BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **20**, No 3
Sofia • 2020 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.2478/cait-2020-0026

Microservices Centric Architectural Model for Handling Data Stream Oriented Applications

Milu Mary Philip, Amrutha Seshadri, B. Vijayakumar

Department of Computer Science, Birla Institute of Technology and Science Pilani, Dubai Campus, Dubai, United Arab Emirates
E-mails: p20120003@dubai.bits-pilani.ac.in f20150046@dubai.bita-pilani.ac.in vijay@dubai.bits-pilani.ac.in

Abstract**:** *The present-day software application systems are highly complex with many requirements and variations, which can only be handled by more than one architectural pattern. This paper focuses on a combinational architectural design, with the micro-services at the center and supported by the model view controller and the pipes and filter architectural patterns to realize any data stream-oriented application. The proposed model is very generic and for validation, a prototype GIS application has been considered. The application is designed to extract GIS data from internet sources and process the data using third party processing tools. The overall design follows the micro-services architecture and the processing segment is designed using pipes-and-filters architectural pattern. The user interaction is made possible with the use of the model view controller pattern. The versatility of the application is expressed in its ability to organize any number of given filters in a connected structure that agrees with inter-component dependencies. The model includes different services, which make the application more user-friendly and secure by prompting client for authentication and providing unique storage for every client. This approach is very much useful for building applications with a high degree of flexibility, maintainability and adaptability. A qualitative comparison is made using a set of criteria and their implementation using the different architectural styles.*

Keywords**:** *Architectural Pattern, Microservices, Model View Controller, Pipes and Filters, Software Architecture.*

1. Introduction

Software Architecture can be defined as a set of principal design decisions that facilitate structural building and assembly of various components that make up a software application system [1]. It is a key aspect of software development and can be defined as the blueprint of any software application system [2]. It requires information on the number and type of inputs and outputs and component inter-dependencies. The software architecture of an application system is thus defined as a group of structures that constitutes the different elements of the application system, the relations and dependencies between them and the features of the same. It provides module, component-connector and allocation structure for application system development.

The implementation of large-scale applications requires carefully planned structural designs and tools to ensure success in the market and industry. An architecturally rigid application ages with time and runs the risk of going extinct. The monolithic design, which is the oldest and traditional way of architectural design [3], will not be able to cope up with the growing needs and complexities of application systems built today. The software application system that is implemented using such a design faces limited scalability, dependency issues that arises due to large number of components and connectors, increases the maintenance cost and efforts and so on. The micro-services architectural style offers a solution to this problem. It structures the application as a collection of services, and each service implements a functionality of the business logic [16].

A typical micro-services architecture comprises of separate modules for client and server side. The server modules can be hosted on a local area network or internet. The client module can reside at a node in the local area network, internet, mobile device or IoT device. Authentication, on the server side, is done by the component API Gateway. It also ensures that the services do not go outside the scope. The service registry in the micro-services architectural style keeps track of the health of the services and notifies the client whenever a service is down. The core component of the server are the micro-services themselves and each service implements specific functionality. Hence testing and debugging can be carried out easily for the micro-services. They can adapt to new technology with gradual transition and continuous integration. The communication between the components is handled using message queues. Micro-services simplify the design and implementation of individual services. The benefits of the micro-services architectural style include agility, scalability, ability to evolve and maintainability of large-scale distributed systems, reliability, resilience, increased developer productivity, separation of concerns and ease of deployment [4].

In this paper, we present an architectural model that taps the potential of three different architectural styles – micro-services, pipes and filters and Model View Controller (MVC). This approach increases overall flexibility, maintainability and adaptability, especially for large application systems that involve many independent components. This architectural model proposed in this work is generic. It has been tested successfully for a prototype GIS maps processing system, that involves processing of the geographical maps in streams.

This paper is organized as follows: Section 2 gives an overview of the existing works using the micro-services, MVC and pipes-and-filters architectural patterns. It also describes a few real-time GIS application systems. Section 3 proposes an architectural model using a combination of three architectural styles. Section 4 discusses the experimental set-up, the implementation aspects and a test scenario to demonstrate the working of a prototype GIS maps processing system. In addition, a qualitative comparison is made using a set of criteria and their implementation using the different architectural styles. The benefits of the proposed combinational architectural style are highlighted. The last section summarizes the current work and suggests directions for future work.

