DORIC: An Architecture for Data-intensive Real-time Applications

Miguel Kassick Cadaviz
University of Vale do Rio dos Sinos
São Leopoldo, Brazil
miguelcadaviz@gmail.com

Kleinner Farias
University of Vale do Rio dos Sinos
São Leopoldo, Brazil
kleinnerfarias@unisinos.br

Lucian José Gonçales
University of Vale do Rio dos Sinos
São Leopoldo, Brazil
lucianj@edu.unisinos.br

Vinicius Bischoff
University of Vale do Rio dos Sinos
São Leopoldo, Brazil
viniciusbischof@edu.unisinos.br

ABSTRACT

This study presents Doric, a software architecture for data-intensive real-time applications. Dimensions of data-intensive real-time applications are introduced, as well as technologies that enable the implementation of such dimensions. A case study involving a portable electroencephalogram (EEG) enabled data collection based on realistic scenarios found in data-intensive real-time applications. The Doric architecture was implemented using recent technologies (e.g., Apache Kafka) for building real-time data pipelines and streaming applications. This prototype was evaluated in five scenarios containing different volumes of data. The obtained results were encouraging and show the potential for applying Doric as a structure to foster the development of modern information systems in organizations and to support serve as a guideline for new corporate architectures.

CCS CONCEPTS

• **Human-centered computing → HCI theory, concepts and models**;

KEYWORDS

Software Architecture, Real Time, Large Scale

ACM Reference Format:
Miguel Kassick Cadaviz, Kleinner Farias, Lucian José Gonçales, and Vinicius Bischoff. 2018. DORIC: An Architecture for Data-intensive Real-time Applications. In *Proceedings of ACM Woodstock conference (SBSI'2018),* Jennifer B. Sartor, Theo D'Hondt, and Wolfgang De Meuter (Eds.). ACM, New York, NY, USA, Article 4, 7 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

In Greece, Doric denotes an architectural structure related to supportability of heavyweight constructions, such as the Temple of Parthenon in Athens. It is one of the three columns of ancient Greek and later Roman architecture used to support heavy marble structures of the Greek temples; the other two canonical columns were the Ionic and the Corinthian. In the context of modern information systems (IS), the reality is not different. Their structures should be designed to withstand or support a heavy volume of data as most of them are data-intensive, real-time (DIRT) applications. These DIRT applications are in a natural way related to Doric architectural structure with regards to how software architecture provides data, integrates heterogeneous data sources, provides information processing, and analysis and storage support.

Nowadays, data are produced comes from devices, such as portables EEG (Electroencephalography), body sensors and cameras. In recent years, the trend of data production has intensified with the rise of the internet of things [13], where interconnected sensors and devices are in practically all everyday objects. These devices are beginning to become popular and to be applied in many areas where information systems are essential, such as mental health care.

For example, portable devices, such as Emotiv Epoc[1], have been used in several actions of mental health care such as collection of mental data, supervision of facial expressions, and the mental health status of patients. These activities produce a big amount of structured and unstructured data. Extracting valuable information from these data implies into properly processing, analyzing, linking, and storing data [12]. However, this becomes a challenging task due to the long processing time in a scenario that users are not willing to wait much for a server response.

Although several works have been carried out by proposing techniques and architectures for information systems, they do not treat information systems as a DIRT application. This implies that hardware and software should operate under time constraints, returns results quickly enough to influence the environment whereas this system receives and processes the data. Still, when developers need to implement DIRT applications, traditional software architectures are no longer adequate [6][10]. There is a lack of a comprehensive architecture to properly represent important aspects of generic DIRT applications.

This study, therefore, presents Doric, a software architecture for data-intensive real-time applications. Dimensions of data-intensive

SBSI'2018, Junho 2018, Caxias do Sul, Rio Grande do Sul Brazil
© 2018 Copyright held by the owner/author(s).
ACM ISBN 123-4567-24-567/08/06.
https://doi.org/10.475/123_4

[1]Emotiv Epoc: https://www.emotiv.com/

real-time applications are introduced, as well as technologies that enable the implementation of such dimensions.

