Towards a Process-Driven Design of Data Platforms

Matteo Francia[1,*], Matteo Golfarelli[1] and Manuele Pasini[1]

[1]DISI – University of Bologna, Cesena, Italy

Abstract

Data platforms are state-of-the-art solutions to implement data-driven applications and analytics, since they facilitate the ingestion, storage, management, and exploitation of big data. Data platforms are built on top of complex ecosystems of services answering different data needs and requirements; such ecosystems are offered by different providers (e.g., Amazon AWS and Apache). However, when it comes to engineering data platforms, no unifying strategy and methodology is there yet, and the design is mainly left to the expertise of practitioners in the field. In particular, service providers simply expose a long list of interoperable and alternative engines, making it hard to select the optimal subset without a deep knowledge of the ecosystem. A more effective approach to the design starts from the knowledge of the data transformation and exploitation processes that should be supported by the platform. In this paper, we sketch a computer-aided design methodology and then focus on the selection of the optimal services needed to implement such processes. We believe that our approach lightens the design of data platforms and enables an unbiased selection and comparison of solutions even through different service ecosystems.

Keywords

Data Platform, Methodology, Big Data, Cloud Computing

1. Introduction

Digital transformation is one of the most disruptive trends of recent years. Digitization is perceived as the most effective solution to innovate every industrial sector thanks to process automation and the (automatic) exploitation of the value hidden in data. Such evolution is pushing information systems towards complex ecosystems of data-oriented services answering different data needs and requirements.

A data platform is a centralized infrastructure that facilitates the ingestion, storage, management, and exploitation of large volumes of heterogeneous data. It provides a collection of independent and well-integrated services meeting the end-to-end needs of data pipelines, where: *centralized* means that a data platform is conceptually a single and unified component; *independent* means that changes in the implementation of a service do not affect other services; *well-integrated* means that services have interfaces that enable easy and frictionless composition; and *end-to-end* means that services cover the entire data life cycle. Data platforms foster collaboration and shared governance (being centralized, data is unified following some integration and it is easier to ensure compliance with data protection and privacy laws through shared security and access control), and scalability (being implemented on a distributed infrastructure, it is easy to add storage and computing resources as needed).

When it comes to building data platforms, no unifying strategy and methodology is there yet: building data platforms is mainly left to the expertise of practitioners in the field. On the one hand, cloud service providers[1] offer ecosystems of services composed of many engines interoperable with each other; however, choosing the optimal set of services is hard since multiple solutions could fulfill the desiderata (e.g., whether disjoint databases and data warehouses or a single Lakehouse [1] should be used) and could require vertical knowledge on the design of data pipelines. On the other hand, several abstract big data architectures have been introduced (e.g., NIST [2], Lambda [3], and Kappa [4]). However, while they provide the necessary functionalities to enable big-data applications, their implementation and adoption require to understand which services should be used.

Neither providers of service ecosystems nor abstract architectures answer a crucial question: *given an ecosystem of services and the data-driven processes to support, which is the optimal subset of services enabling such processes?* Answering this question is hard since (i) each provider offers many (overlapping) services, (ii) different providers offer different service categorizations that can hardly be mapped together, (iii) the evolution of cloud ecosystems is fast and stakeholders without vertical technical knowledge can hardly keep up with the pace, and

DOLAP 2024: 26th International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data, co-located with EDBT/ICDT 2024, March 25, 2024, Paestum, Italy

[*] Corresponding author.

m.francia@unibo.it (M. Francia); matteo.golfarelli@unibo.it (M. Golfarelli); manuele.pasini2@unibo.it (M. Pasini)
0000-0002-0805-1051 (M. Francia); 0000-0002-0437-0725 (M. Golfarelli); 0009-0005-7023-6742 (M. Pasini)

CEUR Workshop Proceedings (CEUR-WS.org)

[1] While data platforms are not mandatorily coupled with cloud computing, cloud computing is proving to be a winning business model since deploying and maintaining such a variety of computational resources and assets requires advanced technical skills that companies hardly have “in-house”.

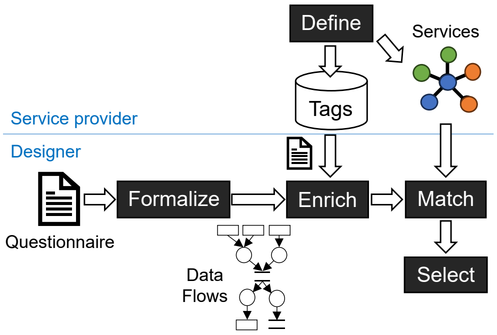

Figure 1: Overview of the methodology

(iv) simply selecting "all" engines is not viable due to cost and management reasons.