2. Related work

A good architecture must be capable of evolving with the technology that builds upon it. As requirements become more diverse and systems that serve these demands become more complex, it is seen that the traditional monolithic design is becoming dated. It is rapidly being replaced by a more efficient and fresher architectural model – the micro-services architecture. D r a g o n i et al. [3] discuss about importance of micro-services and their current applications. A monolithic system design involves a high degree of inter-dependencies between the modules of an application system. This in turn leads to implementation, maintenance issues for the system. The micro-services overcome the above issues. In this architectural style, the application modules work independently, cohesively and communicate solely via messages. A comparison is drawn between monolithic and micro-services architectures by S i n g h and P e d d o j u [5]. Their study shows that a micro-services based application outperforms a monolithic application. The granular design of the micro-services architecture, allows for different parts of an enterprise work independently and efficiently.

Micro-services architecture is an evolutionary design "born out of the industry" [4]. It also discusses how advancements in networking and the introduction of cloud technology have popularized solutions like micro-services. With smart cities and IoT technology soon becoming a reality, micro-services architecture offers a stepping-stone for designers and developers alike. K r y l o v s k i, J a h n and P a t t i [7] discuss about the implementation of a micro-services-based platform for deploying IoT systems in smart cities. F e t z e r [6] explains the importance of the micro-service-pattern for building critical applications. The loose-coupling of the services enables load balancing, elastic scaling and high availability of applications. A micro-kernel-based security system combined with secure containers ensures confidentiality, integrity and correct execution of the application.

The Pipes and Filters (P&F) pattern, a simple yet powerful architecture, can be chosen to model the processing segment for any application. The architectural style comprises of a source and a sink and a series of filters or processing units connected by pipes or links that stream data between filters. This pattern offers benefits such as interconnection of non-interdependent entities and ease of modification of components, thereby reducing the maintenance costs and platform-dependence [8]. This architectural model is also characterized by its granularity, i.e., it can handle both low level and high-level tasks (especially the large-scale image processing tasks that make up the example map-processing application described in this paper). The Filter Chain is an abstracted view of the pipes and filters architectural model. Filters correspond to independent but adjacent processing steps that enrich or refine data arbitrarily. They consume input data and deliver output. Pipes interconnect the filters in uniform channels (buffers) to incrementally process data.

S c h e i b l e r, L e h m a n n and R o l l e r [9], in their paper, propose a solution for the problem of data integration in production systems. As the complexity and cost of storing/maintaining redundant data in several places is high, the need for effective integration patterns that will allow data flow between applications is acute. The application of the P&F model is highlighted as a 'P&F mapped workflow architecture' pattern. It resolves the conflict between Pipes and Filter and Workflow architectures used. The paper also describes how the two solutions can be integrated – P&F architecture implemented using flow technology by transforming P&F elements into workflow constructs. It illustrates that P&F architecture does not require its own implementation but can be converted into appropriate models to suit the system – thus establishing the versatility of the P&F model.

G u g g i [10] presents middleware architecture for smart cameras network by extending the P&F architecture along with the self-aware and self-expressive properties. Every filter in the modified P&F architecture has a number of input and output filters. Sending data back to source filter/forwarding to output filter, upon some parameter modification request, is done automatically. This automatic reconfiguration of parameters represents the self-expressive principle. The overall performance of the system is optimized using the automatic adaptation to parameter changes. The data processing is brought as close as possible to the source. This prevents rejection of data at the destination, thus making the middleware system self-aware. These two key properties make the system more adaptable to changing conditions in the environment.

The internet and its usages have been ever-growing and evolving since its conception. Today, the internet supports a wide range of facilities, most of which require remote processing, easy-to-navigate user-interfaces, etc. and most importantly, the isolation of design, development and deployment. The MVC architectural model is one of the most widely used architectural models as it is designed to cater to the above mentioned needs, specifically. Web applications and services should be capable of processing large amounts of diverse data with minimum overhead and downtime. D r a g o s-P a u l and A l t a r [11] discusses the advantages of using the MVC architectural model in the rapid development of web applications by proposing a thin Model – Fat Controller design. The design reduces application development time drastically and improves quality of work for both beginners and experts alike. The MVC architectural model can also be easily tailored to meet the developer's needs. M a t i a s, P a l m a and O l i v e i r a [12] have presented model-based software application development approach derived from the MVC architectural design, for the specific purpose of computing independence and to develop a standard for the same. The combinational architectural model described in this current work considers MVC as one of the architectural patterns to control user-input and handle user-interaction.