A case study involving a portable electroencephalogram (EEG) enabled data collection based on realistic scenarios found in DIRT applications. The Doric architecture was implemented using recent technologies (e.g., Apache Kafka[2]) for building real-time data pipelines and streaming applications. This prototype was evaluated in five scenarios containing different volumes of data. The obtained results were encouraging and show the potential for applying Doric as a structure to foster the development of modern information systems in organizations and to support serve as a guideline for new corporate architectures.

This article is organized as follows. Section 2 presents the proposed architecture. Section 3 describes the results and discussion of the empirical evaluation. Section 4 presents the related works. Finally, Section 5 presents the conclusion and final considerations.

2 THE PROPOSED ARCHITECTURE

This section describes the proposed architecture. Section 2.1 presents an overview of the Doric architecture. Section 2.2 introduces the architectural design by describing the Doric's requirements and architectural components.

2.1 General View

Figure 1 presents the overview of the proposed architecture. It is composed by six steps to process data and six architectural components. These steps are described below.

- **Step 1:** Producers send information to the application servers to execute the data analysis;
- **Step 2:** The received data are processed. Once the process is finished the information are sent back to the servers. Then, the servers store this information;
- **Step 3:** This data flow is not mandatory. An application may use this flow in the case of requiring a detailed analysis over a specific block of information;
- **Step 4:** The resulting data come from the analysis step that are stored in a distributed database;
- **Step 5:** The consumers can receive the information without any treatment;
- **Step 6:** This data flow is used when consumers desire to access the result of the analysis.

The objective of the proposed architecture is receiving a huge amount of data in real-time from many producers at the same time. For this, our architecture has a broker that receives the input data. This broker controls the processing, analyses, and storage of these data providing them to consumers. This component enables the relation between data source and consumer components. Each Doric component will be described in Section 2.2.2.

Table 1 shows as the Doric components can be supported by current technologies. The Doric architecture defines standards to support DIRT applications. This will enable developers reusing aspects of the architecture in multiple domains, such as smart cities [1], health systems [8], and smart highways [4]. A system organized in this manner allows developers mapping and modelling a

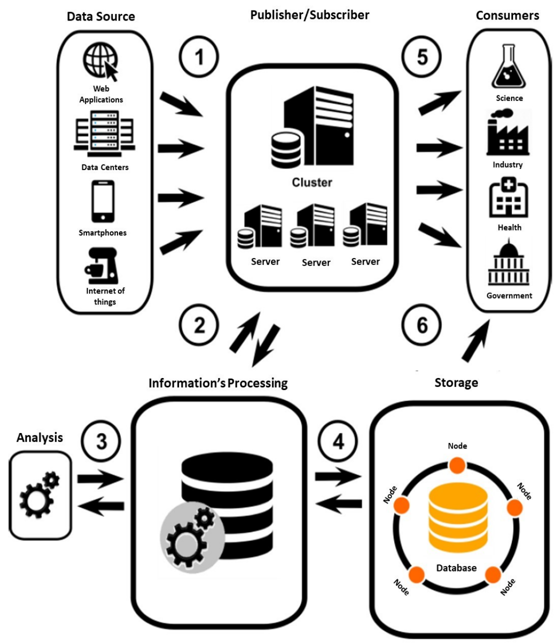


Figure 1: Proposed Architecture.

structure in distinct blocks, as proposed in [14]. This enables the encapsulation of assignments and responsibilities of each block, as well as facilitating the maintenance, and configuration of each step. This is because the well-defined interfaces were designed with the purpose to make feasible an architecture with low coupling.

Table 1: Dimensions of DIRT applications (based on [14]).

Data Sources	Publisher and Subscriber	Information Processing	Analysis and Storage	Consumers
Data Centers	Apache Kafka	Apache Spark	Apache Cassandra	Health
Web Applications	Amazon Kinesis	Apache Storm	Apache Hbase	TI Operations
Air Pictures	Google Pub/Sub	Apache Stamza	Apache Druid	Retail
Machines	MapR Streams	Twitter Heron	Elasticsearch	Government
Buildings	Apache DistributedLog	Amazon Kinesis	Apache Solr	Agriculture
Vehicles		Google Dataflow	MapR-DB	Marketing
Wearables	Azure Event	Apache Apex	FiloDB	Manufacture
Smartphones		Apache Flink	memsql	Industry
Tablets		Apache Flume	Apache Kudu	Science
Internet of Things		Azure Strem Analytics	Google Cloud Storage	Financial Services
		Apache Kafka Streams	Google Cloud Bigtable	Media Agencies

[2]Apache Kafka: https://kafka.apache.org/

2.2 Architectural project

The proposed architecture was built in two steps. In the first step, we identified the requirement an architecture must have. The second step, the technologies that implement each architecture module is described. These technologies are defined according to the defined requirements in Step 1. These steps are described below.

