ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUBPL06

ENERGY CONSUMPTION MONITORING WITH GRAPH DATABASES AND SERVICE ORIENTED ARCHITECTURE

A. Kiourkos[†], S. Infante, K. Seintaridis, CERN, Geneva, Switzerland

Abstract

CERN is not only the biggest particle physics laboratory in the world but also a major electricity consumer. In 2018 alone, CERN consumed 1.25 TWh, equivalent to 1/3 the consumption of the canton of Geneva. Reliable monitoring of this consumption is crucial, not only for obvious operational reasons but also for raising the awareness of users regarding their energy utilization. This monitoring is currently done via a web based system, developed internally at CERN that is quite popular within the community. In order to accommodate the increasing requirements, a migration is underway that utilizes the latest technologies for data modelling and processing. The architecture of the new energy monitoring system with an emphasis on the data modelling, versioning and the use of graphs to store and process the model of the electrical network for the energy calculations is presented. The algorithms that are used are also presented and a comparison with the existing system is performed in order to demonstrate the performance improvements and flexibility of the new approach. The system embraces the Service Oriented Architecture principles and it is illustrated how these have been applied in its design. The different modules and possibilities are also presented with an analysis of their strengths, weaknesses, and integration within the CERN infrastructure.

MOTIVATION

Energy Management

Energy Management has become an essential element of operations management and allows the users to plan and make decisions based on the historical data about their energy consumption [1]. The management of energy in the industry and facilities like CERN is very context specific, as it largely depends on the process. This implies that energy management solutions from other industries cannot be easily copied. It is therefore important that users are able to get accurate, reliable and easily accessible information about the energy across the site and the different accelerator installations.

Current Solution & Data Flow

To accommodate the energy management needs, a web based application was developed at CERN (WebEnergy) more than five years ago, Fig. 1. This application extracts data archived by the SCADA system that is responsible for the monitoring of the electrical network. These are energy consumption measurements that are provided by the protection relays (IEDs) installed at the high voltage level of the electrical network. These measurements are then collected by the SCADA and subsequently archived to an internal long term data storage system.

Using APIs provided by the archiving system, WebEnergy pulls the measurement data and combines it with the electrical network model. The model has been defined and is maintained by the system administrator within the system. The application uses this combined data to calculate the energy consumption at the different levels of the network and for different consumers. The calculations occur daily with help of a scheduled task and with the system containing data up to the previous day.

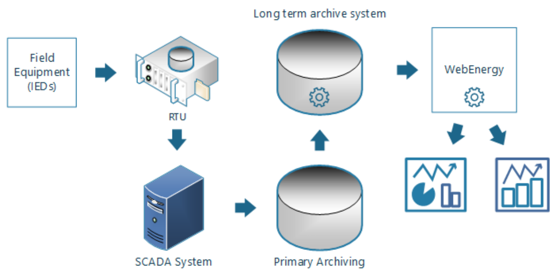

Figure 1: WebEnergy dataflow.

The users can access the previously calculated data via various dashboards where there are categorised and visualized in various charts.

Weaknesses

The application is used daily by numerous people at CERN. The results are quite accurate and have been validated over time against the energy consumption bills generated by the electricity suppliers.

Despite its success though, there are still opportunities for further enhancements and a number of additional features that will boost performance and add value for the users of the application and the organization.

The major area of improvement of the current application is the data model. Although simple to understand it is purely hierarchical, Fig. 2, allowing single parent-child relationships and most importantly: it is missing the notion of time.

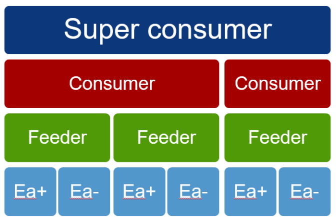

Figure 2: WebEnergy data model.

The electrical network in a facility like CERN changes continuously in order to accommodate the emerging needs. Storing only the energy consumption over time is not enough, the state of the model at any point in time is required in order to make meaningful comparisons. Because

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUBPL06

of the rigidity of the current structure, retrospective calculations or corrections are tedious, error prone and in many cases not feasible. Overall the current model leads to data loss with regards to the electrical network evolution over time, and is quite "expensive" to keep updated.

To address these issues several different concepts and technologies have been examined with an emphasis on open source technologies, maintainability and future proof design. An introduction on the most important ones and the ones chosen for the implementation of WebEnergy 2.0 is attempted in the sections that follow.