Geographic Imaging System (GIS), is becoming more and more relevant today with the advent of the GPS and other mapping technologies. It offers solutions for the processing of large amounts of real-time data. The architectural design of a prototype GIS maps processing system is discussed in the current work. It is based on a combination of three architectural patterns – Micro-services, Pipes & Filter and

Model View Controller. The applications of GIS technologies are numerous. Landslide hazard prediction, detection and assessment is one of the most popular applications of GIS technology in disaster management. H u a b i n et al. [13], present a GIS-based solution for landslide assessment. Their solution is based on GIS and Digital Elevation Modelling (DEM) techniques to map the landslides and evaluate hazard, with high reliability and accuracy. The current work involves the GIS map-overlaying feature, for effective spatial comparison of different maps at common geographical locations.

S h i r a z i et al. [14] describe a modified DRASTIC solution that uses GIS for groundwater mapping. The DRASTIC method evaluates the pollution level for groundwater regionally and it is widely used in Europe and Latin America. The most common procedures included in this method are -converting hardcopy maps to digital format, using well log to create groundwater depth map, using well location information and well log pumping data to create map of saturated hydraulic conductivity, creating final vulnerability maps from overlaying individual characteristic maps. A similar application of GIS maps and software was used to study thermal perception maps to determine the most comfortable times and regions for recreation in the Isparta Province [15]. These procedures, implemented as independent tasks, require considerable amount of effort and time. The combinational architectural model proposed in this paper significantly enhances the operation and performance of the above-mentioned complex tasks. In this paper, the micro-services architecture has been used in combination with architectural patterns like model view controller and pipes and filters. This approach serves the purpose of user interactivity and stream mode of operation for any application system in a significant manner.

With this background, the paper realizes the combination of the aforementioned architectural models through the prototype GIS map-processing application. GIS, or Geospatial Imaging Systems, which has recently become indispensable for almost every large-scale mapping and planning project. It is widely used in navigation, geology, disaster management, resource management and agriculture, to name a few. Integrated with AI and Data Mining, it is bound to be part of every application that uses real-time data analysis.

3. Model description

This section describes a micro-services-centric architectural model for data stream applications as shown in Fig. 1. It is supported with MVC and P&F patterns to realize the functions of data streaming applications. The server-side handles client authentication, provides separate micro-services for *input*, *processing* (download, union and transform) and *output* and also maintains a service registry to track the availability of the micro-services. The client-side enables multiple clients to connect to the server and avail the micro-services pertaining to the application.

This model has been realized and validated for a prototype GIS map processing system. The GIS application system processes vector data in the form of maps using third party GIS processing software. Here, the input, processing and output services are realized using micro-services architectural style.

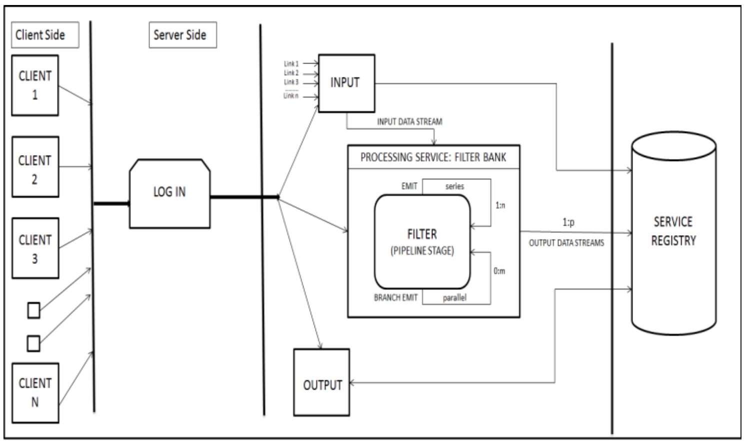


Fig. 1. Micro-services Centric Architectural Model for Data Stream Applications

The processing service is implemented using pipes-and filters architectural style and interactivity with client is established using MVC architectural style. The different components of the architectural design are as follows.