2.2.1 Requirements of the Proposed Architecture. The requirements were elaborated based on suggested features on the articles and research reports. We elaborated three non-functional requirements (**NFR**). These requirements are described below.

- **NFR01: support the process and reception of large-scale information with integrity.** The architecture must support and guarantee the correctness of information's that are transferred in high volume.
- **NFR02: able to return real-time responses.** The analyzed information's must be provided in real-time to the consumers by the system;
- **NFR03: the architecture must be extensible.** The architecture must be well encapsulated and provide extensible points. This enables reuse of modules and the addition of new ones.

2.2.2 Architectural components. This section describes the Doric components as well as suggests some technologies to implement them. Figure 3 shows how the components of the architecture were organized. Table 1 suggests some technologies to implement the Doric components. The chosen technologies seek to meet the requirements previously mentioned. Note that each dimension present in Table 1 corresponds to a Doric component, which is implemented by using technologies for building real-time data pipelines and streaming applications. The following each component is briefly described:

- **Data source.** The data source is not present in the architecture because they are external components. This architecture is adapted to receive information from any data source.
- **Publisher/subscriber.** We choose the Apache Kafka for this component. Kafka is largely applied in the industry and is open source. Kafka is suitable for two kinds of applications: first, obtaining data in real time from many application and systems safely. Second, systems that convert or react to a certain data flux in real-time. Kafka also promotes the abstraction of the data record from a determined topic. Topics are categories for which the records are published. The topics in Kafka are always multi-subscriber, i.e., a topic can have "n" subscribers to access their data. The version of Apache Kafka used in this architecture is 0.10.2.0. In addition, in this module the Apache ZooKeeper are implemented to cooperate with kafka [7]. Apache ZooKeeper [3] provides several functionalities for distributed applications, such as distributed configuration management, information coordination, and locks. The deployed version was ZooKeeper 3.4.9. Both Kafka and ZooKeeper use Java 8 [11] release 131 of the Java Runtime Environment (JRE). JRE consists on a Java virtual machine for running applications.
- **Information Processering:** The Apache Spark [2] is responsible to manage the information processing. This technology is powerful and open source.
- **Data analysis:** We implemented an application using Java to manage the data analysis. This is because there is no suggestion and consolidate techniques to conduct the data analysis.
- **Storage of information:** These components implement the MongoDB [9] database. This database is the state-of-art technology for storing massive data, and non-structured information's. We used the MongoDB version 3.4.4.
- **Consumers:** Consumers are not defined in this architecture because they are external entities. Any external entity can consume the data provided by the architecture.

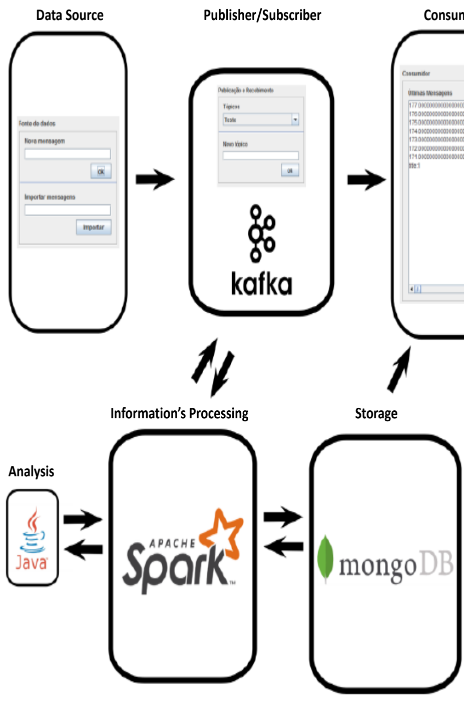


Figure 2: Implemented architecture.

