GreyCat: A Framework to Develop Digital Twins at Large Scale

Francois Fouquet, Thomas Hartmann, Cyril Cecchinel, Benoît Combemale

GreyCat: A Framework to Develop Digital Twins at Large Scale

Francois Fouquet
DataThings
Luxembourg, Luxembourg
francois.fouquet@datathings.com

Thomas Hartmann
DataThings
Luxembourg, Luxembourg
thomas.hartmann@datathings.com

Cyril Cecchinel
DataThings
Luxembourg, Luxembourg
cyril.cecchinel@datathings.com

Benoit Combemale
University of Rennes
Rennes, France
benoit.combemale@irisa.fr

ABSTRACT

Digital Twins (DTs) have become a pivotal technology for enhancing the understanding, monitoring, and ultimately autonomous piloting of systems across various domains, including large-scale critical infrastructures such as smart electricity networks. The development of DTs necessitates developing diverse services that utilize different models and a digital shadow, which encompasses both real-time and historical data from the physical counterpart. The extensive scale of large infrastructures presents significant challenges, including managing numerous parameters, heterogeneous data, and the complex computations required, particularly with the increased use of AI algorithms. Current technologies, built by stacking multiple databases and using general-purpose languages, are inadequate for efficiently implementing digital twin services that need runtime reactivity. This tool demonstration paper introduces GreyCat, a framework designed for the development of digital twins over large-scale digital shadows. GreyCat combines imperative object-oriented programming, database persistent indexes, and scalable memory management to facilitate the creation of comprehensive and efficient digital twins. We demonstrate the ease use of GreyCat through simple examples and showcase its effectiveness in constructing the national digital twin of Luxembourg's electricity grid, which is currently operational and managing billions of data points. Reflecting on the development of GreyCat over the past years, we discuss the main lessons learned and identify open questions for future digital twin development frameworks.

CCS CONCEPTS

• **Software and its engineering → Development frameworks and environments**; • **Information systems → Data management systems**; **Graph-based database models**.

KEYWORDS

Digital Twins, Digital Shadow, Development Framework

MODELS Companion '24, September 22–27, 2024, Linz, Austria

ACM ISBN 979-8-4007-0622-6/24/09
https://doi.org/10.1145/3652620.3688265

ACM Reference Format:
Francois Fouquet, Thomas Hartmann, Cyril Cecchinel, and Benoit Combemale. 2024. GreyCat: A Framework to Develop Digital Twins at Large Scale. In *ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion '24), September 22–27, 2024, Linz, Austria.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3652620.3688265

1 MOTIVATION AND REQUIREMENTS

GreyCat's origin lies in the development of large-scale DTs handling smart infrastructures, such as for power grids but also water and transportation network. These DTs need to aggregate data from various different systems, e.g., geographical information systems *(GIS)*, enterprise resource planning *(ERP)*, but also supervisory control and data acquisition *(SCADA)*. Additionally, measurements from the field must be considered, such as consumption and production data from smart meters, and weather data. This leads to terabytes and hundreds of billions of data points that need to be consolidated. DTs should enable simulations to test any change before physical application, often using prediction [5] or discrete models. Staying at the example of power grids, this can be consumption and production forecasts [5] or PowerFlow solver. Performance is crucial considering that the time to react can lead to physical damage, which often means reaction of seconds or below. Building such DTs is challenging due to the amount and diversity of data that must be synchronized, processed, and stored in the digital shadow. We argue that this complexity comes from the number of data transformations and mappings required to turn raw data into actionable insights. For instance, if implemented with multiple databases and transformation code, such use cases can quickly lead to a large number of queries, often in various languages, making the DTs both hard to maintain and slow. To solve performance issue, similarly to PLSQL our approach follows a simple concept: code as close as possible to the data storage. Nevertheless to keep the expressive object-oriented programming foundation we based our execution on a multi-paradigm graph database and an imperative language. Complementary to solutions like Azure Digital Twins, AWS IoT TwinMaker, or Eclipse ditto which have a strong focus on data definition, we focus on runtime scalability and speed of computation. Our manifesto is build around the following rationales:

No query. But an evolving graph structure that you traverse [4].
No mapping. Instead, a single model from disk to API.
Any scale. The hardware defines the speed not the dataset limits.
Any data. Link everything, be it time, geo, tabular, or vector data.

