International Journal of Latest Trends in Engineering and Technology
Special Issue IC3NS 2018, pp. 047-050
e-ISSN:2278-621X

MODEL-DRIVEN DASHBOARD FOR REAL-TIME DATA MONITORING AND ANALYSIS

Charul Rathore[1], Dr.A.K.Singh[2], Alok Jadhav[3]

The increasing dissemination of sensors in a wide variety of scenarios has lead to an explosive growth of data and that trend is only going to continue. To take advantage of this data proliferation, we have developed a standardization of data interoperability (D.I.) and using that model we have created a dashboard for real-time data analysis: MONITIO.

1. INTRODUCTION

Real-time data feedback is growingly indespens-able for information and communication based intercessions in the developing world. The processes for summarization and analysis of datasets are increasingly needed in data-centers, the sales and marketing department inventories [2], patient databases in hospitals [1] etc. These datasets update over time and that time-period can be as small as every millisecond and can go up-to a day or a week depending on monitor-ing needs. MONITIO sits between data collec-tion and reporting [3]. Its user interface (UI) is written in Ruby with rails framework and data visualization is done by React-Redux. D3.js (Javascript library) powerful visualization com-ponents are used to bring data to life by creat-ing SVG graphs and charts for smooth transi-tion and interaction. Since we have used MVC pattern, the code will be easily extendable for encouragement of community use and progres-sive development.

2. RELATED WORK

Model-Driven methodologies [6][7] are a grow-ing trend for development of system software of large scale due to code regeneration (or re-use) and high level abstraction. Their wide application in related areas include software reuse [8], easier reverse engineering [9], and UI design. The adoption of model-driven design [3] have several bene ts including bringing down the software development time, escalate code maintenance, and ameliorate code quality.

The categorization of existing solutions for analysis can be done as custom, hosted and o ine tools [4]. The Custom tools are con-siderably expensive and are outside the range of most organisations since they are built by third party or in-house. Moreover, new task adaption is toilsome for them and result in double e orts. There are functional limitations of Hosted tools which makes them inapposite for our needs. O ine tools, though possessing exibility, create barriers to entry because they require programmatic wrappers that allow truely dynamic experience and work ows. In summary, these tools are challenging to maintain and adapt, not exible and do not allow a centralized data storage, respectively.

Several companies are providing relevant solutions as the concept of dashboards is gaining a lot of interest and attention, such as Business Objects, Hyperion and IBM. Nevertheless, these approaches and solutions do not coalesce e ciently with the monitoring analysis and performance models, and also require e orts to develop and maintain. Hence, in contrast to these approaches we propose a model-driven method for dashboard design.

This high level model integrate coherently with the performance models which in succession enables an end-to-end design process [5]. An-other example of most commonly used tool for visualizing time series data for application analytics and infrastructure is Grafana [10] which is an open source metric analytics and visualization suite.

To accomplish the requirements of our and many other research tasks, we built an easy to use dashboard that provides the ideal combination of aggregations and updates.

3. ARCHITECTURE

This section brie y discusses the architecture of the application which is an amalgamation of the collector (our database) and the dashboard (our user-interface).

[1] Mody University of Science and Technology, Laxmangarh, India
[2] Mody University of Science and Technology, Laxmangarh, India
[3] SGGSIET, Nanded, India

3.1. Standardization

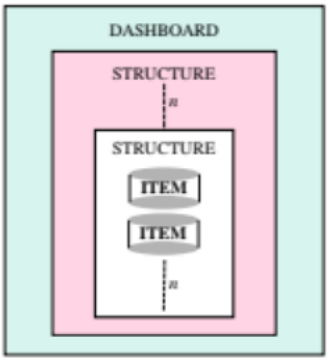

Fig. 1: Representation of Standardized Format for Data Interoperability.

The rudimentary concept of the design is to make the input feed for dashboard mainstream: data from miscellany sources can be brought to-gether for monitoring and analysis purpose us-ing JavaScript Object Notation (JSON) web re-quests. The overview of our standardized for-mat can be seen in Fig. 1. It includes a UI as dashboard which is a combination of nested structures. A structure can consist of another structure or an item. An item is the most pro-found unit of the format as shown in Fig. 2 and it can consist of a data from sensor, SVG graphics or timeline data.