3 ASPECTS OF IMPLEMENTATION AND EVALUATION

This section presents an empirical evaluation of the proposed architecture. The architecture is evaluated through realistic scenarios, in which a large amount of data is generated in real time from a portable electroencephalogram (EEG). Section 3.1 describes the implementation aspects of the proposed architecture. Section 3.2 describes the adopted evaluation method. Finally, Section 3.3 presents the results and discussion about the performance of the implemented system that adopted the proposed architecture.

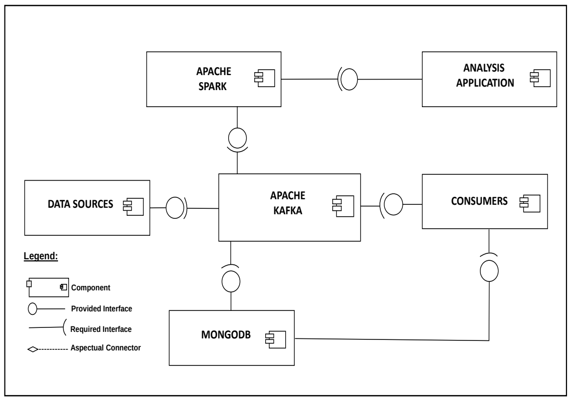
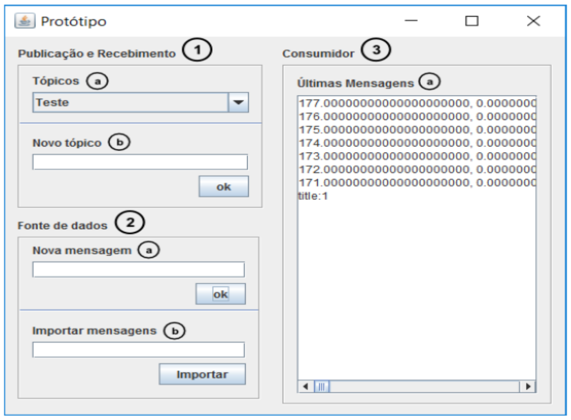


Figure 3: Component diagram.



```
177.00000000000000000000, 0.0000000
176.00000000000000000000, 0.0000000
175.00000000000000000000, 0.0000000
174.00000000000000000000, 0.0000000
173.00000000000000000000, 0.0000000
172.00000000000000000000, 0.0000000
171.00000000000000000000, 0.0000000
title:1
```

Figure 4: An overview of the prototype.

3.1 Implementation Aspects

The publishing and receiving stage, implemented with Apache Kafka, controls the entire data flux, acting as the central nervous system of the structure, but the available producer and consumer in the prototype work via prompt through the use of batch files. In addition, activities such as starting the Kafka server, creating a new information topic, or even listing messages also are managed through the prompt command lines.

The prototype of the Doric architecture was constructed using Java language. Maven[3] were used to manage the construction, and dependencies of the project. User interface was created using Swing java library using Netbeans IDE. The prototype was built to simulate simple interactions with the Kafka cluster, the sending of information to the servers, and consumers obtaining the analyzed information.

Figure 4 shows an overview of the prototype. It is possible to verify the existence of areas representing the dimensions of DIRT applications. Each of these areas implements some functions related to the respective area. The list bellow describes these functions according to each dimension:

The DORIC components implemented in the prototype were:

(1) **Publish and subscribe.** This component, represented in Figure 4(1), is related to the functions responsible for: (a) *change topics*: it allows to switch between the existing topics in Kafka. As previously mentioned, a topic is a category in which the input data are recorded; (b) *create topics*: it allows the creation of a new topic.
(2) **Data source:** We have used the EPOC Emotiv [4] as data source. This component, represented in Figure 4(2), is responsible for the functions related to manage of data sources. These functions enable: (a) create new message: generates a single and unique message in the selected topic; (b) import messages: performs the import of plain text messages from the file to the selected topic.
(3) **Consumer:** responsible to accommodate functions to manage Consumers, such as: (a) last messages: displays the last messages sent to the selected topic.