3.1. Authentication

The identity of the user is being validated in this segment. Upon launching the application, the user will be required to register or log-in to the system. The credentials are being checked in the user database. The access to the subsequent segments of the application system is granted only to the valid users. To register as a new user or to remove an existing user from the database, administrator privileges are required (authentication done using the admin-password). The user database will be reset at every launch.

3.2. Input as a service

The input service is invoked by the main method of the application. It prompts the client to enter the names of the filters required for the processing of the input maps, the input entities (here, links (URLs) of the GIS vector maps that must be downloaded) and a user-tag to identify the client's data. The input entities are typically stored in a text file. This is sent to the processing service as an input data stream.

3.3. Service registry

The application is supported by the Eureka Server (also known as Discovery Server) for service registration and discovery. The Eureka Server is used in conjunction with the Spring Cloud Framework for developing the micro-services. It keeps track of all the micro-services present in the application system. It records the IP address and ports on which these services run and notifies the user whenever a service is down.

3.4. Processing as a service

The processing service has a pool of filters that implement different processing functionalities of any application system. It contains a filter bank which comprises one or more filters. The individual filters can be connected in series (one to *n* number of filters), in parallel (zero to *m* number of filters) or in series-parallel connection. Each filter implements a specific functionality based on the user needs and demands. For the GIS map processing application system that has been discussed in this paper, the client or the user has selected the different functionalities to be – Download, Union and Transform. Each function is realized using a filter program that processes the input in streams and provides the processed output to pipes.

Fig. 2 gives an overview of the structure of the processing service in the GIS map processing application system. The following subsections discuss each component in detail.

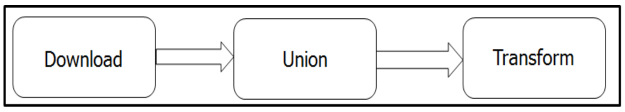


Fig. 2. Filter Bank for the processing service of a prototype GIS application system

3.4.1. Download (F_1)

The download filter takes the input text file provided by the client and downloads the vector maps as shape-files (.shp) from different URLs. Every set of URLs will invoke an instance of the downloader. The filter downloads for every set concurrently. The downloaded maps are stored in specific locations for each client.

3.4.2. Union (F_2)

The *union* filter takes the downloaded maps from the *download* filter. The QGIS application is then invoked to launch the union filter for processing. The input maps are processed incrementally. The processed output maps will be piped to the consecutive *transform* filter.

3.4.3. Transform (F_3)

The transform filter partially geo-references the maps obtained from the *union* filter. The QGIS application is invoked to launch the *transform* filter for processing. This filter outputs the transformed map to the output service.

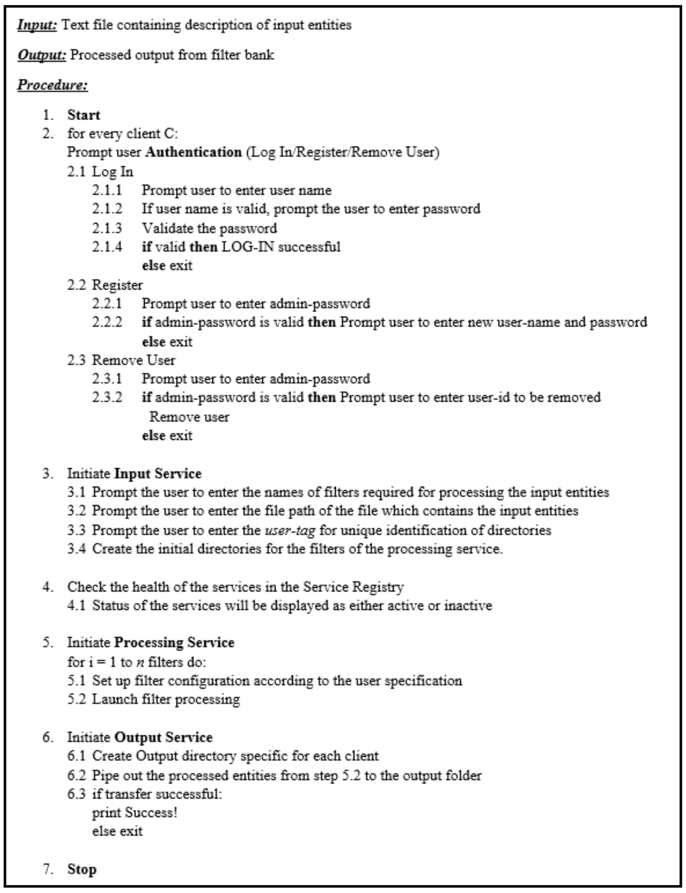
3.5. Output as a service

The output service creates a final output folder for every client. It pipes the output of the processing service to the respective output folder in p output streams for p sets of input given by the client.