Any branches. What-if [3] scenario as copy-on-write branches.

The remainder of this paper is as follows. Section 2 presents an overview of GreyCat's architecture, sections 3 and 4 discuss and exemplify challenging characteristics of targeted DTs, before section 5 and 6 discuss perspectives and conclude the paper.

2 GREYCAT OVERVIEW

At its foundation, GreyCat is built over a dynamic, evolving graph structure [2], where nodes serve as containers that can store various properties. Node properties are designed to handle potentially large time series or geographical data. GreyCat also defines an executable language that allows to describe types, traverse and mutate graph nodes with an imperative syntax. This language is depicted in 2.1, the architecture in 2.2, and the memory management in 2.3.

2.1 Language Overview

Like any DDL (Data Description Language) or OOP (Object-Oriented Programming language), GreyCat enables the description of types and associated attributes. We use a JSON-like syntax together with the keyword **type**. Graph specifics are covered using nodes as first-class citizens and single, multiple, indexed, temporal, and geo relationships using the respective keywords: *node*, *nodeList*, *nodeIndex*, *nodeTime*, and *nodeGeo*.

```
type Measurement { index: int; voltage: float; }
type SmartMeter {
    id: String;
    location: geo;
    values: nodeTime<Measurement>?;
    child: node<SmartMeter>;
    fn predict(t: time): Measurement{ /*...*/ }
}
```

GreyCat objects are then instances of a defined type, such as:

```
var measure = Measurement { index: 3,voltage: 231.12 };
var meter_a = SmartMeter { id: "meter_XY" };
var meter_b = SmartMeter { id: "meter_XZ" };
```

To be persisted, objects must be attached to a node container.

```
var node_meter_a = node::new(meter_a);
var node_meter_b = node::new(meter_a);
node_meter_a.child = node_meter_b;
```

Similarly, objects can be attached to a node and time-point such as:

```
meter_b.values = nodeTime::new();
meter_b.values.setAt(time::now(), measure);
```

For graph traversals, we use the -> operation such as:

```
var values_resolved = node_meter_a->child->values;
```

Finally, the *@expose* annotation transform any function to an API.

```
@expose
fn sayHello(name: String){ return "Hello ${name}";}
```

2.2 Architecture Overview

The GreyCat Language is compiled into a program and deployed and executed within a GreyCat runtime instance. An instance plays the role of a database, a virtual machine, and an API server in one. This architecture is depicted in Figure 1. A GreyCat program is composed of **symbols**, **types** and **opcodes**. Symbols are a dictionary of all names for types or fields. Types are collections of structures, composed of name and type fields. Opcodes, like Java bytecode are sequences of operations based on source code compilation. On every

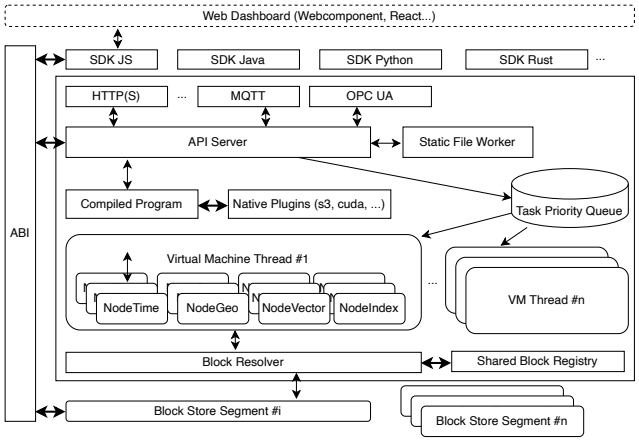

Figure 1: High Level architecture of GreyCat

program update, the GreyCat instance encodes types and symbols into a structure named ABI (Application Binary Interface). Similarly to Avro schema, the ABI is stored alongside the data segments and aims to capture all structural information to reduce data store and API payloads. We offer SDKs that exchange the ABI during an handshake step to avoid any transfer overhead, as we it for example with JSON derivatives. GreyCat's internal store also leverages the ABI to transform memory structures into compact binary blobs, stored on hard drives. Compiled programs are augmented by native plugins to allow to expose native libraries as GreyCat functions. Among others we can cite *algebra* or *ONNX* plugins that enable GreyCat to manipulate machine learning models. Compiled programs also contain the list of exposed functions accessible through an API server. Various transport protocols are implemented to reach this API server such as HTTP, but also OPC-UA, which is commonly used in Industry 4.0. To cope with the complexity of multi-thread operations over the graph structure, GreyCat follows an architecture pattern named share nothing, commonly used in some DBs. This imposes to avoid data exchange between threads during request execution. Requests are first inserted into a shared waiting queue and then processed by any available thread. The execution is then done on a snapshot of the database making the execution of the code comparable to a transaction for developers. On success, updated block addresses are merged into a block registry.