3.2 Evaluation method

The purpose of architecture evaluation is to present some results obtained after submitting information to the publishing and receiving component. Then, the objective is to demonstrate the architecture to handle large-scale and real-time information. The architecture was built on a virtual machine with the following requirements:

- Windows 10 Operating System;
- Intel i7 (2,7 GHz) CPU with 4 cores;
- 4 GB RAM DDR4 2133 MHz of memory;
- 30 GB (5400 RPM) of Hard Disk.

The scenario where this architecture is evaluated consists in processing and analyzing data from a health-care system. This system collects mental indicators data from a portable EEG. Then the test is divided in two steps.

The first assessment considers the architecture's ability to read large files. For this, several files with information of a wireless electroencephalogram in a csv file were submitted to the infrastructure. The largest of them has 50 MB of information, totaling more than 50 thousand of lines records, and around of 51 million characters. The second step tests the concept of large scale. For this, several threads are being created to process the associated data of the producer, to the system publish information simultaneously.

3.3 Results

Measurements of time used in the experiment were carried on the upper layers of the prototype, thus creating a more realistic environment for the end user. However, this method of measuring just considering the time spent in the transmission of information such as processing time. Several tests of single message submission were performed using the prototype, which obtained an average response time of 318 ms. We did not observe a correlation between response time and the size of the message, the type of content, neither the frequency of the topic structure.

[3]Maven: https://maven.apache.org/
[4]EPOC Emotiv: https://www.emotiv.com/epoc/

The evaluation of capacity of real-time response was performed with five different file sizes. Table 2 presents the obtained results. The import process was repeated fifty times for each file to obtain reliable values. The size of the file influences in the processing time. The time increased when larger files were processed. We also observed the time between sending the last record and the notification of process completion. In this case, the average time also increased according to the size of the file. Specifically, the average time increases significantly for 30 MB files. Figure 5 illustrates the results contained on Table2.

Table 2: Test results of data provided from direct files.

File Size (MB)	Dispatch Time Averarge (ms)	Receiving Time Averarge (ms)
10	343,66	1274,3
20	396,82	1400,25
30	425,62	2019,19
40	906,72	2227,83
50	1628,45	2262,6

Next, we also evaluated the architecture's capability to handle large-scale data. For this, we executed multiple threads to simulate the role of data sources. These threads simultaneously executed and provided the same topic. Each thread creates a producer and generates a certain number of messages. We monitored two average times in this case: (1) the average time spent on sending the messages to the server; and (2) the average time the infrastructure takes to signalize that it received the information. Figure 5 depicts the results. Table 3 specifies the results obtained.

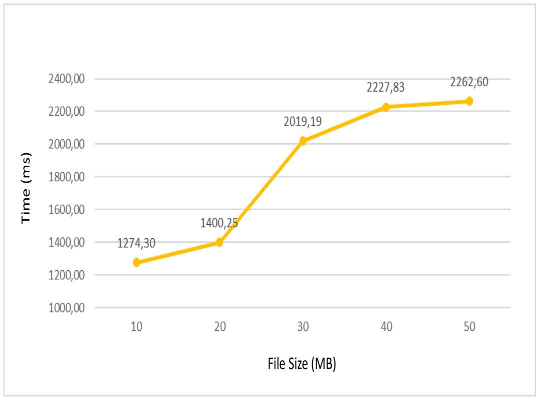


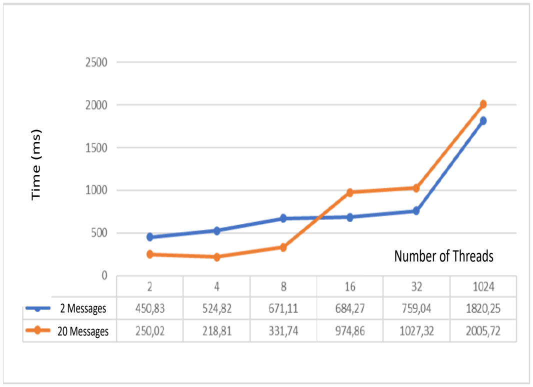
Figure 5: Receiving time average.