4. Algorithm M_PF_MVC

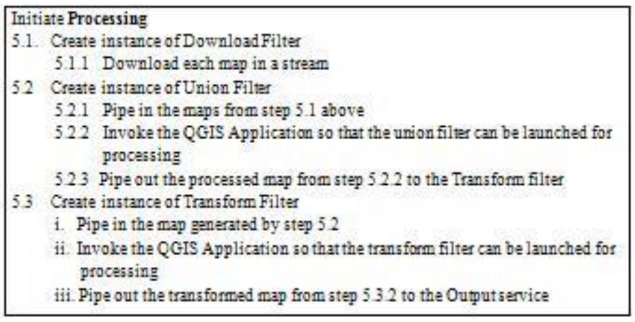
This section discusses in detail the steps involved in realizing the combinational architectural model described in the previous section.

The actions of the algorithm outlined in Fig. 3 are described as follows. When a client establishes a session with the application system, the application prompts for authentication. The client is required to log-in or register with the application. However, only administrators are permitted to register new users or remove existing users from the user database. Administrators are authenticated using the admin-password. Once authentication is completed successfully, the *input* service is initiated. It prompts the client to choose the filters required for processing the input maps, from the filter bank. It also requires the client to enter the file-path to the file containing the download links of the input maps and a unique identifier for identifying the client's data.



```
Input: Text file containing description of input entities

Output: Processed output from filter bank

Procedure:

  1. Start
  2. for every client C:
     Prompt user Authentication (Log In/Register/Remove User)
     2.1 Log In
         2.1.1  Prompt user to enter user name
         2.1.2  If user name is valid, prompt the user to enter password
         2.1.3  Validate the password
         2.1.4  if valid then LOG-IN successful
                else exit
     2.2 Register
         2.2.1  Prompt user to enter admin-password
         2.2.2  if admin-password is valid then Prompt user to enter new user-name and password
                else exit
     2.3 Remove User
         2.3.1  Prompt user to enter admin-password
         2.3.2  if admin-password is valid then Prompt user to enter user-id to be removed
                  Remove user
                else exit

  3. Initiate Input Service
     3.1 Prompt the user to enter the names of filters required for processing the input entities
     3.2 Prompt the user to enter the file path of the file which contains the input entities
     3.3 Prompt the user to enter the user-tag for unique identification of directories
     3.4 Create the initial directories for the filters of the processing service.

  4. Check the health of the services in the Service Registry
     4.1 Status of the services will be displayed as either active or inactive

  5. Initiate Processing Service
     for i = 1 to n filters do:
     5.1 Set up filter configuration according to the user specification
     5.2 Launch filter processing

  6. Initiate Output Service
     6.1 Create Output directory specific for each client
     6.2 Pipe out the processed entities from step 5.2 to the output folder
     6.3 if transfer successful:
         print Success!
         else exit

  7. Stop
```

Fig. 3. Generic Algorithm M_PF_MVC



```
Initiate Processing
5.1.  Create instance of Download Filter
        5.1.1  Download each map in a stream
5.2   Create instance of Union Filter
        5.2.1  Pipe in the maps from step 5.1 above
        5.2.2  Invoke the QGIS Application so that the union filter can be launched for
               processing
        5.2.3  Pipe out the processed map from step 5.2.2 to the Transform filter
5.3   Create instance of Transform Filter
        i.   Pipe in the map generated by step 5.2
        ii.  Invoke the QGIS Application so that the transform filter can be launched for
             processing
        iii. Pipe out the transformed map from step 5.3.2 to the Output service
```

Fig. 4. Processing Service for prototype GIS Application system

The *processing* service is then initiated and filter processing is launched. Fig. 4 shows the operation of the *processing* service specific to the GIS application system. The prototype GIS application system realized in this paper offers three filters for processing, namely: *download*, *union* and *transform*. Finally, after all the data has been processed, the output of the last filter is piped to the output service and made available for the client to collect and view.