We believe that to ease the data platform design, the description of data-driven processes should drive such activity. Indeed, data pipelines are the backbone of a *data* platform, determine information-rich representations, and are congenial to designers since they encode many constraints on the choices to be made.

The paper is organized as follows: Section 2 sketches the overall methodology; Section 3 describes the case study used along the paper; Section 4 and Section 5 describe and formalize the design steps; and Section 6 analyzes the related literature. Finally, Section 7 draws the conclusions and future directions.

2. Methodology Overview

Engineering a data platform is nontrivial. A data platform must be designed upon a principle of modularity to be extensible with new components and functionalities. While it is true that reference architectures exist (Section 6), not all their modules and services might be necessary. Indeed, running useless functionalities and services results in a waste of resources and money (e.g., in cloud environments where users pay for the running time of the allocated resources).

To effectively build a data platform, we propose a methodology (Figure 1) that involves three types of users: *cloud service providers* (IT experts with in-depth knowledge of services available in the ecosystem); data platform *designers* (consultants or people with expertise in designing data flows but with no vertical knowledge on cloud/service ecosystems) and *clients* asking for the design of the data platform blueprint (e.g., partners involved in the same European project).

The methodology aims to assist designers in selecting the services necessary to implement the clients' data pipelines out of the "unstructured" lists of services provided by cloud providers.

Here, we sketch the five steps composing the methodology, and we mainly focus on steps (4) and (5) in the remainder of the paper due to space constraints.

(1) Define the service ecosystem.
A cloud service provider identifies *una tantum*: (i) the alternative candidate services to compose the blueprint of the data platform, and (ii) a taxonomy of tags that describe and characterize such services. The identified services are organized in a service graph describing the preferences and dependencies between them as well as the tags characterizing each service. Following the assumption that cloud service providers do not provide two identical engines, our guideline in designing the taxonomy of tags is to have enough expressiveness (tags) to distinguish all services (i.e., no two services have exactly the same tags).

(2) Formalize the requirements through a Data Flow Diagram.
When building the blueprint of a new data platform, clients compile questionnaires that collect information about their data-driven processes and the main steps, subjects, and goals of their analysis. Note that the data platform should support (possibly independent) processes from multiple clients.

Designers then refine the answers to the questionnaires and formalize the processes into a Data Flow Diagram (DFD); during the process, a few interviews with the clients might be necessary. We choose the DFD formalism since it represents flows of data through an information system at a high level of abstraction, emphasizing the movement and transformation of data while hiding details such as decision points and interactions: knowing which repositories and processes compose the data-driven processes is enough to return a blueprint. To build the DFD, designers decompose the data flows into agents, processes, and repositories. Starting from an aggregated overview, our guideline is to recursively split candidate processes and repositories until each of them is characterized by homogeneous tags (e.g., if a repository contains both unstructured images and relational tables, split it into two homogeneous repositories).

(3) Enrich the DFD with service tags.
In order to match the DFD with the services deployed in a cloud ecosystem, the two must share the same characterization. Each process and repository in the DFD is enriched with the tags from the taxonomies previously identified by the cloud service provider. To properly characterize each process/repository (agents are not subjects of enrichment, since they are out of the scope of the platform), clients answer an additional set of questions; we recall that clients are not required to have vertical knowledge about computer and data science or engineering. Such questions are defined by the cloud service providers and driven by the tag taxonomies. Note that since our final goal is selecting the most appropriate services, we are

not interested in identifying and tracking the data/processes but rather their types and their flows.

Example 1. *Given a repository from the DFD, we ask the question: "What are the main types of collected data?"*
□ *Sensor data* □ *Images* □ *Videos*
□ *Satellite observations* □ *Tables*
Answering "Satellite observations" tags the repository with the properties (Volume, Big), (Data Model, File), *and* (Data Nature, Raster) *since earth observations are data-heavy files (e.g., around 1 GB for 100* km^2*) and tags the process to download such data as* (Collection, Pull) *since files are downloaded from an FTP server [5].*

(4) Match the DFD and service graphs.
Once the DFD and service graphs are characterized by tags from the same taxonomies, it is possible to match them. A DFD process or repository matches (i.e., can be implemented by) a service only if the service has the same or more functionalities to fully implement it.