The results evidence a gradual increase in the receiving time when there are more threads in execution. The same occurs in the moment that the number of messages increases. This behavior was expected because the tests were conducted on a virtual machine. There is a limitation regarding the hardware capability, e.g., there is only one CPU. This implies that a larger number of threads causes congestion on the processing queue. There is more processes waiting on the queue because there is no available resource to execute all simultaneously. Furthermore, this overall hardware resource is shared with the operating system on which the virtual machine is deployed. This also impacts negatively in the average time.

Table 3: Test results of data provided from threads.

Threads	Menssages Number	Dispatch Time Averarge (ms)	Receiving Time Averarge (ms)
2	2	0	450,83
4	2	0	524,82
8	2	0	671,11
16	2	0	684,27
32	2	3,2	759,04
1024	2	259,2	1820,25
2	20	0	250,02
4	20	0	218,81
8	20	0	331,74
16	20	0	974,86
32	20	6,32	1027,32
1024	20	314,48	2005,72

Figure 6 shows the results obtained. Analyzing the extreme case on the second test, there are 20 messages being sent from 1024 data sources simultaneously. This is a total of 20,480 messages. These numbers of messages were processed in 2005,72 ms. In the case of transmitting messages from a single producer, we reached at the number of 106,650 messages processed in 784ms. Due to resource limitation, the increasing of threads impacts negatively on the performance of messages sent.



	2	4	8	16	32	1024
2 Messages	450,83	524,82	671,11	684,27	759,04	1820,25
20 Messages	250,02	218,81	331,74	974,86	1027,32	2005,72

Figure 6: Representation of receiving average time from tread.

4 RELATED WORKS

There is a growing concern on which architecture must be used for applications that process large volumes of data in real time. Recent literature has explored this challenge in the last years. However, little has been done to clarify about how different architectural approaches might be brought together to address particular facets of DIRT information systems.

By doing so, this section explores three architectures for DIRT applications applied to different domains. Section 4.1 describes an architecture for scientific applications. Section 4.2 describes an architecture for weather applications. Section 4.3 presents an architecture for application control. Finally, Section 4.4 presents a comparative analysis between the Doric and the architectures discussed in order to identify the similarities and differences between them, as well as the research gaps the proposed architecture fulfills.

4.1 Large-Scale and Real Time Processing for Scientific Applications

Based on dynamic data-driven application systems (DDDAS) concept, Cao and Li propose in [5] a framework to optimize the performance of data processing, where data is generated in high volume, about 10 MB per second, almost 1 TB per day. Another challenge is the issue of transportation, since the equipment is spread over thousands of miles away. In this context, performance is a critical factor, since the processing of the collected information must be completed in 30 minutes.

The architecture proposed is based on four layers: (1) application: allows the input of parameters and publishes the results in a user-friendly way; (2) data processing: performs additional data processing based on monitoring outputs; (3) data monitoring: dynamically accompanies information on a central monitor; and (4) instrumentation: receives elements collected from one or more measurement systems and submits them to treatment steps, subsequently storing the assembly in the instrumentation layer. According to [5], the framework's strengths are in data reduction algorithms, which can reduce a block of data from 9063 MB to 29 MB, thus reducing possible bottlenecks in the transport area.

The authors believe that, in the near future, the data scales will grow exponentially. In this scenario, although the concept of DDDAS is directed to simulation applications, the framework will have the possibility of revealing its potential for several scientific applications.

4.2 Real-time Processing for Weather Stations

In China, it is common to use autonomous weather stations with more than 30,000 units [16]. Data-generating devices and control terminals are commonly located in different locations, such as fields, islands, reservations, among others.

Using the concept of client/server, all collected data is sent to a data server simultaneously. In this system, where there is a large data transmission network that is transmitted with a small amount of information and in large scale, making use of sequential transmissions, it is necessary to implement an efficient real-time processing, capable of receiving and treating the data synchronously.

Guangsheng and colleagues in [16] presents a detailed discussion on the design of a system for receiving and processing data from autonomous meteorological stations in Guangdong province. Each observation station produces a group of data every 5 minutes on days of good weather or 1 minute in atypical situations. Thus, the model must be auditable and extensible according to the technical or operational needs of the meteorological stations, in addition to avoiding the congestion in the observation data uploaded at critical moments. For this, the premise is that the entire processing and data storage cycle must be completed within 1 minute or less.

