



**POLITECNICO**  
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE  
E DELL'INFORMAZIONE

EXECUTIVE SUMMARY OF THE THESIS

## A model-based methodology to define data-intensive architectures

LAUREA MAGISTRALE IN ENGINEERING OF COMPUTING SYSTEMS- INGEGNERIA INFORMATICA

**Author:** ARIANNA DRAGONI

**Advisor:** PROF. ALESSANDRO MARGARA

**Co-advisor:**

**Academic year:** 2022-2023

---

### 1. Introduction

In recent years, we have witnessed extraordinary advancements in the field of big data management and analysis. These advancements have driven the adoption of increasingly sophisticated and complex infrastructures aimed at extracting information more efficiently from massive volumes of data. However, we can notice that these infrastructures exhibit significant diversity in their requirements, including but not limited to the types of data to be analyzed, processing methods, and data storage durability. Furthermore, it is essential to consider that the same infrastructure may need to interface with a variety of different stakeholders, both data producers and consumers, each with unique and specific needs. This diversification of stakeholders introduces the complexity of managing and handling data in highly customized ways to meet the specific demands of each group. The wide range of requirements and the heterogeneity of infrastructures result in highly diversified data-intensive architectures, each designed to address specific challenges. Consequently, there arises the need to integrate various solutions and systems to fully meet all the requirements.

This complexity underscores the necessity for a *reference architecture*. This term refers to an

architecture that abstracts away from a concrete infrastructure, describes a class of infrastructure, and can be used to design concrete architectures for different infrastructures within this class[2]. Many reference architectures have been presented in the literature. By following different research approaches, [4] and [2] have both introduced reference architectures that unify the various data life-cycles that can be implemented within an infrastructure. They both conceptualize the life-cycle as a sequence of phases, starting with data acquisition and continuing with data processing to generate results. However, their reference architectures primarily delve into functional requirements, thus focusing mainly on extracting and showcasing data processing characteristics of different tasks. The data management aspect is confined to determining whether data is stored in persistent or temporary storage, and it primarily concentrates on non-functional requirements directly related to data processing. Aspects such as transactional requirements are not thoroughly examined in these works. [3] introduce a model for data-intensive systems that consolidates crucial design and implementation decisions into a cohesive framework. They take into account both data processing and data management systems in data-intensive scenarios. However, they do not discuss architectures

for integrating these systems to define a complete infrastructure. In the reference work [1], numerous reference architectures are presented for various architectural paradigms, including Lambda, Kappa, and others. The analysis assesses how each of these architectures addresses a diverse range of requirements, both functional and non-functional. However, it is noteworthy that there is no single comprehensive reference architecture that encompasses all the necessary components to represent any of the mentioned reference architectures.

In the vast body of literature dedicated to software engineering for big data architectures, various models have been developed to address specific requirements. These models provide structured approaches to integrate processing and storage systems for big data to meet specific needs. These models enable developers of concrete architectures to navigate the choice of the types of systems to use. A significant example in this context is the Lambda architecture, which models infrastructures where there is a need to address both throughput requirements, i.e., the speed at which data is processed and made available, and complex processing requirements[1]. However, despite the numerous models and frameworks proposed for engineering data-intensive architectures, there is currently no comprehensive model that encompasses all the characteristics of such architectures in a single structure. Furthermore, there is a lack of a definitive methodology that systematically identifies a set of components to be orchestrated to develop a concrete architecture. The choice of components and their orchestration depends on various factors, such as the nature of the data, processing requirements, available resources, and application objectives.

The main objectives of this thesis are

- to provide a model that attempts to encompass all the characteristics that a data-intensive architecture can exhibit and demonstrate how this model can be adapted to various existing architectures.
- to provide a methodology for creating application-specific data-intensive architectures. This will involve starting with the requirements previously identified by the architect and identifying a set of components that can be used to construct such architec-

tures.

## 2. Model

In this thesis, we introduce a model for data-intensive architectures, focusing on the components that make up these systems. The model describes the big data life-cycle within an architecture. Here, we define *components* as discrete software elements within the architecture, each assigned specific roles in data management and processing. The model encompasses potential components that could be integrated into a data-intensive architecture, along with their respecting characteristics. Subsequently, different architectures adapt to their specific use cases by *activating* the required components based on their needs.

Components are represented by a *phase* and a *storage*. The phase represents a step in the data life-cycle that the component implements and is directly activated with the component. The storage represents the component's objective to persist the data it produces. It can either be activated, in which case we define the component as *state-centric*, or not activated, in which case we define the component as *data-centric*. Components inherently possess various capabilities, referred to as *features* which are explicitly represented within each component. These capabilities can be delivered by a component through different mechanisms, a concept we refer to as *abilities* in terms of providing a specific capability. For each capability, different abilities can be activated to represent the application-specific architecture.