(5) Select the optimal services.
Out of all the services that are candidate implementations, it is necessary to select the minimal blueprint of the data platform that covers all the DFD entities (the fewer the services, the lower the cost and management efforts). Furthermore, dependencies and compatibilities between services have to be taken into account.

3. Case Study: Agritech

Within the Agritech spoke of the NRRP European project [6], we deploy a data platform supporting prescriptive analytic tasks from 8 clients (mainly research institutes) in the field of precision agriculture. We will use this as a working case study throughout the paper. Here follows a qualitative description of one of the clients' data flows that the platform must support.

Example 2 (Analytics solutions to manage crops for optimum quality and sustainability)**.** *The analysis entails a flow that is structured as follows. (i) Data comes from soil moisture sensor grids, weather stations, and SENTINEL-2 satellites; (ii) Sensor and weather data is uploaded every 15 minutes to the platform, while satellite data is periodically downloaded; (iii) Soil moisture data is interpolated using mathematical (e.g., bilinear interpolation) and machine learning (e.g., neural networks) techniques; (iv) The interpolated data is stored in a relational database with a spatial extension (PostGIS); (v) Vegetation indexes are computed out of the raw satellite observations and integrated with enriched sensor data; (vi) Reports are periodically generated out of enriched data; (vii) Given an optimal soil moisture matrix, the enriched data is used to decide how much to irrigate the soil.*

After gathering questionnaires from the clients, we (designers) iteratively refined the interview into a DFD.

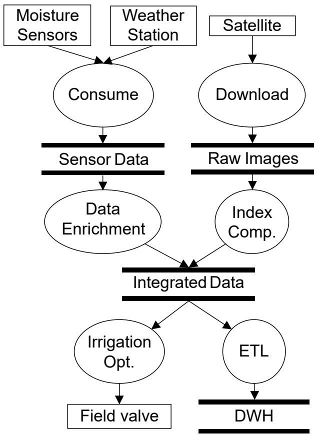

Figure 2: DFD from Example 2

Example 3 (DFD)**.** *Figure 2 depicts the tasks from Example 2 using the DFD formalism.*

- Moisture Sensors *and* Weather Stations *stream data into the platform, such data is* Consume*d and stored in the original format into* Sensor Data*;*
- Satellite *images are periodically* Download*ed and stored into* Raw Images*;*
- Sensor Data *are* Enrich*ed, interpolated, and stored into* Integrated Data*;*
- Raw Image*s are used for the computation of vegetation indexes and integrated with sensor data;*
- Integrated Data *are used to fuel a* Data Warehouse *through* ETL*;*
- *The* Irrigation Optimization *algorithm controls the* Field Valve*s installed in Emilia Romagna, Italy.*

Following the guidelines, the data collection processes cannot be merged into a single one since (i) Moisture Sensors *streams data into the platform while* Satellite *images are periodically downloaded, and (ii)* Sensor Data *and* Raw Images *cannot be grouped into a single repository since they contain heterogeneous data types. More details on such tasks are described in [7, 8].*

4. Mapping the Service Ecosystem

Tags (Table 1) are organized as a collection of hierarchies that can be fueled bottom-up from the documentation of cloud services providers (i.e., the set of tags that are attached to each service or that are inferrable through the service description), and top-down from the literature; for instance, the big data V's [9], such as volume (small

Table 1
Examples of taxonomies of tags

Data Model (All)	Structured	Relational
		Multidimensional
	Semi-structured	Document
		Wide-column
	Unstructured	Key-value
		Graph
		File (text, binary)
Data Nature (All)	Spatial	Vectorial
		Raster
	Temporal	
Volume (All)	Small	
	Big	
Functionality (All)	Landing (Raw)	
	Archive	
	Processed	
Goal (All)	OLAP	
	Operational	
	Machine learning	Classification
		Regression
Collection (All)	Pull	
	Push	
Computing (All)	Batch	
	Mini-batch	
	Streaming	

to big), variety (structured to unstructured), and velocity (low to high).

Definition 1 (Hierarchy)**.** *A* hierarchy h *is a taxonomy of categorical values* $Dom(h)$*.* $\geq_h$ *is the partial order that defines the taxonomy. We denote with* H *the set of hierarchies.*

Example 4 (Tags and hierarchies)**.** *Table 1 depicts an example of hierarchies* $H = \{\text{Data Model}, \text{Volume}, ...\}$*. In the partial order of the hierarchy* $h = \text{Data Model}$ *it is, for instance,* Relational $\geq_{\text{Data Model}}$ Structured $\geq_{\text{Data Model}}$ Data Model (All).