This system was deployed in more than 1700 weather stations for over a year. Running with stability, efficiency and reliability. All stations send their packets to the control center at the same time every minute. The time for receiving, processing and uploading the response for each cycle was less than 1 minute and the hardware requirements are low, reducing maintenance and reducing the complexity of the systems used.

4.3 Real-Time Scheduling for Large Scale Application Control

In [15], Waknis and Sztipanovits argue that real-time application requirements can be satisfied by a scheduling scheme appropriate to a model-based programming environment. They developed an intelligent process control system, which is a model-based system and acts as a real-time supervisor of large-scale industrial processes, able to control various activities, monitor different operations, and making use of the sensor information to diagnose a system failure [15].

The schema is constructed in three layers: (1) model building tools layer, which allows the qualitative and quantitative representation of the information; (2) model interpretation layer, where the information is stored in a descriptive way, which is used for monitoring, control and diagnosis, besides having the capacity to transform the models into executable computational structures; and (3) the environment execution layer, which is implemented with the multigraph architecture, using directional graphs. The nodes of the graph represent the computational blocks and their connections represent the flow of data between them. Each block has a program segment that is staggered according to demand and data flow.

One notable feature that is offered by this framework is the ability to dynamically reconfigure. This is not limited only to adjustments in system parameters, allowing for architectural changes at runtime by the inclusion and removal of nodes and connections. The structure also presents interfaces for receiving information from the external environment and integration with its consumers; in addition, it also has an interface, implemented by each node, responsible for the parallel and scalable processing of the input data and a similar function that processes the output data.

4.4 Comparative Analysis

This section presents a comparative analysis of the related works. This comparison serves as a basis to identify the similarities and differences between the Doric and the selected works. The comparison criteria seek to reveal specific characteristics of the related works, which can be contrasted with the proposed architecture. The comparison criteria are discussed as follows:

- **Data source:** Studies that seek to offer expandable architectures, allowing to support several data sources. Modern architectures for information systems need to give their protection to multiple types and formats of data.

Table 4: Comparative Analysis

	Modularized	Data Source	Publish and Subscrib		Information Processing		Analysis		Storage		Consumers
		Expandable	Expandable	Scalable	Expandable	Scalable	Expandable	scalable	Expandable	Scalable	Expandable
Real-time and Large-scale processing for Scientific Applications	~	-	~	+	~	+	~	+	~	+	-
Real-time and Large-scale processing for Weather Stations	~	-	~	+	~	+	~	+	~	~	+
Real-time Scheduling For Large-scale Control Applications	~	~	+	+	~	+	~	+	~	+	+
Proposed Architecture	+	+	+	+	+	+	+	+	+	+	+

Legend: "+": Supports "~": Partially Supports "-" : Does not Support

- **Publish/Subscribe:** Studies that are based on the publish-subscribe pattern.
- **Information processing:** Studies that are scalable and expandable for information processing. Processing information is fundamental to any architecture for the development of modern information systems.
- **Analysis:** Studies that provide support for data analysis through machine learning techniques.
- **Storage:** Studies that are extensible and scalable with regard to the type and volume of data storage.
- **Consumers:** Studies that are flexible to support different types of customers.

Table 4 presents the comparative analysis considering these criteria. It is observed that only the Doric fully meets the defined criteria, highlighting the contribution and the differential of this work.

It is possible to verify that, in spite of the great scalability of the evaluated projects, there is a deficiency in the issue of expansion and reuse of the structure, for this reason, the proposed architecture aims not only to satisfy the demands presented, but also to provide a structure divided into modules and reusable, making use of microservices and emphasizing the separation between its processing stages. Such an implementation favors the construction of expandable, reusable and easily understood structures, contributing both to existing systems and to those that will still be built.

5 CONCLUSION

The adoption of DIRT applications increased in recent years. However, there is a lack of standardization and modularized architectures. This implies that a specific infrastructure was needed to be projected to attend a different demand of the applications. In other words, a different construction was made despite they have the same essential purpose. This purpose is to serve as a large-scale and real-time architecture.