The final model, which is depicted in Figure 1, aims to represent data-intensive architectures by delineating the data life cycle they oversee through the orchestration of these components. The final architecture is constructed from a collection of these components, which all together define the data life-cycle within the architecture.

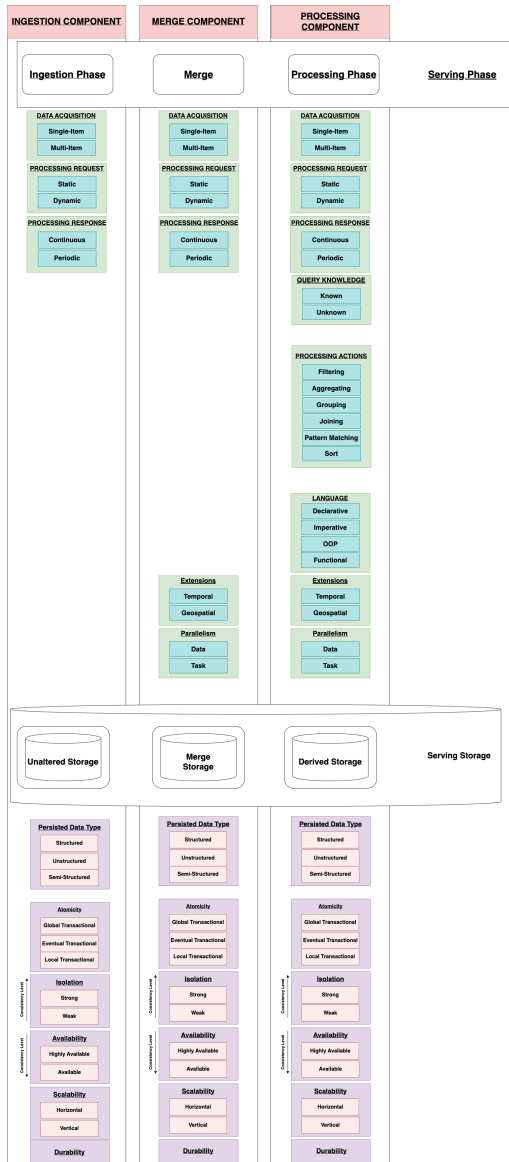


Figure 1: A Model for Data-Intensive Architecture.

We initially treat components as black boxes. The overall model consists of four components, each with its associated phase and optional storage

- The *Ingestion* component acquires data from external sources as it comes; it is composed of adapters that communicate with the outside world and convert the protocols used by the outside with what is in the system. The *Unaltered Storage* is used to store *raw* data as it comes from the outside, which doesn't necessarily have a defined format or schema, and it can store data in different formats.
- The *Merge* component acquires data from

the previous phase. It is present if and only if some union over the *raw* data coming to different sources is necessary to create *merged* data and facilitate the work that will be done by the following phases. The *Merged Storage* can persist *merged* data produced in this phase.

- The *Processing* component is responsible for doing processing actions over the *raw/merged* data in order to perform more in depth analysis and produce *derived* data. The *Derived Storage* stores the *derived* data that is produced in this phase.

The *Serving* component extends across the other components to indicate their capability to perform in-memory computation (for the phases) and in-memory storage and management (for the storage).

Components do not necessarily represent different components of the real system. For example, a single system can first transform raw data in *merged* data, and then can process *merged* data to obtain derived data. In the model, this is split in two phases: the merge phase and the processing phase.

In the context of component interactions, we have established two distinct types of flows. Firstly, there is the *Data-Flow* which enters the model from the left and proceeds towards the right. This flow facilitates the transfer of data from one component to another, and includes also static queries, meaning queries performing mostly processing tasks and known in advance at architecture implementation time. Secondly, we have the *Dynamic-Query-Flow* which enters the model from the right and moves leftward. This flow is responsible for transferring queries and their corresponding responses between components. The Data-Flow primarily focuses on data transfer among components, while the Query-Flow is designed to interrogate data and generate additional knowledge.

Upon examining individual components, the model provides a comprehensive view of all the capabilities that can be offered. Both the phases and storage aspects possess distinctive features. In the case of phases, these features represent various abilities in performing data manipulations and computations. As for storage, these features denote different abilities in persisting data and the assurances that can be provided in

terms of non-functional requirements.

Data-intensive architectures often serve as infrastructures tasked with handling various query-flows, each potentially necessitating distinct data-flows tailored to specific query requirements. In this scenario, a single architecture may find itself simultaneously managing multiple data-flows, each corresponding to the various query-flows it handles. Since our model is designed to describe the data-flow for a single query-flow, the overall architecture is visualized as comprising multiple parallel instances of the model, each dedicated to a different data-flow/query-flow pairing. These instances may share producers and/or consumers, thereby optimizing the architecture's efficiency and resource utilization.