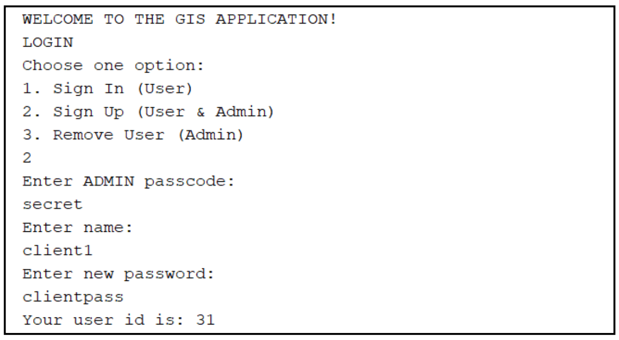
5. Implementation

The GIS map processing system, has been implemented using the JAVA and Python programming languages along with QGIS (Version 2.18.17), which is an open-source software application that handles GIS maps. The model has been implemented successfully on a campus wide local area network of our university. The server side included a host computer with a designated IP address and running WINDOWS operating system. The client side included the local area network nodes accessible from around 200 locations inside the campus. The map-sets used for testing the application were vector data stored as shape-files (.shp format), collected from Natural Earth data repository. It spans the entire earth, on a 1:50 m scale. The GIS map processing system is implemented using both JAVA and PYTHON libraries. It can process multiple maps simultaneously. It is also highly modifiable and can handle different sources of download. It has been successfully tested for the input maps collected from different client nodes. The test scenario is shown below.

5.1. Test scenario

Input: Text file with the URLs from where the maps must be downloaded and the choice of filters.
Output: Merged/Transformed map.

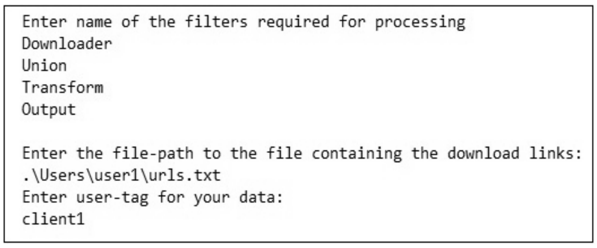
The interface shown in Fig. 5 clearly shows the interaction with the user. The user authentication as described in Section 3.1 above is detailed here.



```
WELCOME TO THE GIS APPLICATION!
LOGIN
Choose one option:
1. Sign In (User)
2. Sign Up (User & Admin)
3. Remove User (Admin)
2
Enter ADMIN passcode:
secret
Enter name:
client1
Enter new password:
clientpass
Your user id is: 31
```

Fig. 5. User Interface for Authentication

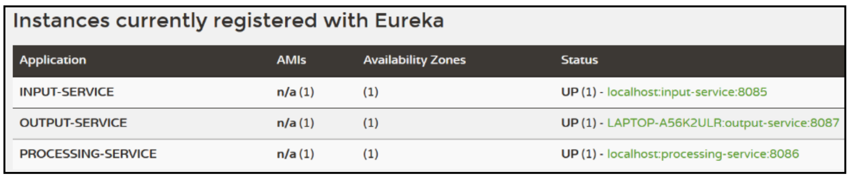
The client is prompted for the *Sign In*, *Sign Up* and *Remove User* options. The client enters the login details and will be added to the user database by the administrator. Only valid clients are given the permission to proceed further in the application system. The client supplies a selection of filters required for the processing of the input maps to the input service, along with the download links for the input maps in a text file and a user tag that will be used later to uniquely identify the client's data. This is shown in Fig. 6.



```
Enter name of the filters required for processing
Downloader
Union
Transform
Output

Enter the file-path to the file containing the download links:
.\Users\user1\urls.txt
Enter user-tag for your data:
client1
```

Fig. 6. User Interface to enter Client requirements

This is followed by the service registry and discovery for the services as shown in Fig. 7.



Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
INPUT-SERVICE	n/a (1)	(1)	UP (1) - localhost:input-service:8085
OUTPUT-SERVICE	n/a (1)	(1)	UP (1) - LAPTOP-A56K2ULR:output-service:8087
PROCESSING-SERVICE	n/a (1)	(1)	UP (1) - localhost:processing-service:8086

Fig. 7. Eureka Server Registry displaying all discovered services

All the required initial directories tagged with the user-name (here, *client1*) are created and the maps are downloaded, as shown in Fig. 8 (i-iv) using the download filter. The maps are then piped to the Union filter. The QGIS algorithm for union is invoked to process the maps. The merged map shown in Fig. 9 is then piped to the transform filter. The QGIS algorithm geo-transformation is invoked to process the merged map. The output-service makes the final output available for the client in the respective folder.

