Service Oriented Computing and Applications (2024) 18:1–12
https://doi.org/10.1007/s11761-024-00387-x

EDITORIAL

Towards automating microservices orchestration through data-driven evolutionary architectures

Giacomo Bergami[1]

Published online: 27 February 2024

Abstract
This paper briefly outlines current literature on evolutionary architectures and current links with microservices orchestration and data integration. We also propose future research directions bridging the field of service-oriented architectures with the data science domain.

Keywords Evolutionary architectures · Microservice orchestration · Architecture description language

1 Introduction

The rapidly evolving landscape of computer and data science entails the early adoption of new technologies in pre-existing ecosystems. Evolution is not necessarily gradual, as this might trigger a massive refactoring in pre-existing systems. The major forces driving such changes are the piling-up of additional business functionality required by the new customer [1], the improvement of pre-existing functionalities by delivering alternative solutions [2], or still the disruptive introduction of a new technology requiring a massive restructuring of the entire software ecosystem.[1] In this scenario, long-term planning is neither possible nor sustainable, as the current trends suggest that any future technology will drastically differ from the currently available ones. This then motivates the adoption of evolutionary architectures, which are usually characterised in terms of traditional software architectures (*topology*) as well as of (non-)functional requirements and the way to carry out computations through explicitly wired or "linkable" components (*governance*).

Concerning the principles of *evolutionary architecture topology* (Fig. 2), this proposes structuring highly decoupled architectures to minimise the dependencies across components. This was favoured by the recent emergence of platform as a service (PaaS) infrastructures such as Docker[2] providing simulated development. Regarding Fig. 1, each microservice would then allow to break apart traditional monolithic architectures, where each component might have different persistency requirements [1]. While a traditional relational or graph database might be better suited for storing an ENTERPRISE RESOURCE PLANNING (ERP) tracking all the crucial steps supporting Business Process Management (sales, purchasing, physical warehouse management, finance and accounting) [5], full-text documents might better support internal reports written by business analysts, while information related to social network data information gathered to carry out market research on purchasing customers might be better stored in a graph database. Then, the access to the internal database and data representation is mitigated through a Microservice Middleware layer, thus providing the querying interface to the database and mediating the access through a procedural language.

Within the same scenario, suppose that the company of interest handles a product purchase web service, which the users might use to buy products and post reviews of such products within the company's internal social network. In this scenario, we might outline three different kinds of services satisfying different information needs [6]: a company CEO might need access to all the information being generated by the organisation but might not necessarily need to have direct access to the data processing and machine learning algorithms, which can be then in turn used by the other microservices for integrating the data into a human-

✉ Giacomo Bergami
Giacomo.Bergami@newcastle.ac.uk

[1] School of Computing, Newcastle University, 1 Science Square, Newcastle upon Tyne NE4 5TG, England, UK

[1] https://yowcon.com/sydney-2022/sessions/2336/building-evolutionary-architectures-principles-and-practices.

[2] https://www.docker.com/.