A GRAPH THEORY & GRAPH DATABASES PRIMER

History

Graphs have existed for centuries with the most famous example dating back to 1736. It was when Leonhard Euler solved the "Seven Bridges of Königsberg" problem. In brief, the problem in question was whether it was possible to visit all four areas of the city connected by seven bridges, while only crossing each bridge once. This lead to the groundwork of graph theory and its mathematics by Euler [2].

Basic Concepts & Terminology

Graphs and their use in graph databases consist usually of the following four building blocks:

Nodes or Vertices – These are the objects that make up the graph, the "nouns" in object oriented terms.

Relationships or Edges – These are links between the nodes, the "verbs" that give context to the nodes.

Labels – Labels are a mechanism to logically group nodes.

Properties – These are attributes that can contain a variety of data types providing the state of the model.

Graph Types

Graphs exist almost everywhere and they come in different shapes and sizes. Although in classic graph theory nodes have one relationship between nodes, most real-world graphs have many relationships and can contain even self-referencing edges [3], Fig. 3.

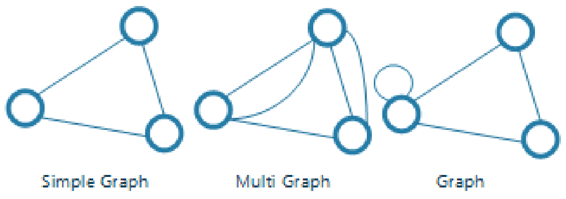

Figure 3: Graph types.

Table 1: Graph Types

Graph properties	Description
Connected	A path between any two in the graph exists
Disconnected	Nodes many not have a path between them
Weighted	The relationships between the nodes contain domain specific values
Unweighted	The relationships between the nodes do not contain any values
Directed	The edges have direction
Undirected	The edges do not have a direction
Acyclic	A graph with no cycles (starting and ending in the same node)
Cyclic	A graph with cycles
Sparse	A graph with few edges
Dense	A graph with many edges
Monopartite, bipartite or k-partite	Whether nodes connect to one or many other node types

Graph Algorithms

The number of graph algorithms is vast [4] and it would be impossible to list them all here but they can be very broadly grouped into the following four categories:

Graph search algorithms – Algorithms used for traversing the graph either for searching a specific item or general exploration (e.g. Breadth First Search, Depth First Search).

Pathfinding algorithms – Algorithms that allow you to find the optimum path between nodes (e.g. Dijkstra's algorithm).

Graph centrality algorithms – Algorithms to identify the most important nodes within a graph and their impact on the overall network (e.g. PageRank, Eigenvector Centrality).

Community detection algorithms – Detection community formation in networks (e.g. Louvain algorithm, Balanced Triads).

Graph vs Relational Databases

Relational databases (RDBMS) have existed for decades and have been the workhorse of the industry for storage in data oriented applications. Their value is undisputed with features like concurrency control, transaction management, data integrity mechanisms, common query language (SQL), abundance of tools, frameworks and expertise.

In the recent years though, due to the size, speed and complexity of the data, the relational model has shown its weaknesses that are particularly evident when complex interconnected data is involved. Furthermore, the schema based data model sets limits on how the data will be stored, requiring extensive normalization and filtering in order to handle the vast amounts of unstructured data that is generated today [5].

RDBMS use foreign keys to mark relationships between tables. This is fine for "shallow" relationships but when "deep" relationships are involved the limitations of join operations become apparent, with performance and complexity implications. Foreign keys are not "real relationships" as context information missing, Fig. 4.

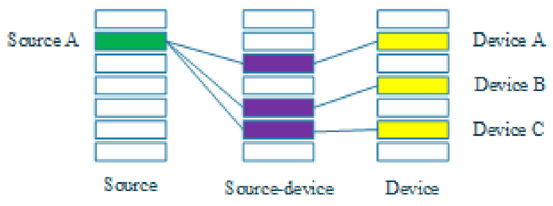

Figure 4: Relational model for capturing connections.

In comparison graphs databases are more performant with connected data, Fig. 5. The relationships in graph database systems are first class citizens and their schema-less approach allows to move very quickly from the drawing-board to the database without elaborate normalising steps.

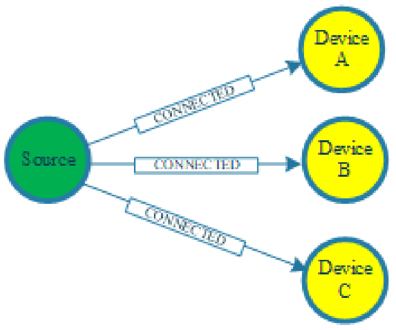

Figure 5: Graph model for capturing connections.