The services available on a specific ecosystem differ for each provider, and no single and shared organization of services is available. Table 2 shows an example of services from the AWS website (as of November 2023).

While a plethora of services is available, there is no necessity to consider all of them at once. Indeed, it is important that the considered services cover the "basic" functionalities necessary to run a data platform such as the ones proposed by NIST [2], among them, storage and processing engines.

Also, it is necessary to consider the dependencies between services (i.e., whether the adoption of a service requires another) and preferences in their choice (e.g., due to differences in performance reasons). This is why we map the services into a directed property graph.

Definition 2 (Directed property graph)**.** *A* directed property graph *is a tuple* $G = (N, A, P, L)$ *where* $N = \{..., n_i, ...\}$ *is a finite set of* nodes*,* $A = \{..., a_{ij}, ...\}$ *is a finite set of* arcs *connecting nodes* n_i *and* n_j*,* $P = \{..., (h, v), ...\}$ *is a set of key-value* properties *with* $h \in H$ *and* $v \in Dom(h)$*, and* L *is a set of* labels*.* $props : (N \cup A) \rightarrow P$ *returns the properties of a node or arc.* $label : (N \cup A) \rightarrow L$ *returns the labels of a node or arc.*

Definition 3 (Service graph)**.** *A* service graph *is a directed property graph* G^S *where nodes are labeled as* Service*, while arcs are alternatively labeled as* $\{\text{Requires}, \text{IsCompatible}\}$.

The semantics of the labels is the following:

- Service: is any engine from the service ecosystem;
- Requires: represents whether a service mandatorily relies on another;
- IsCompatible: represents whether a service natively interfaces with another (i.e., their interaction is supported by default and does not require custom/additional libraries or connectors).

Cloud service providers can optionally tag some services with the property (Preferred, True) to specify whether a service should be considered more than others.

Example 5 (Service graph)**.** *Examples from the service graph* G^S *built out of Table 2.*

$$N = \{n_1, n_2, n_3, n_4, ...\},\ A = \{a_{23}, a_{34}, ...\}$$
$$n_1 = \texttt{S3},\ n_2 = \texttt{GeoServer},\ n_3 = \texttt{EC2},\ n_4 = \texttt{EMR}$$
$$label(n_1) = \text{Service},\ label(a_{23}) = \text{Requires}$$
$$label(a_{41}) = \text{IsCompatible}$$
$$props(n_1) = \{(\text{Data Model}, \text{File}), (\text{Volume}, \text{All}), (\text{Preferred}, \text{True})\}$$
$$props(n_2) = \{(\text{Data Model}, \text{File}), (\text{Data Nature}, \text{Raster})\}$$

`GeoServer` *requires* `EC2` *since it is deployed on it,* `EMR` *is compatible with* `S3` *since it can natively read from and write to the object storage.*

5. Process-Driven Match and Selection

Given the DFD describing the clients' analysis[2], it is first necessary to match such graph with the service graph and then select the optimal subset of services. We recall that the DFD is enriched with tags identified by the cloud service provider (Table 1).

[2] In this paper, we mainly focus on the optimization of the platform design rather than on how, starting from questionnaires, designers refine the DFD of such flows.

Table 2
An excerpt of services from the Amazon AWS ecosystem (from https://aws.amazon.com/big-data/datalakes-and-analytics)

Solution areas	Use cases	AWS services
Advanced analytics	Interactive analytics	Athena
	Big data processing	EMR
	Data warehousing	Redshift
	Real-time analytics	Managed Service for Apache Flink
	Operational analytics	OpenSearch Service
	Dashboards and visualizations	QuickSight
	Visual data preparation	Glue DataBrew
Data management	Real-time data movement	Managed Streaming for Apache Kafka, Kinesis Data Streams, Kinesis Data Firehose, Glue
	Data governance	DataZone, Glue, Entity Resolution, Lake Formation, S3, Data Exchange, Clean Rooms
	Object storage for data lakes	S3, Lake Formation
	Backup and archive for data lakes	S3 Glacier, Backup
	Data catalog	Glue, Lake Formation
	Third-party data	Data Exchange, Clean Rooms
Machine learning	Frameworks and interfaces	Deep Learning AMIs
	Platform services	SageMaker

Definition 4 (Data Flow Diagram). *A* Data Flow Diagram *(DFD) is a directed property graph G^D where nodes are alternatively labeled as* {Agent, Repository, Process}, *while arcs are labeled as* Flow. *Arcs must connect with at least one node with label* Process.