To resolve this problem this work proposed the Doric, a flexible and modular architecture for data-intensive and real-time systems. This architecture was built to accommodate the state-of-the-art technologies related to this purpose. A prototype was implemented to conduct an empirical evaluation. This evaluation demonstrated that the architecture could process a large volume of data in a short period of time. This was possible even with a hardware limitation. Lastly, the future work consists of deploying the architecture in an environment with improved hardware capability, and evaluate the architecture adaptability in more scenarios.

6 ACKNOWLEDGMENTS

Thank you to Unisinos for providing the spaces and resources to conduct this research.

REFERENCES

[1] Ricardo Alexandre Afonso, WM Da Silva, GHRP Tomas, Kiev Gama, Alezy Oliveira, Alexandre Alvaro, and Vinicius Cardoso Garcia. 2013. Br-SCMM: Modelo Brasileiro de Maturidade para Cidades Inteligentes. *Simpósio Brasileiro De Sistemas De Informação* (2013).
[2] Apache. 2018. Apache Spark. "https://spark.apache.org/". [online, accessed on january 2018].
[3] Apache. 2018. Apache Zookeper. "https://zookeeper.apache.org/". [online, accessed on january 2018].
[4] Luciana Regina Bencke, Anderson Luiz Fernandes Perez, and Osvaldo da Costa Armendaris. 2017. Rodovias Inteligentes: uma visão geral sobre as tecnologias empregadas no Brasil e no mundo. *iSys-Revista Brasileira de Sistemas de Informação* 10, 4 (2017), 80–102.
[5] Junwei Cao and Junwei Li. 2011. Large-scale real-time data-driven scientific applications. In *Networking and Distributed Computing (ICNDC), 2011 Second International Conference on*. IEEE, 116–121.
[6] Dave Evans. 2012. The Internet of Things how the next evolution of the internet is changing everything (april 2011). *White Paper by Cisco Internet Business Solutions Group (IBSG)* (2012).
[7] Apache Kafka. 2017. Apache Kafka: a distributed system platform. Introduction. "http://kafka.apache.org/intro.html". [online, accessed on january 2018].
[8] Júlio Menezes Jr and Cristine Gusmão. 2014. InteliMED–Proposta de Sistema de Apoio ao Diagnóstico Médico para Dispositivos Móveis. *iSys-Revista Brasileira de Sistemas de Informação* 6, 1 (2014), 44–61.
[9] Mongo. 2018. MongoDB. "https://www.mongodb.com/". [online, accessed on january 2018].
[10] Linda Northrop, Peter Feiler, Richard P Gabriel, John Goodenough, Rick Linger, Tom Longstaff, Rick Kazman, Mark Klein, Douglas Schmidt, Kevin Sullivan, et al. 2006. *Ultra-large-scale systems: The software challenge of the future.* Technical Report. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST.
[11] Oracle. 2018. Java 8. "http://www.oracle.com/technetwork/pt/java/javase/downloads/jre8-downloads-2133155.html". [online, accessed on january 2018].
[12] Dan Puiu, Payam Barnaghi, Ralf Toenjes, Daniel Kümper, Muhammad Intizar Ali, Alessandra Mileo, Josiane Xavier Parreira, Marten Fischer, Sefki Kolozali, Nazli Farajidavar, et al. 2016. Citypulse: Large scale data analytics framework for smart cities. *IEEE Access* 4 (2016), 1086–1108.
[13] The Statistics Portal STATISTA. 2017. Internet of Things (IoT): number of connected devices worldwide from 2015 to 2025 (in billions). https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide
[14] STRATA. 2015. Large-Scale Real-Time Applications. [Strata + Hadoop World Conferences].
[15] Prashant Waknis and Janos Sztipanovits. 1993. Hard real-time scheduling for large scale process control applications. In *Proceedings of the IEEE Workshop on Real-Time Applications*. IEEE, 71–75.
[16] Guangsheng Wu, Zhenlang Ao, Jianyong Li, and Qinqiang Zhou. 2011. A Real-time Receiving and Distributed Processing System for Large-scale Burst Data. In *2011 Second International Conference on Networking and Distributed Computing (ICNDC)*. IEEE, 111–115.