There are two prevalent models in the realisation of the graph in modern graph databases, the Labelled property graph and the Resource Description Framework [6]. Description of these models is beyond the scope of this paper but it is sufficient to say that the Labelled Property graph is the most popular form of graph model and is used by the most popular graph database currently, Neo4j [7].

SERVICE ORIENTED ARCHITECTURE & MICROSERVICES

Definition

Service Oriented Architecture (SOA) is a software architecture style that aims to achieve loose coupling between interacting software components and reusability in different contexts. This takes the form of services that are provided by the different application components. These components usually communicate via a specific communication protocol over the network [8]. One of the defining principles of a service in SOA is that it owns the data under its responsibility and operations on that data.

The concept is not new and has existed in many forms over years, some examples are:

- Web services
- OPC-UA
- gRPC
- Messaging (ActiveMQ, JMS etc.)

The main focus of SOA systems has been the loose coupling between the components and in order to accomplish this in many occasions some type of communication bus is involved, Fig.6. This is called an Enterprise Service Bus (ESB).

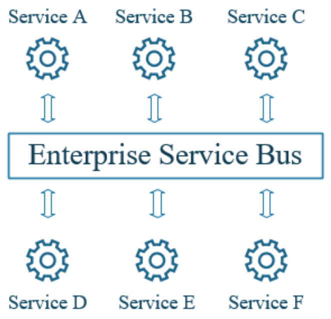

Figure 6: Service Oriented Architecture.

Microservices

The latest architectural trend in software systems is microservices. The concept can be considered an evolution of the SOA with the main differences being the communication patterns, granularity of the services and stronger embrace of cloud technologies that are now ubiquitous [9].

Microservices communicate directly with each other using lightweight protocols and messaging. This allows to have separate release cycles between services and different teams working independently with clearly defined interface points, Fig.7. This modularization aims to also improve reusability and lead to shorter release cycles allowing to deliver new features in shorter times. On the other hand the presence of a greater number of "moving parts" within a microservice based system increases the operation overhead and makes integration testing more complicated.

Regarding the granularity of microservices there is no universal consensus and it is highly dependent on the subject domain. In general the service should not be too small so that the runtime overhead complexity exceed its benefits.

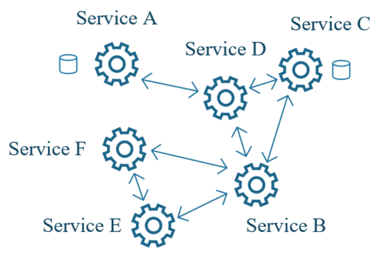

Figure 7: Microservices Architecture.

WEBENERGY 2.0

Design Goals & Architecture

The main goals that were set for the design of the new system have been:

- Increase data model flexibility so that different versions of the network model can co-exist.
- Introduce the notion of time in the network model so that past calculations can be matched to the corresponding layout of the network.
- Split the system into clearly defined services to allow to mix different technologies in order to benefit from different frameworks and tools.
- Explore the possibility of power calculations.
- Introduce live energy monitoring consumption (last 15 ~ 30 minutes) compared to only the previous day of the current system.
- Explore energy consumption forecasting.
- Integrate virtual invoicing to make the users aware of their energy usage.

The architecture of WebEnergy 2.0 has been designed around separate services that collaborate and make the complete energy management system. The layout of these architecture and the interaction between the different services is explained in the sections that follow.

Hybrid Data Management

WebEnergy 2.0 uses two data management paradigms, relational and graph. The redesigned data model with the support of versioning is stored in a relational database along with all other application data (settings, statistics etc.), and as it is demonstrated later is exposed to the different services of the system via a dedicated REST API.

The graph database (Neo4j) is used as an ephemeral data store, where the different versions of the model are loaded in order to utilise its powerful traversal algorithms and perform the energy consumption calculations quickly and efficiently.

This approach allows to utilise existing infrastructure for availability and data consistency and at the same time use more modern approaches for the data handling and calculations.

Services

The services have been designed to be self-contained, represent a specific business entity and act as black box for the consumers of the service, Fig. 8.

Data REST API – With the use of Spring Data REST, this service exposes the different tables of the database as REST endpoints following the HATEOAS principles. The hypermedia format supported by Spring Data REST is HAL. These endpoints will be used by the different services to access the relational database.

Data Sanitization Service – This service is responsible for the detection of abnormalities in the incoming measurement data from the devices. It can happen that there is a faulty equipment reporting erroneous measurements or an IED that is replaced which causes its internal energy counter to be reset. These events impact the correctness of the energy consumption calculation and it is important that they are filtered out.