2.3 Memory Management

Handling datasets greater than memory capacity is a major issue for many processing technologies. To ensure scalability, software engineers usually have to split, load and unload data chunks and with this increase the number of data mappings. We argue that a DT framework should be able to process any size of data regardless of the hardware capacity. The more powerful the hardware, the better the performance, but processing should not crash. Our framework proposes an automatic management of memory to disk mapping by using a strategy based on automatic reference counting. This method, compatible with massive heap size, unlike garbage collector-based approaches, mark and unmark objects based on program frames. The downside of this is the risk of inter-references

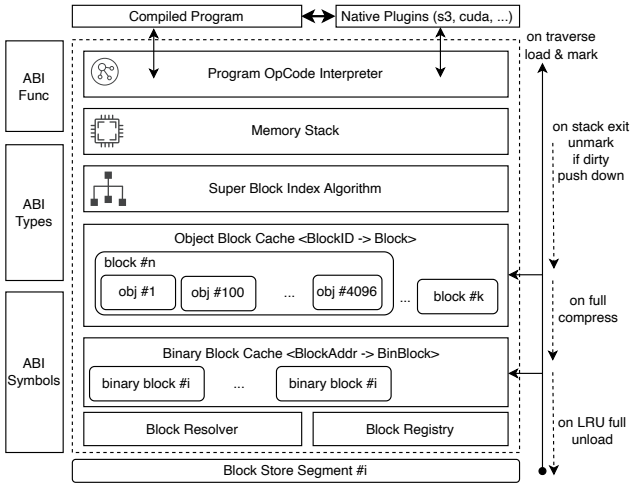

Figure 2: GreyCat Worker Architecture

that we avoid forcing all objects to be contained by a single node similarly to C++ smart pointers. Objects are usually serialized in few bytes making way to small as unit to be store as disk packet since most of modern operating systems requires 4096 bytes minimum packet size. Our SBI algorithm packs objects into bigger blocks with the objective of hosting objects that are likely used together. Marks are propagated to the host block level. When the count reaches zero, the block is sent to the various level of caches. Graph traversal operations are then translated to block resolutions. Nevertheless, considering disk versus memory latency, block read operations must be minimized using caching techniques. To be stored on disk, blocks are compress and serialized into a binary blob. This binary version of a block is an order of magnitude smaller than it's memory counterpart. For this reason, every GreyCat thread leverages a dual layer cache both using a Last Recently Used (LRU) strategy to reflect code locality of object usage depicted in figure 2. Blocks are always store into segments, which can be a disk partition or a file. Whenever GreyCat faces a traversal operation, the object block cache is queried with the block_id to find an existing copy in memory. On success, the block is marked and moved to the LRU head, on miss, a block registry translates the block_id into a block_addr on disk and the binary cache gets queried. If key is absent, the binary block is loaded from disk. In any case, the binary block is moved to the binary cache LRU head, decompressed, inserted into object cache and marked. When the Object Block Cache LRU is full, the tail is either dropped, in case the page is unmodified, or compressed and inserted to Binary Block Cache. When the Binary Block Cache is full the block is unloaded and written to disk using ABI encoding and the newly created address is reported to the block registry.