The semantics of the labels is the following:

- Agent: an external entity that communicates with the system and stands outside of the system;
- Process: transforms inputs to outputs;
- Repository: stores data for later use;
- Flow: shows the transfer of data from one part of the system to another.

Example 6 (DFD). *Examples from the DFD G^D from Figure 2.*

$$N = \{n_1, n_2, n_3, ...\},\ A = \{a_{12}, ...\}$$
$$n_1 = \text{Consume},\ n_2 = \text{Sensor Data},\ n_3 = \text{Raw Images}$$
$$label(n_1) = \text{Process},\ label(n_2) = \text{Repository}$$
$$label(n_3) = \text{Repository},\ label(a_{12}) = \text{Flow}$$
$$props(n_2) = \{(\text{Data Model}, \text{Relational}), (\text{Data Nature}, \text{Vectorial})\}$$
$$props(n_3) = \{(\text{Data Model}, \text{File}), (\text{Data Nature}, \text{Raster})\}$$

5.1. Match the DFD and Service Graphs

Given the service graph and the DFD, we can automatically join them by matching their tags.

Definition 5 (Node Match). *Given a DFD and a service graph, a node n^D of the DFD matches a node n^S of the service graph if each property in $props(n^D)$ matches a property in $props(n^S)$. A property (h_i, v_i) matches another property (h_j, v_j) if $h_i = h_j$ and $v_i \geq_h v_j$.*

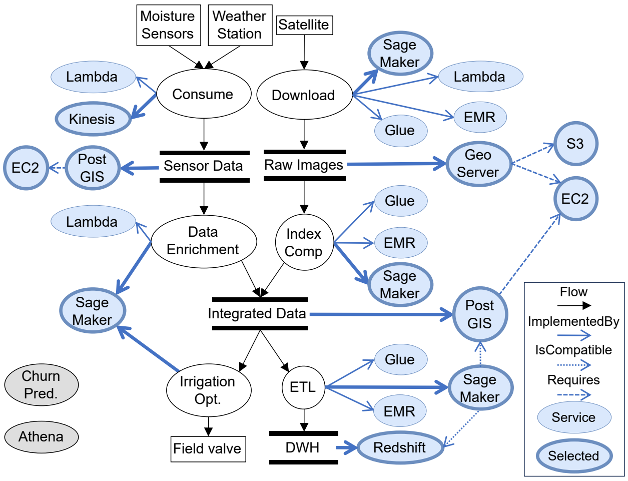

Figure 3: Matching the DFD (white) and service (blue) graphs; in bold the services that have been selected as the optimal blueprint for the data platform design

A match represents whether a DFD node can be implemented using a specific service (solid arcs in Figure 3); i.e., services that have the same or more generic values for the properties specified in the DFD node. For instance, a DFD repository tagged as Relational could be implemented by services supporting Relational or all Structured data. If no match is found for a DFD node, we force the match to a *default* (e.g., to a virtual machine where any functionality can be implemented) ensuring that all DFD nodes have at least one match.

Example 7 (Matched graph). *With reference to Figure 3, given the node* Raw Images *from the DFD, and the nodes* `S3` *and* `GeoServer` *from the service graph*

$$props(\text{Raw Images}) = \{(\text{Data Model}, \text{File}), (\text{Data Nature}, \text{Raster})\}$$
$$props(\texttt{GeoServer}) = \{(\text{Data Model}, \text{File}),$$

$$\text{(Data Nature, Raster)}\}$$
$$props(\texttt{S3}) = \{\text{(Data Model, File)}, \text{(Volume, All)}\}$$

Raw Images *can be implemented by* `GeoServer` *but not in* `S3` *since the former is natively capable of managing geographical raster images (while* `S3` *would only provide storage for the images).*

Definition 6 (Matched graph)**.** *Given a DFD* $G^D = (N^D, A^D, P^D, L^D)$ *and a service graph* $G^S = (N^S, A^S, P^S, L^S)$*, a matched graph* $G^M = (N^D \cup N^S, A^D \cup A^S \cup A, P^D \cup P^S, L^D \cup L^S \cup \{\text{ImplementedBy}\})$ *is a directed property graph obtained as the union of* G^S *and* G^D*.* A *is an additional set of arcs, one for each node match between* $n^D \in N^D$ *and* $n^S \in N^S$*. Arcs in* A *are labeled as* ImplementedBy*.*

The result of a match is a graph that is composed of the union of the nodes, and the union of the arcs plus additional arcs that represent candidate implementations for the DFD processes/repositories. The label ImplementedBy represents whether a DFD process or repository can be implemented by a specific service.