Fig. 2: Types of Items.

3.2. Collector

The Collector uses Ruby SNMP library to re-quest and collect data from sensors. Data is collected from multiple sources, and converted into our standardized JSON format for output which can be read by any source. The versa-tile nature of rails is used in both backend (as collector in our case) and as user-interface. The former is not mandatory because it is only cru-cial to get information in our standardized data interoperability le format as JSON data but in the latter, Rails is used instead of React in UI to save the data permanently which is most advantageous and desirable in our case.

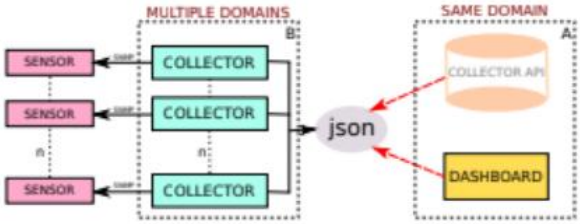

Fig. 3: Schematic Diagram showing two di erent ways (path A and B) to access JSON le for moni-toring.

3.3 Dashboard

Since our dashboard falls in the category of web-browser based applications, Fig. 4, it is dis-played on a web page that is linked to our col-lector which at-a-glance provides the instance of a data set. This link between the dashboard and collector allows data to be constantly updated. It allows the user to view Key Performance In-dicators (KPIs) and other critical data without delving into the semantics of the source system that manages the detailed data.

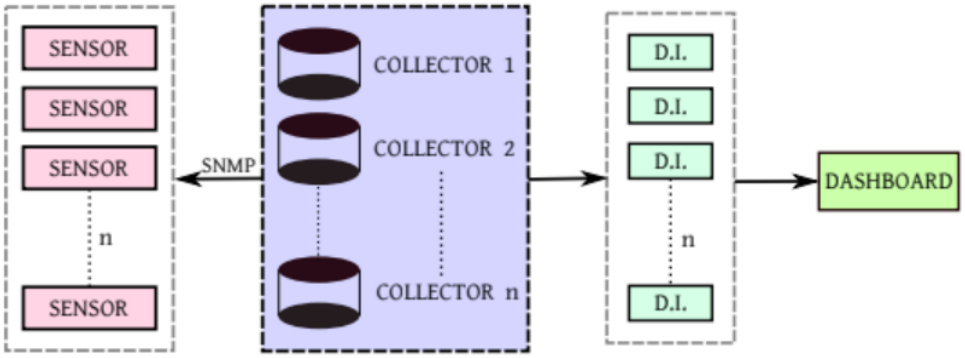

Fig. 4: Model-driven Dashboard Framework representing end-to-end Component Flow.

4. PROGRAMMING DESIGN

This section gives an insight on the program-ming design of the project by giving an overview of the languages used, their components and purpose, and logical centers.

Ruby on rails is used to design the col-lector as discussed in section 3.2. This dynamic and open source programming language fo-cuses on two key features - simplicity and productivity. The naturally read and easily written syntax makes it an extremely produc-tive web application framework along with rails. It includes everything needed to create a database-driven web application using the Model-View-Controller (MVC) pattern. The MVC principle divides the work of an applica-tion into three separate but closely cooperative subsystems- Model (ActiveRecord), View (Ac-tionView) and Controller (ActionController). The Controller facilitates querying the models for peculiar data from sensors on one side while systemizing that data into our standardized form that ts the requirement of a given view on the other, all within the application that is directing the tra c. The following piece of code is an example of a JSON le in our Standardized format.

Listing 1: A JSON le in our Standardized Format

```
...
structure:{[
item:{
value:    70
critical_min:      10
critical_max:      80
warning_min:      15
warning_max:      75
unit:      "Celsius"
},
]}
...
```

For creating our interactive UI, we have used a JavaScript library React. We have simple views for each state in our application, and React is e ciently updating and rendering just the precise components when our data modi es in collector. Its declarative view and Component-Based build makes the code more predictable, manages their own state and then compose them to UI. To make our application behave consistently and run in di erent en-vironments, we have used Redux which is a predictable state container for JavaScript apps.