3 CHARACTERISTICS OF THE RESULTING DIGITAL TWINS

In this section, we characterize the digital twins developed with GreyCat using the 21 characteristics outlined in the systematic reporting framework for DTs proposed by Gil et al. [1]. While the specifics may vary on the DT created, we here capture the common characteristics of the resulting DTs developed with GreyCat. GreyCat has been designed specifically for managing large-scale infrastructures that integrate various assets, connected to distribution networks (e.g., electricity, water, transportation) with real-time field measurements and geographical systems describing static information. The physical twins typically involve remotely controllable devices (e.g., on-off) or PID-based controllers, and real-time field sensors. Real-time telemetry data extraction feeds monitoring services, and simulations calibrated according to the current state of the system can be conducted to generate a series of events for the physical twin. Consequently, the DT services primarily include monitoring, deviation detection, and what-if simulations powered by AI-based predictors. These services operate on different timescales, ranging from high-fidelity simulations requiring significant computation to real-time services using fast simulations, machine learning and surrogate models. The underlying digital shadow is constructed as a holistic graph with time-series data capturing both historical and simulated future states, used throughout the physical twin's lifecycle for continuous design, monitoring, and optimization. DT models are described using the GreyCat language, encompassing both data types and executable code based on native libraries for AI and scientific computing. GreyCat supports the development of web-based dashboards (e.g., maps) for decision-making, using either real-time views or simulated what-if scenarios for anticipation. GreyCat also includes an API server and message bus connectors for real-time data collection. The resulting DTs offer specific APIs for connecting to a wide range of technologies used in physical twins (e.g., HTTP, Kafka, MQTT, OPC-UA), often dictated by the application domains. Additionally, APIs are available to link the digital shadow to various services, either developed with GreyCat or integrated into the digital twin. Data access is controlled at the API level through access control mechanisms that support permission-based data quality level access. Privacy considerations must be addressed leveraging those permission by GreyCat framework users. GreyCat is designed to run on standard hardware and resulting DTs can be deployed on cloud but also on-premises to ensure sovereignty of disconnected DTs. DT validation is primarily achieved through cross-validation of live measurements and model drift detectors. Currently, GreyCat does not provide specific support for DT integration beyond the aforementioned APIs. While the resulting DTs adhere to the standard ISO23247 [6], future standardization of various interfaces would enable the reuse and sharing of DT building blocks, thereby simplifying the engineering of digital twins, including the cross-fertilization of tools for DTs.

4 ILLUSTRATIVE EXAMPLES

We highlight two case studies - one small-scale and one large-scale - successfully deployed using GreyCat.

4.1 Monitoring Tree Roots

Trees must be carefully watered to ensure roots development especially in harsh urban areas. Tensiometry sensors and microdendrometry sensors enable measurement of roots development versus ground humidity. With the company Urbasense, we have developed a DT ingesting those sensors data together with weather stations

data to monitor tree health and trigger watering devices. Implementation takes less than 2.5k lines of code *(including watering date predictor and root growth KPIs)* which illustrate the effectiveness of direct graph traversal to cope with geographical-temporal statistics. Collecting data from 9966 trees and 743 weather stations and hosted on a rather pool of small 4GB RAM virtual machines for high availability, this DT handle 80681 daily keeping latency below 7ms for monitoring queries and below 70ms for predictive ones.

4.2 Alva

Alva is a DT for electricity grids developed exclusively with GreyCat. Alva unifies, visualizes, and analyzes data in real-time and helps distribution system operators to monitor, forecast, simulate, and plan their networks more efficiently. The user interface of Alva is depicted in Figure 3. Deployed in Luxembourg on the scale of the whole country, Alva ingests yearly more than 50 billion time series data coming from more than 350.000 smart meters, computes the power flow of around 320.000 cables continuously, and monitors tens of thousands of grid elements in real-time. In addition, more than 1.2 million distributed machine learning models profile and forecast consumption and production around the country. A simulation engine orchestrating forecasting models and real-time data allows to simulate grid failures but also expansion of heat pumps, PV cells, or e-mobility. Alva illustrates perfectly the challenges of developing large scale digital twins. We have a large amount of data coming from different sources and data of very different kind, from time series data over relational, i.e., graph-based data to machine learning models. Moreover, this data changes over time and at very different frequencies. Some structural elements might only change once in a while, while other data, such as sensor measurements, might change very frequently. All of this data needs to be seamlessly integrated into one model and it must be easy to query (or navigated) this data. For example, in order to computer the power flow, we first need to explore the electricity grid to evaluate which cables are fed by a certain transformer. It is important to note that this can change over time. Meaning, this is basically a breadth (or width) first search in a temporal graph over a massive data set. Once those cables are detected we have to query the time series of the consumption and production values of every household connected to those cables in order to finally solve the power flow equations. Such queries are very difficult to implement in existing frameworks but shows the strength of GreyCat, where we can simply navigate in the temporal graph structure. As a result, Alva complex queries are solved in less than 400 hundred lines of code and resulting HTTP API stay lower than 20ms latency.