Sub-graphs that are not candidate implementations nor required by candidate implementations can be pruned a priori (gray in Figure 3).

Example 8 (Matched graph)**.** *Figure 3 depicts an excerpt of the matched graphs for the DFD from Figure 2; for the sake of clarity, not all the arcs have been presented. Nodes from the DFD have white background, while services are represented in blue. Solid arrows represent arcs labeled as* ImplementedBy *(e.g.,* Consume *can be implemented by either* `Lambda` *or* `Kinesis`*). Dotted arrows represent arcs labeled as* IsCompatible *(e.g.,* `SageMaker` *reads from and writes to* `Redshift`*). Dashed arrows represent arcs labeled as* Require *(e.g.,* `GeoServer` *requires* `EC2` *since it is deployed on it).* `Churn Prediction` *and* `Athena` *can be discarded a priori since they are not reachable from any DFD entity (i.e., they are neither candidate implementations nor required by other services).*

5.2. Select the blueprint

Out of the set of all possible matching services, only some of them must be selected such that (i) all the DFD repositories and processes are covered (agents are out of the scope of the platform) and (ii) the amount of services is minimized. The latter is an optimization goal that captures both the needs to minimize the economic cost of the platform and its management complexity. However, selection is not an easy task. For instance, some DFD entities can be implemented atop alternative services (e.g., a data lake that is implemented either on `S3` or `HDFS`), and some services are meaningful only if they cover at least two DFD entities (e.g., we use a `Lakehouse` to replace both a data lake and a data warehouse). Additionally, the optimization must be compliant with the following constraints.

1. *Coverage*: all processes and repositories in the DFD must be covered.
2. *Dependency*: if a service is selected, all its required services must be (recursively) selected too (e.g., `Geoserver` is deployed on `EC2`).
3. *Compatibility*: a service can be selected only if it is compatible with the services selected for the previous (if existing) and following (if existing) nodes in the DFD.
4. *Preference*: services marked as preferred should have more chances to be selected (for instance, preferences can be expressed for services entailing lower costs or higher performance).
5. *External pattern injection*: additional (architectural) constraints could be injected to force some service selection (e.g., use only services supporting Java language in compliance with legacy services); this can be implemented by pruning the unfeasible services from the matched graph.

Given a matched graph $G^M = (N, A, P, L)$, the selection of the optimal blueprint can be modeled as a linear programming problem inspired by the standard *facility location problem.* The formulation in Figure 4 reads as follows.

(1) The optimization function minimizes the weighted sum of the selected services. Weights w_i specify preferences for services s_i.

(2) s_i are binary variables modeling services. $s_i = 1$ if the service is selected, 0 otherwise.

(3) s_{ij} are binary variables modeling the services implementing DFD processes and repositories. $s_{ij} = 1$ if $n_i \in N$ s.t. $label(n_i) \in \{\text{Repository}, \text{Process}\}$ is implemented by $n_j \in N$ s.t. $label(n_j) = \text{Service}$.

(4) Binding variables s_j and s_{ij}: service s_j is selected if it implements ($s_{ij} = 1$) a DFD node $n_i \in N$ s.t. $label(n_i) \in \{\text{Repository}, \text{Process}\}$.

(5) *Coverage* constraints: every repository and process must be implemented by exactly one service.

(6) *Dependency* constraints: if a service is selected, all the services it depends on must be selected.

$$min \sum_i w_i s_i \tag{1}$$

$$s.t.\ s_i \in \{0,1\} \text{ for all } n_i \in N, label(n_i) = \text{Service} \tag{2}$$

$$s_{ij} \in \{0,1\} \text{ for all } a_{ij} \in A, label(a_{ij}) = \text{ImplementedBy} \tag{3}$$

$$s_j \geq s_{ij} \text{ for all } s_{ij} \tag{4}$$

$$\sum_{j, label(a_{ij})=\text{ImplementedBy}, a_{ij} \in A} s_{ij} = 1 \text{ for all } n_i \in N, label(n_i) \in \{\text{Process}, \text{Repository}\} \tag{5}$$

$$s_j \geq s_i \text{ for all } a_{ij} \in A, label(a_{ij}) = \text{Requires} \tag{6}$$

$$s_{ij} + s_{kh} \leq 1 \text{ for all } (a_{ik} \in A \text{ s.t. } label(a_{ik}) = \text{Flow}), (a_{jh} \notin A \text{ s.t. } label(a_{jh}) = \text{IsCompatible}) \tag{7}$$

Figure 4: Selecting the minimal set of services considering coverage (5), dependency (6), and compatibility (7) constraints

(7) *Compatibility* constraints: if two services are incompatible, they cannot be selected to implement two consecutive processes/repositories linked through a data flow in the DFM.