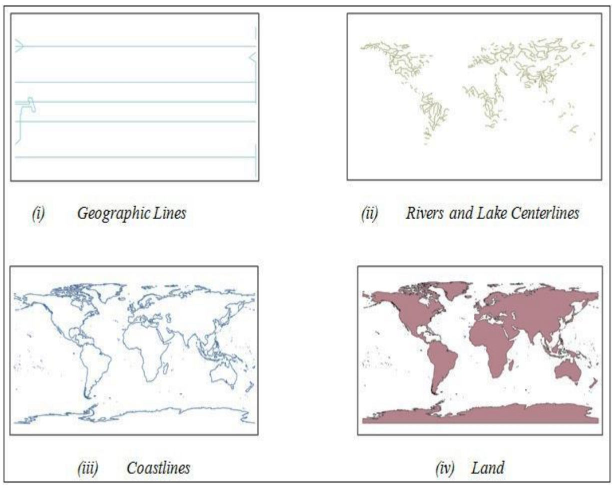


Fig. 8. Downloaded maps from "download" filter

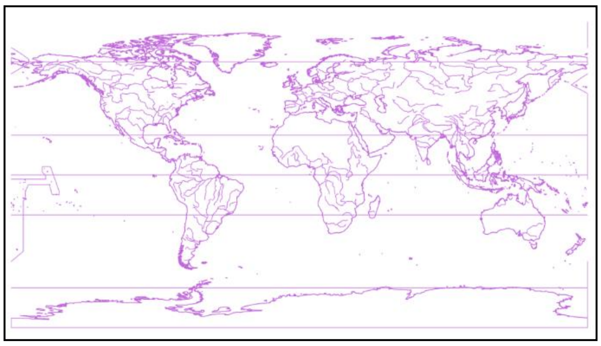


Fig. 9. Processed Map after download, union and transform steps

5.2. Discussion

Table 1 shows a qualitative comparison using a set of criteria and their implementation using the different architectural styles.

Table 1. Architectural Patterns Comparison with proposed Combinational Architectural Design

Sl No	Architectural Patterns	Flexibility	Scalability	Stream Processing	Availability
1.	Pipes and filters	Possible	Not possible	Possible	No
2.	Model view controller	Limited, due to decoupling of models, views and controllers	Possible	Not possible	No
3.	Micro-services	Possible	Possible	Not possible	Yes
4.	Architectural design using combination of P&F, MVC and micro-services	Possible	Possible	Possible	Yes

The architectural design discussed in this paper, uses a combination of three architectural patterns. It facilitates the processing of input data streams, user interactivity and the execution of various functions of an application system as independent services. The component-connector structure facilitates insertion/deletion of components. The order of processing of the components can be changed according to the user requirements. It is easy to identify a faulty component in the application and isolate it rather than traversing through the entire code. This approach makes it possible to locate and correct the faults in any component quickly, without affecting the regular run of the application. The pipes enable continuous flow of data between the components. This minimizes lags during the processing. Since the pipes are the only means of communication between the components, data access is limited to only the participating components thus ensuring their safety and integrity. The proposed architecture is flexible and scalable to any number of clients. The components are modifiable without affecting others, and the possibility of a single point of failure is significantly reduced.

6. Conclusion

This paper deals with a combinational software architectural model for data streaming applications. The micro-services pattern has been deployed for the distribution of services among the various filter components. The pipes and filters architectural pattern facilitates the concurrent processing of input data streams originating from different client sites. The model view controller pattern handles the user interaction and initiates the execution of the filter components.

When a client node sets up a session with the application, it is first authenticated using a user-name and password that is specific to every registered user. The input service accepts the different links or URLs for the input data stream from the client and selects the filters that are needed for processing. The service registry checks the health status of all services, before processing is initiated. The processing service uses the components or filters from the filter bank and executes them in the pipeline mode. Finally, the output service directs the final processed output from the processing service to the corresponding output folders of the registered users.