5 OPEN QUESTIONS AND PERSPECTIVES

At DataThings, we had the opportunity to deploy many GreyCat-based DTs for various domains. Drawing on our experience, we have found that computation performance of such platform is key for adoption. Numerous optimization where necessary to bump from hours to milliseconds response time for Alva, nonetheless it remains mandatory to be considered in a user decision process. Hosting hardware technology also need to be carefully selected when it come to scale a DT, for example when selecting storage

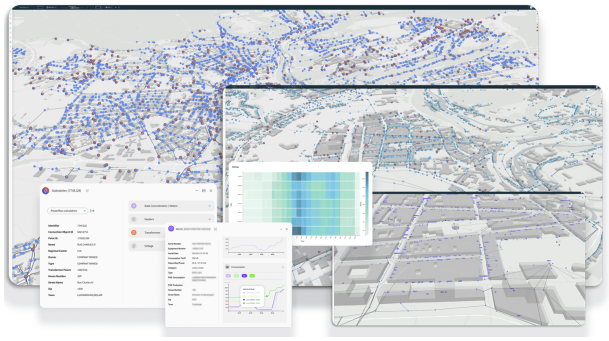

Figure 3: Alva: a digital twin of electricity networks

technology such as NVMe disks. Our expertise also led us to conclude that establishing a legal framework to centralize data under a single stakeholder can be a significant barrier. Instead, we must transition from the holistic approach enabled by GreyCat to a federated model where each stakeholder maintains control over their data and models. For instance high voltage and low voltage networks are usually different stakeholders yet requires to federate APIs to enable the depicted grid simulation of Figure 3. Our SDKs and ABI can be technical foundation towards such goal but such federation must be introduced at the modeling language for the same reasons that motivates our manifesto.

6 CONCLUSION

In this paper we presented a framework that tackles the challenges of heterogeneous data processing at scale by blending a graph store and an imperative programming language. Using object blocks and disk-to-memory management inherited from database techniques, we tackle the deployment of national-wide DT for electricity grid.

REFERENCES

[1] Santiago Gil, Bentley Oakes, Claudio Gomes, Mirgita Frasheri, and Peter G. Larsen. 2024. Towards a Systematic Reporting Framework for Digital Twins: A Cooperative Robotics Case Study. *SIMULATION* (2024), 1–27.

[2] Wentao Han, Youshan Miao, Kaiwei Li, Ming Wu, Fan Yang, Lidong Zhou, Vijayan Prabhakaran, Wenguang Chen, and Enhong Chen. 2014. Chronos: a graph engine for temporal graph analysis. In *Proceedings of the Ninth European Conference on Computer Systems*. 1–14.

[3] Thomas Hartmann, Francois Fouquet, Assaad Moawad, Romain Rouvoy, and Yves Le Traon. 2019. GreyCat: Efficient what-if analytics for data in motion at scale. *Information Systems* 83 (2019), 101–117. https://doi.org/10.1016/j.is.2019.03.004

[4] Thomas Hartmann, Francois Fouquet, Gregory Nain, Brice Morin, Jacques Klein, Olivier Barais, and Yves Le Traon. 2014. A Native Versioning Concept to Support Historized Models at Runtime. In *Model-Driven Engineering Languages and Systems*. Springer International Publishing, Cham, 252–268.

[5] Thomas Hartmann, Assaad Moawad, Francois Fouquet, and Yves Le Traon. 2017. The Next Evolution of MDE: A Seamless Integration of Machine Learning into Domain Modeling. In *2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 180–180. https://doi.org/10.1109/MODELS.2017.32

[6] ISO23247-1:2021 2000. *Automation systems and integration — Digital twin framework for manufacturing*. Standard. International Organization for Standardization.

A DEMONSTRATION OUTLINE

GreyCat development environment as well as the runtime and documentation can be downloaded at **https://greycat.io**. Alva product presentation and especially visual demo showing the simulation features are available at **https://datathings.com/alva**.