Data Import Service – The data import service is for importing the raw data from the long term archiving system and/or HDFS. The new long term archiving system (NXCALS) is heavily based on Apache Spark and at the same time the archive data from the SCADA archive database is imported daily into HDFS as parquet files. This implies that the data can be imported via two sources, as required, either from the long term archive storage or directly via HDFS. The service will be able to handle both data sources.

Graph & Calculation Service – The existing WebEnergy application uses iterative algorithms for the calculation of the energy consumption at the different levels of the network model. This although simple to implement, suffers in terms of performance.

During the investigation for the new architecture, a graph database (Neo4j) was examined as the engine for the energy calculations. Neo4j supports user-defined functions that can be used to take advantage of it power graph engine to offload tasks to the graph database [10]. The electrical network model consists of highly interconnected data, thus suitable use case for a graph database.

We have developed a user function inside Neo4j that uses recursion to calculate the energy consumption in various versions of the network model. The tests showed that for any version of the network, with approximately 5000 nodes and 10000 relationships, the computation of the energy consumption for the complete model takes less 20ms. This gives the possibility to the users to perform on the fly calculations for any version of the network model.

The graph and calculation service is responsible for the following tasks:

- Loading the model from the relational database and import to Neo4j for the energy calculations.
- Retrieve the results for the calculation of the different versions of the model from Neo4j and save to the database or serve them to the Administration or User service for displaying in the Administrator or User Interface.
- Receive alternative representations of the model from the administrative service and import to Neo4j for the energy calculations.

Administration Service – The Administration Service takes care of the functionalities related to the management of the model, like:

- Modifying the existing network model
- Creating different versions of the model
- Launching calculations

User Stats service – The User Stats service is responsible for retrieving information from the user interface of the application and store usage data via the REST API to the database. This enables the analysis in order to further understand how the application is utilised by the users, identify potential problems and in general provide relevant usage analytics