### 3. Systems

We define *Systems* as the technological components that can be combined to create a data-intensive architecture. Following the same line of the logical model, these systems can be categorized into two main groups: state-centric systems and data-centric systems. State-centric systems encompass technologies primarily designed for data storage and management. This category includes SQL databases, NoSQL databases, and lighter alternatives like Distributed File Systems. Data-centric systems, on the other hand, are technologies primarily focused on data processing. These systems can be further divided into batch-processing and stream-processing systems. For each category, we conducted a comprehensive review of technology documentation and literature to understand their capabilities in providing various functionalities. According to this, we illustrate how each type of system addresses different features within the model, highlighting common patterns within each category and any necessary specializations. This research is an integral part of our methodology for identifying the required system categories. Additionally, we provide examples of how well-known systems within each category operate to fulfill different requirements. These examples serve as guidance in the methodology for selecting a specific system after determining the appropriate category, although other systems offering similar capabilities may also be considered.

### 4. Methodology

In the second part of the thesis a software methodology to create and implement a data-intensive architecture is provided. The methodology assumes that the architect

- knows how many producers and consumers are present, what they produce and the queries they ingest in the system;
- has already carried out the requirement collection process, from which he/she has extracted a list of requirements to be covered when designing the architecture.

This information is used by the architect to create the methodology input in a format

The methodology, whose high-level pseudo code is shown in Figure 2, guides the adaptation of the overall model based on this input for the infrastructure, activating the necessary components and capabilities. At the end of this step, there is a specific instance of the model adapted for each data flow. These different instances are then overlaid, creating a set of components and capabilities for the specific application architecture. In other words, starting from the overall model, the methodology guides its customization based on the specific architecture, activating essential elements. It is important to note that during the methodology, not all the capabilities that the architecture will exhibit in the end will be activated, but only those that are fundamental to meet the specific requirements identified by the architect in the input and that must be present in the systems comprising the architecture. At the end of the first step of the methodology, you obtain the model for the specific application architecture, with the most relevant capabilities to be covered.

The second step involves selecting the systems to orchestrate for implementing the architecture. To do this, follow the guidelines explained in Section 3. The primary emphasis is on highlighting specific abilities, although it's worth noting that the overall architecture will likely encompass additional capabilities that are typical of data-intensive architectures in general.

```

main (infrastructure) {
  // identify data flows
  dataflows = find_subgraphs(infrastructure);

  // parametrize data flows
  dataflows = parametrize(dataflows);

  // STEP1: From Data-Flows to Model Mapping Instances
  model_instances = methodology1(dataflows);

  // STEP2: From Model Mapping Instances to Architecture Mapping
  architecture_model = methodology2(model_instances);

  // STEP 3 : From Architecture Mapping to Architecture Systems
  architecture_systems = methodology3(architecture_model);

  return architecture_systems;
}

```

Figure 2: Overview of the Methodology.

## 5. Conclusions

In summary, this research has dedicated its first part to the analysis of the theoretical aspects of big data system architectures, with the aim of creating a generalizable model capable of adapting to a wide range of real-world scenarios. Subsequently, we examined how these theoretical concepts find practical application in various systems, attempting to categorize them based on their abilities in both data processing and storage. The second part of this work focused on implementing these concepts to develop a decision-making algorithm for technology selection based on specific requirements.

Regarding future work, there are exciting opportunities for improvement. Firstly, the technology selection process could be further automated, perhaps by developing an intelligent framework that interrogates the architect or end user about their needs and suggests a set of appropriate systems automatically. Furthermore, we could explore further developments in the field of artificial intelligence and machine learning to enhance the model's efficiency and recommendation accuracy. Another possible future step could involve expanding this research to consider factors such as data security and regulatory compliance, which are increasingly relevant in the realm of big data systems.

## References

- [1] Ali Davoudian and Mengchi Liu. Big data systems: A software engineering perspective. *ACM Computing Surveys (CSUR)*, 53(5):1–39, 2020.
- [2] Markus Maier, A Serebrenik, and ITP Van derfeesten. Towards a big data reference architecture. *University of Eindhoven*, pages 1–144, 2013.
- [3] Alessandro Margara, Gianpaolo Cugola, Nicoló Felicioni, and Stefano Cilloni. A model and survey of distributed data-intensive systems. *ACM Computing Surveys*, 56(1):1–69, 2023.
- [4] Pekka Pääkkönen and Daniel Pakkala. Reference architecture and classification of technologies, products and services for big data systems. *Big data research*, 2(4):166–186, 2015.