We implemented our approach (available at https://github.com/big-unibo/DataPlatformDesign) in Python and we leverage the CPlex library to effectively solve the optimization. As to preferences, we set

$$w_i = \begin{cases} 0.5 \text{ if (Preferred, True)} \in props(n_i) \\ 1.0 \text{ otherwise} \end{cases}$$

Example 9 (Selection). *Given the matched graph from Figure 3, the optimal blueprint of the data platform is represented by the services highlighted in bold (with a cost of 6.5), namely* `Kinesis`*,* `PostGIS`*,* `EC2`*,* `SageMaker`*,* `GeoServer`*,* `S3`*,* `Redshift`*.* `PostGIS` *has been selected since it is the only available implementation for* Sensor Data*, while* `EC2` *has been selected since it is required by* `PostGIS` *(similarly for* `GeoServer`*).* `SageMaker` *has been selected and preferred to* `Lambda`*,* `EMR`*, and* `Glue` *since four DFD processes can be implemented on it.*

6. Related Work

Understanding which "building blocks" should be part of a data platform is nontrivial.

Several functional architectures (system representations that focus on its functions and their interactions) have been proposed. NIST designed the Big Data Reference Architecture [2] which comprised vendor-, technology- and infrastructure-independent logical functional components that are necessary for managing and processing big data (data provider, data consumer, system orchestrator, big data application provider, and big data framework provider). Lambda [3] is an architecture designed to handle massive quantities of data by taking advantage of both batch and stream-processing methods. Kappa [4] overcomes some limitations (e.g., "redundant" batch and streaming implementations) of Lambda by using a pure streaming approach with a single code base. While these architectures define functional components necessary for (big) data platforms, they are abstract and do not address the problem of selecting the optimal services necessary for designing, implementing, and deploying working data platforms.

Cloud service providers such as Amazon [10], Google [11], and Microsoft [12] provide ecosystems of independent and interoperable services. While ecosystems enable easy deployment of data pipelines, choosing the *optimal* services is hard: (i) each provider offers different services, (ii) a shared categorization and organization of services is missing, and (iii) it is hard for the users to understand which services are needed based on their data-driven processes [13]. Some multi-objective optimization techniques have been proposed, where users express requirements and preferences (e.g., minimal QoS) about single services [14, 15, 16], cloud deployment models (public and private) [17], and service providers [18, 19]. Finally, domain languages and ontologies have been introduced (e.g., [20]) to enable some service composition (e.g., [21]).

With respect to these works, the novelties of our approach are (i) the introduction of a methodology to enable data platform design, (ii) a computer-aided approach to support designers in their choices, and (iii) process-driven optimization to select the optimal blueprint out of many data flows, even from multiple stakeholders.

7. Conclusion and Future Works

The design of data platforms is a nontrivial task. In this paper, we have introduced a methodology to aid designers in selecting the optimal services (out of a service ecosystem) supporting data-driven processes from multiple stakeholders (e.g., partners in a research project), and we have addressed such selection as a facility location optimization problem.

Although our implementation delivered a valuable blueprint, the result lends itself to improvement in multiple aspects. *Expressivenes*: matching and selection should consider more complex architectural patterns (Lakehouse to replace both data lakes and warehouses) as well as support additional constraints (e.g., consider only some service vendors or programming languages). *Resource provisioning*: additionally to selecting the services, a complete approach should also consider how many instances of a service are required (to do so, a cost model should be studied). *Metadata integration*: while catalog and metadata management services do not directly introduce functionalities for data transformation and exploitation, the design should also recommend services helping in the management of the platform itself [22]. *User evaluation*: the produced blueprints should be compared with the ones recommended by expert designers.

References

[1] M. Zaharia, A. Ghodsi, R. Xin, M. Armbrust, Lakehouse: A new generation of open platforms that unify data warehousing and advanced analytics, in: CIDR, www.cidrdb.org, 2021, p. 8. URL: http://cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf.

[2] W. Chang, N. Grady, N.-P. NIST, NIST big data interoperability framework: Volume 2, big data taxonomies [version 2], 2018. doi:https://doi.org/10.6028/NIST.SP.1500-2r1.