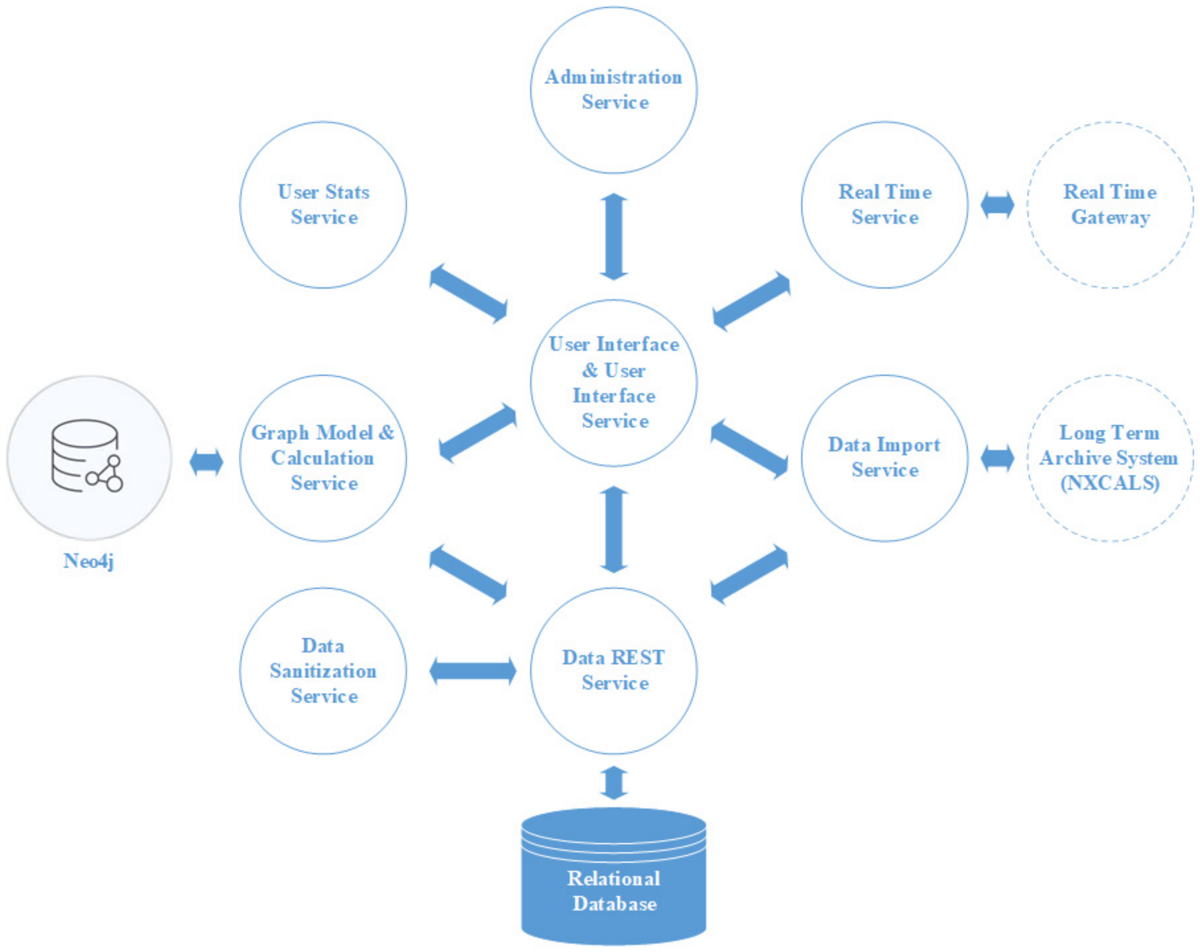

Figure 8: WebEnergy 2.0 Architecture.

Real Time Service – This service is responsible for calculating the status on network model in terms of energy consumption using real time telemetry data.

This allows the visualisation of the current energy dataflow within the last hour or less.

The service will be implemented using the actor model [11] where every actor is responsible for maintaining the status of the different nodes in the network model. The service exposes http endpoints for the monitoring, control and data feed for the visualisation of the current values

User Interface Service - The user interface service shall be responsible for serving the user interface of the system. This would be equivalent to an API gateway in microservices terms. It will be implemented using Spring Boot and a number of REST endpoints shall be exposed for monitoring, control and serving the front end.

CONCLUSIONS

The development of WebEnergy 2.0 is underway with some services of the application completed and others being developed. The microservices approach has provided benefits in terms of independent work streams and early delivery, as some services are operational without the need for the whole system to be in place. In addition the modularity of the codebase has enabled us to introduce new members to the project in less time when compared to monolithic applications where the new members require significant time to get acquainted with large codebases. The additional operational overhead has also become visible as additional infrastructure is required to deploy and monitor these services. The hybrid approach for the data model (relational + graph) has allowed us to use the best of both worlds and address the shortcomings of the previous data model.

REFERENCES

[1] K. Vikhorev, R. Greenough, and N. Brown, "An advanced energy management framework to promote energy awareness," Journal of Cleaner Production, vol. 43, pp. 103-112, 2013/03/01/ 2013, doi: https://doi.org/10.1016/j.jclepro.2012.12.012.

[2] I. Gribkovskaia, Ø. Halskau Sr., and G. Laporte, "The bridges of Königsberg—A historical perspective," *Networks, vol. 49, no. 3, pp. 199-203, 2007,* doi: 10.1002/net.20159.

[3] M. Needham and A. E. Hodler, Graph Algorithms: Practical Examples in Apache Spark and Neo4j. O'Reilly Media, 2019.

[4] "Graph algorithms." `https://en.wikipedia.org/wiki/Category:Graph_algorithms` (accessed 2019/08/21, 2019)

[5] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen, and D. Wilkins, A comparison of a graph database and a relational database: A data provenance perspective. 2010, p. 42.

[6] R. Angles and C. Gutierrez, "Survey of graph database models," *ACM Comput. Surv., vol. 40, no. 1, pp. 1-39, 2008*, `doi: 10.1145/1322432.1322433.`

[7] "Graph DB Engines ranking." `https://db-engines.com/en/ranking/graph+dbms/` (accessed 2019/08/21, 2019).

[8] H. He, "What is service-oriented architecture," *Publicação eletrônica em*, vol. 30, pp. 1-5, 2003.

[9] N. Dragoni et al., "Microservices: Yesterday, Today, and Tomorrow," in *Present and Ulterior Software Engineering*, M. Mazzara and B. Meyer Eds. Cham: Springer International Publishing, 2017, pp. 195-216.

[10] Extending Neo4j with User-defined functions. `https://neo4j.com/docs/java-reference/current/extending-neo4j/procedures-and-functions/functions/` (accessed 2019/08/21, 2019).

[11] P. Haller, " On the integration of the actor model in mainstream technologies: the scala perspective," in Proceedings of the 2nd edition on Programming systems, languages and applications based on actors, agents, and decentralized control abstractions, 2012: ACM, pp. 1-6..