Curry. Message-Oriented Middleware. In Qusay H Mahmoud, editor, *Middleware for Communications*, chapter 1, pages 1–28. John Wiley and Sons, Chichester, England, 2004.

[2] Tobias Dürschmid, Maximilian Söchting, Amir Semmo, Matthias Trapp, and Jürgen Döllner. Prosumerfx: Mobile design of image stylization components. In *SIGGRAPH Asia 2017 Mobile Graphics & Interactive Applications*, SA '17, pages 1:1–1:8, New York, NY, USA, 2017. ACM.

[3] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[4] Zeev Farbman, Raanan Fattal, Dani Lischinski, and Richard Szeliski. Edge-preserving decompositions for multi-scale tone and detail manipulation. *ACM Transactions on Graphics*, 27(3):67:1–67:10, 2008.

[5] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Rfc 2616, hypertext transfer protocol – http/1.1, 1999.

[6] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, 2000. AAI9980887.

[7] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423, Los Alamitos, 2016. IEEE Computer Society.

[8] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar, and Yves Lafon. Soap version 1.2 part 1: Messaging framework (second edition). World Wide Web Consortium, Recommendation REC-soap12-part1-20070427, 2007.

[9] Kaiming He, Jian Sun, and Xiaoou Tang. Guided image filtering. In *Proc. European Conference on Computer Vision (ECCV)*, pages 1–14. Springer, 2010.

[10] Alexander Jungmann and Bernd Kleinjohann. Automatic composition of service-based image

 ISBN 978-80-86943-41-1

processing applications. In *Proc. IEEE International Conference on Services Computing (SCC)*, pages 106–113. IEEE Computer Society, 2016.

[11] Michael Kass and Justin Solomon. Smoothed local histogram filters. *ACM Transactions on Graphics*, 29(4):100:1–100:10, 2010.

[12] Jan Eric Kyprianidis, John Collomosse, Tinghuai Wang, and Tobias Isenberg. State of the 'Art': A Taxonomy of Artistic Stylization Techniques for Images and Video. *IEEE Transactions on Visualization and Computer Graphics*, 19(5):866–885, 2013.

[13] Jan Eric Kyprianidis, John Collomosse, Tinghuai Wang, and Tobias Isenberg. State of the "art": A taxonomy of artistic stylization techniques for images and video. *IEEE Transactions on Visualization and Computer Graphics*, 19(5):866–885, 2013.

[14] Angel Lagares Lemos, Florian Daniel, and Boualem Benatallah. Web service composition: A survey of techniques and tools. *ACM Computing Surveys*, 48(3):33:1–33:41, 2015.

[15] Matthias Mueller and Benjamin Pross. *OGC® WPS 2.0.2 Interface Standard*. Open Geospatial Consortium, 2015. http://docs.opengeospatial.org/is/14-065/14-065.html.

[16] Mike P. Papazoglou and Willem-Jan Heuvel. Service oriented architectures: Approaches, technologies and research issues. *The VLDB Journal*, 16(3):389–415, 2007.

[17] Pietro Perona and Jitendra Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):629–639, 1990.

[18] Amir Semmo, Tobias Isenberg, and Jürgen Döllner. Neural style transfer: A paradigm shift for image-based artistic rendering? Proceedings International Symposium on Non-Photorealistic Animation and Rendering (NPAR), pages 5:1–5:13, New York, 7 2017. ACM.

[19] Amir Semmo, Daniel Limberger, Jan Eric Kyprianidis, and Jürgen Döllner. Image Stylization by Interactive Oil Paint Filtering. *Computers & Graphics*, 55:157–171, 2016.

[20] Amir Semmo, Daniel Limberger, Jan Eric Kyprianidis, and Jürgen Döllner. Image stylization by interactive oil paint filtering. *Computers & Graphics*, 55:157–171, 2016.

[21] Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *Proc. International Conference on Computer Vision (ICCV)*, pages 839–846. IEEE, 1998.

[22] Mircea-Florin Vaida, Valeriu Todica, and Marcel Cremene. Service oriented architecture for medical image processing. *International Journal of Computer Assisted Radiology and Surgery*, 3(3):363–369, 2008.

[23] Joachim Weickert. *Anisotropic diffusion in image processing*, volume 1. Teubner Stuttgart, 1998.

[24] Robert P. Winkler and Chris Schlesiger. Image processing rest web services. Technical Report ARL-TR-6393, Army Research Laboraty, Adelphi, MD 20783-119, 2013.

[25] Holger Winnemöller, Sven C. Olsen, and Bruce Gooch. Real-time video abstraction. *ACM Transactions on Graphics*, 25(3):1221–1226, 2006.

[26] Marcel Würsch, Rolf Ingold, and Marcus Liwicki. Divaservices - a restful web service for document image analysis methods. *Digital Scholarship in the Humanities*, 32(1):i150–i156, 2017.

 ISBN 978-80-86943-41-1