[3] M. Kiran, P. Murphy, I. Monga, J. Dugan, S. S. Baveja, Lambda architecture for cost-effective batch and speed big data processing, in: BigData, IEEE, Santa Clara, CA, USA, 2015, pp. 2785–2792. doi:10.1109/BIGDATA.2015.7364082.

[4] J. Kreps, Questioning the Lambda Architecture, volume 2, O'Reilly, 2014.

[5] Earth Observation Guide, https://business.esa.int/newcomers-earth-observation-guide, 2020. Accessed 2023-10-31.

[6] Agritech, https://agritechcenter.it/, 2023. Accessed 2023-10-31.

[7] M. Francia, J. Giovanelli, M. Golfarelli, Multi-sensor profiling for precision soil-moisture monitoring, Computers and Electronics in Agriculture 197 (2022) 106924. doi:10.1016/j.compag.2022.106924.

[8] E. Baldi, M. Quartieri, G. Larocca, M. Golfarelli, M. Francia, J. Giovanelli, E. Xylogiannis, M. Toselli, Smart irrigation system for precision water management: effect on yield and fruit quality of yellow fleshed kiwifruit in northern italy, in: ECPA, Wageningen Academic Publishers, 2023, pp. 59–66. doi:10.3920/978-90-8686-947-3_5.

[9] J. Anuradha, et al., A brief introduction on big data 5vs characteristics and hadoop technology, Procedia computer science 48 (2015) 319–324. doi:10.1016/j.procs.2015.04.188.

[10] Amazon Web Services, https://aws.amazon.com/, 2023. Accessed 2023-10-31.

[11] Google Cloud Platform, https://cloud.google.com/, 2023. Accessed 2023-10-31.

[12] Microsoft Azure, https://azure.microsoft.com/en-us, 2023. Accessed 2023-10-31.

[13] L. Sun, H. Dong, F. K. Hussain, O. K. Hussain, E. Chang, Cloud service selection: State-of-the-art and future research directions, J. Netw. Comput. Appl. 45 (2014) 134–150. doi:10.1016/J.JNCA.2014.07.019.

[14] E. Cavalcante, F. Lopes, T. V. Batista, N. Cacho, F. C. Delicato, P. F. Pires, Cloud integrator: Building value-added services on the cloud, in: NCCA, IEEE, Toulouse, France, 2011, pp. 135–142. doi:10.1109/NCCA.2011.29.

[15] M. H. Nejat, H. Motameni, H. Vahdat-Nejad, B. Barzegar, Efficient cloud service ranking based on uncertain user requirements, Clust. Comput. 25 (2022) 485–502. doi:10.1007/S10586-021-03418-W.

[16] L. Lu, Y. Yuan, A novel TOPSIS evaluation scheme for cloud service trustworthiness combining objective and subjective aspects, J. Syst. Softw. 143 (2018) 71–86. doi:10.1016/J.JSS.2018.05.004.

[17] R. Garg, Mcdm-based parametric selection of cloud deployment models for an academic organization, IEEE Trans. Cloud Comput. 10 (2022) 863–871. doi:10.1109/TCC.2020.2980534.

[18] M. B. Kar, R. Krishankumar, D. Pamucar, S. Kar, A decision framework with nonlinear preferences and unknown weight information for cloud vendor selection, Expert Syst. Appl. 213 (2023) 118982. doi:10.1016/J.ESWA.2022.118982.

[19] A. Tomar, R. R. Kumar, I. Gupta, Decision making for cloud service selection: a novel and hybrid MCDM approach, Clust. Comput. 26 (2023) 3869–3887. doi:10.1007/S10586-022-03793-Y.

[20] M. Noura, M. Gaedke, Wotdl: Web of things description language for automatic composition, in: Int. Conf. on Web Intelligence, ACM, Thessaloniki, Greece, 2019, pp. 413–417. doi:10.1145/3350546.3352558.

[21] H. B. Mahfoudh, A. Caselli, G. D. M. Serugendo, Learning-based coordination model for on-the-fly self-composing services using semantic matching, J. Sens. Actuator Networks 10 (2021) 5. doi:10.3390/JSAN10010005.

[22] M. Francia, E. Gallinucci, M. Golfarelli, A. G. Leoni, S. Rizzi, N. Santolini, Making data platforms smarter with MOSES, Future Generation Computer Systems 125 (2021) 299–313. doi:10.1016/j.future.2021.06.031.