As shown in Fig. 3, the dashboard can access data in two di erent ways. Possibility A is when dashboard and collector are in the same domain, and collector downloads the data from multiple sources and provide the result as an API. Possibility B is when dashboard and collector are on di erent do-mains, then dashboard accesses the data from multiple sources provided each source sup-ports Access-Control-Allow-Origin header. Axios is a Promise-based HTTP client for JavaScript which is used in the application to intercept request and response. It automati-cally transforms for JSON data.

The following piece of code is an instance representation of Axios performing a GET

request.

Listing 2: Axios performing a GET request

```
...
dispatch({type: "REQUEST_DATA"})
axios.get("...URL...",
{ crossdomain: true })
.then((response) =>{ dispatch({...})
...
```

The following piece of code is used in Ruby ActionController to grant access to the domain requested by Axios.

Listing 3: Ruby ActionController

```
...
response.set_header("Access-Control-Allow-Origin","...URL...")
render :json => @structure ...
```

5. CONCLUSION

Monitio allows users with little programming ex-pertise to carry-out data analysis in real-time. It makes performance monitoring easy to im-plement and allows disparate discovery within classi cation data. Hence, we are expecting expeditious application in context of develop-ment. It reduces the collection-analysis as well as analysis-reporting time by systematizing the process of generating KPIs for domain speci c datasets. The model we are employing is cover-ing many elements of the process starting with the standardization of collected data from vari-ety of sources and its display, the system users and their role, access privileges and grants of each user, information and content of each page view and the navigation process through these views. Since all the necessary code for sequen- tial organisation of the dashboard is automat- ically generated by our application, it removes the task of wearisome programming and reduces the delivery time of the solutions to apprecia- ble extent. Moreover, it is much easier to make amendments to the design of the dashboard as the changes are only needed to be made in higher level models which in turn, are automat-

6. FUTURE WORK

There are many opportunities for extending the scope of this application that remain. This sec-tion presents some of these directions:

- Extending and developing of Dashboard In-teroperability Standard.
- Data exchange optimization (JSON out-put).
- Adding Graphics and Timeline data items.
- Improving Dashboard Framework.

7. REFERENCES

[1] Yin Zhang, Meikang Qiu, Chun-Wei Tsai, Mohammad Mehedi Hassan, Atif Alamri, Health-CPS: Healthcare Cyber-Physical System Assisted by Cloud and Big Data, IEEE (2017).

[2] Wen YANG1, Syed Naeem Haider, Jian-hong ZOU, Qian-chuan ZHAO1Industrial Big Data Platform Based on Open Source Software, Advances in Computer Science Research (ACRS), volume 54, International Conference on Computer Networks and Communication Technology (CNCT2016)

[3] Vanderdonckt, Jean Model-Driven Engi-neering of User Interfaces: Promises, Suc-cesses, and Failures, (2014).

[4] P. Lubell-Doughtie, P. Pokharel, M. John-ston, V. Modi Improving Data Collection and Monitoring through Real-time Data Analysis, 3rd ACM Symposium on Com-puting for Development, Article. 28 (2013).

[5] Katrien Verbert, Erik Duval, Joris Klerkx, Sten Govaerts, Jos Luis Santos Learn-ing Analytics Dashboard Applications, Sage journals (2013).

[6] Kleppe, A., Warmer, J., Bast, W. MDA Explained: The Model Driven Architecture Practiceand Promise, Addison Wesley, Reading, MA (2003).

[7] Miller, J., Mukerji, J. (ed.): MDA Guide Version 1.0.1.Object Management Group, http: // www. omg. org/ docs/ omg/ically regenerating the code.03-06-01. pdf (2003).

[8] W. B. Frakes and K. Kang, Software Reuse Research: Status and Future, IEEE TSE 31, No. 7, 2005.

[9] Y. Yu, Y. Wang, J. Mylopoulos, S. Liaskos, A.Lapouchnian, J.Cesar Sampaio do Prado Leite: Reverse Engineering Goal Models from Legacy Code, ICRE, 2005.

[10] Grafana Labs. https: // grafana. com/