The model discussed in this paper is very generic. It can handle any data stream oriented application effectively. It has been tested successfully for a prototype GIS maps processing system. This application system has four independent services – authentication, input, processing and output. The *download*, *union* and *transform* are the user selected filters and they constitute the processing service. The output service is used to provide the processed output maps to the client. Thus, the failure of any one service does not affect the others or disrupt the run of the application system. The current work also offers the following benefits such as ease of recombination and reuse, concurrent processing of streams of data, absence of intermediate files and increased scalability. In addition, a software architect can develop flexible applications using different filters or component libraries. This work can be extended further by including additional micro-services and architectural patterns to enhance agile software development.

R e f e r e n c e s

1. T a y l o r, R. N. Software Architecture: Many Faces, Many Places, yet a Central Discipline. – In: ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2009, pp. 303-304.
2. B a s s, L., P. C l e m e n t s, R. K a z m a n. Software Architecture in Practise. 3rd Ed. 2012.
3. D r a g o n i, N., S. G i a l l o r e n z o, A. L. L a f u e n t e, M. M a z z a r a, F. M o n t e s i. Microservices: Yesterday, Today and Tomorrow. – In: Present and Ulterior Software Engineering, Springer, 2017, pp. 195-216.
4. A l s h u q a y r a n, N., N. A l i, R. E v a n s. A Systematic Mapping Study in Microservices Architecture. – In: Proc. of IEEE 9th International Conference on Service Oriented Computing and Applications, 2016, pp. 44-51.
5. S i n g h, V., S. K. P e d d o j u. Container Based Microservices Architecture for Cloud Applications. – In: Proc. of International Conference on Computing, Communication and Automation (ICCCA'17), 2017, pp. 847-852.
6. F e t z e r, C. Building Critical Applications Using Microservices. – IEEE Security and Privacy, Vol. **14**, 2016, No 6, pp. 86-89.
7. K r y l o v s k i, A., M. J a h n, E. P a t t i. C Designing a Smart City Internet of Things Platform with Microservices Archotecture. – In: Proc. of 3rd International Conference on the Future of the Internet of Things and Cloud, 2015, pp. 25-30.

8. L a n g h o r s t, A., S. M a r t i n. Pipes and Filters Architectural Pattern. Hasso-Plattner-Institute for Software Systems Engineering, Potsdam, Germany, 2004.
9. S c h e l e r, T., F. L e h m a n n, D. R o l l e r.Container Based Microservices Architecture for Cloud Applications. – In: Proc. of International Conference on Computing, Communication and Automation (ICCCA'17) , 2017, pp. 847-852.
10. G u g g i, H. Self-aware Middleware for Smart Camera Networks. – In: IEEE International Conference on Pervasive Computing and Communications Workshops, 2012, pp. 566-567.
11. D r a g o s-P a u l, P ., A. A l t a r. Designing an MVC Model for Rapid Web Application Development. – Procedia Engineering, Vol. **69**, 2014, pp. 1172-1179.
12. M a t i a s, L. V., J. G. P a l m a, F. O l i v e i r a. Definition of a Computing Independent Model and Rules for Transformation Focused on the Model View Controller Architecture. – International Journal of Computer, Electrical, Automation, Control and Information Engineering, Vol. **11**, 2017, No 2, pp. 244-251.
13. H u a b i n, W., L. G a n g j u n, X. W e i y a, W. G o n g h u i. GIS-Based Landslide Hazard Assessment: An Overview. – Progress in Physical Geography, Vol. **29**, 2005, No 4, pp. 548-567.
14. S h i r a z i, S. M., H. M. I m r a n, S. A k i b, Z. Y u s o p, Z. B. H a r u n. Groundwater Vulnerability Assessment in the Melaka State of Malaysia Using DRASTIC and GIS Techniques. – Environmental Earth Sciences, Vol. **70**, 2013, No 5, pp. 2293-2304.
15. T o p a y, M. Mapping of Thermal Comfort for Outdoor Recreation Planning Using GIS: The Case of Isparta Province (Turkey). – Turkish Journal of Agriculture and Forestry, Vol. **37**, 2013.
16. G o s, K., W. Z a b i e r o w s k i. The Comparison of Microservice and Monolithic Architecture. – In: Proc. of IEEE International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH'20), 2020, pp. 150-153.

Received: 22.10.2018; Second Version: 15.06.2020; Accepted: 09.07.2020