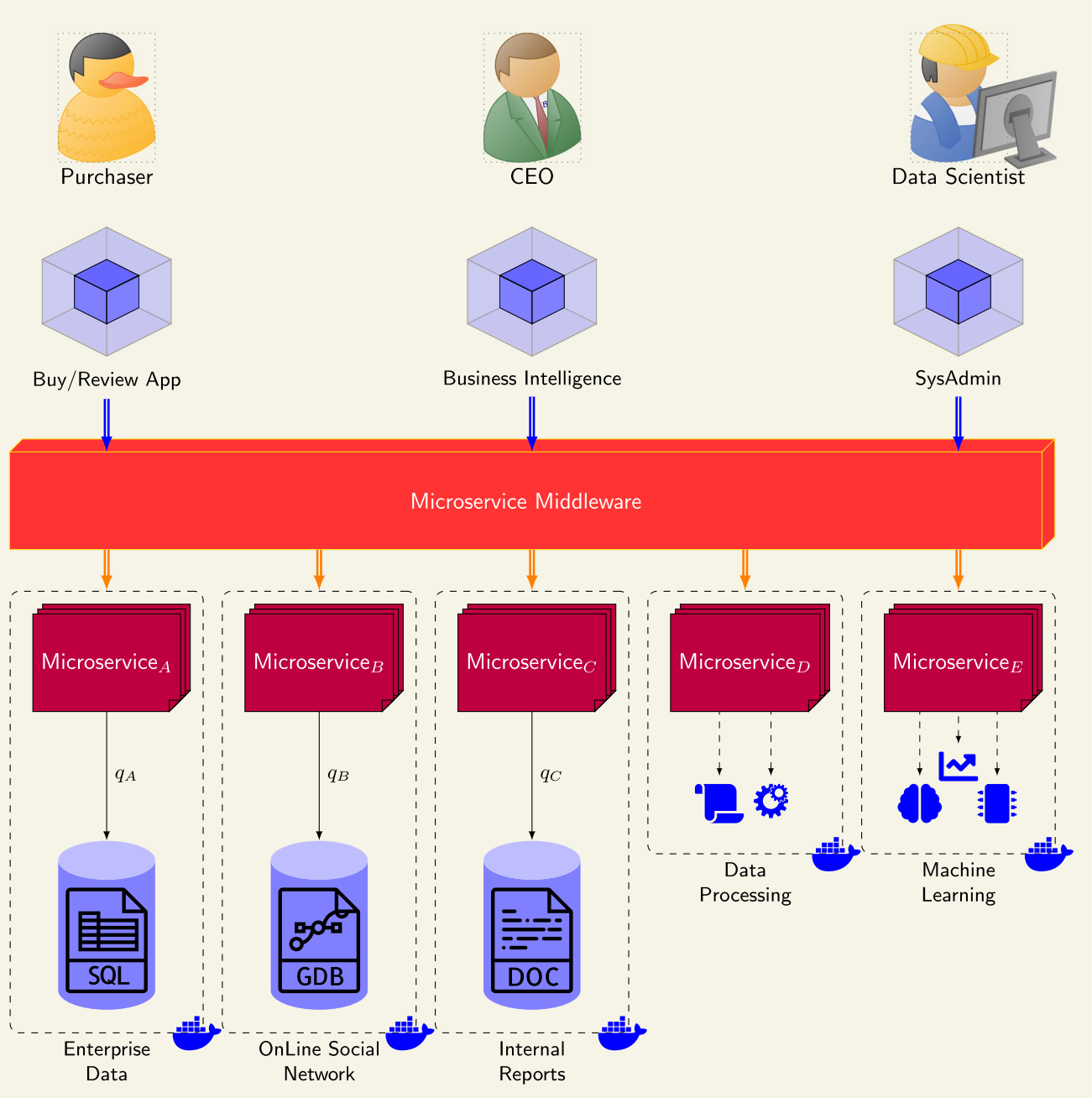

Fig. 1 An example of a data mesh architecture abiding by the evolutionary architecture topology requirements [1]. As users with different scopes and capabilities might access the same API layer (e.g. a CEO and customer using the platform for purchasing products), the API layer should selectively access the information to be processed and forwarded to the user. This information can be explicitly encoded at the schema level [3]

readable form; furthermore, such CEO might only read the data but not maintain it, thus having read-only access to the entire system. On the other hand, a purchaser using the company's platform might want to merely navigate the online catalogue without updating it and should not have access to further enterprise data; such user shall be able to update the enterprise social network via specific APIs while it shall not be allowed to update pieces of information concerning other users; they also shall have limited access to the machine learning and data processing capabilities of the company, as it might merely access to the product recommendation system without necessarily using any further AI infrastructure disrupting the company policies. Finally, a Data Scientist might have full control and access to the API and Microser-

Listing 1 Wright# [4], an example of an Architecture Description Language (ADL) using calculi for representing the behaviour of components and connectors. Arrows are used for prefixing.

```
connector CSConnector {
    role client(j) =
      request → req!j → res?j → process →
       client(j);
    role server() =
      req?j → invoke → process → res!j →
       server();
}

component SPClient {
    port test() = precheck → output → test();
}
component SPServer {
    port run() = invoke → execution → run();
}

system SampleCS {
    declare cslink = CSConnector;
    attach SPClient.test() = cslink.client();
    attach SPServer.run() = cslink.server();
    execute SPServer.run() | SPClient.test();
}
```

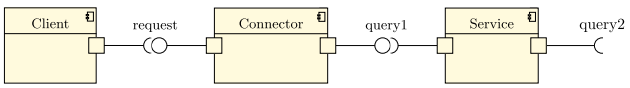

Fig. 2 Expressing a simple architecture topology regarding the UML component diagram. This represents a client directly forwarding a request to a (micro)Service through a Connector. Squares reflect ports, while circles identify roles [4]

vices layer, through which it might be able to deploy and terminate specific microservices; such a user should also be responsible for designing other client interfaces to the system for corporate clients, which will only need the access to the data processing and machine learning components of such a system. With this example in mind, we can clearly see that implementing a data science pipeline connecting clients to microservices boils down to solving a multi-database integration and a service orchestration problem.

Evolutionary architecture governance is concerned with verifying the goodness of the entire orchestrated architecture based on boundary conditions that might vary dynamically. In the context of microservices architecture, this reflects either the behaviour associated with each single microservice or the overall orchestrated components' behaviour to satisfy a client request (Listing 1). The discrepancy between the adherence of the architecture to such configuration and the expected outcome is then assessed through a multi-objective user-defined fitness function $F\colon \mathfrak{S} \to \mathbb{R}^{+}_{\geq 0}$ [1] taking as an input the configuration of the current software architecture $\mathfrak{S}$ and evaluating its fitness against an environment $\mathfrak{E}$ in terms of a numerical score to be maximised. This formulation enables the inclusion of load balancing requirements as crucial components for assessing the overall goodness of a specific dynamic configuration of the architecture [7, 8].

Despite the extensive literature on both sides of the computer science spectrum, a careful observation suggests that these, due to force majeure, often describe orthogonal and complementary approaches while a holistic solution encompassing all the features for both service orchestration and data integration is still missing.

The paper is structured as follows: after providing some historical remarks within the field of formal verification, planning, and process composition (Sect. 2), we collect some evidence on currently available features on the literature both on evolutionary architectures and microservices orchestrations (Sect. 3) and analyse some current challenges in the field (Sect. 4).

2 Historical remarks

2.1 Calculi for communication systems

Most of the current service composition approaches rely on the calculi for communication systems [9]. This is a Turing-complete formalism expressing all the possible computable actions performed by services within distributed or concurrent systems. Still, more up-to-date literature on the matter argues that being Turing-complete is insufficient for expressing all the desiderata from concurrency problems, such as enforcing the running of atomic operations required for synchronisation operations. Due to this, such authors proposed Multi-CCS [10], by extending the former calculus with the possibility of enforcing atomic operations for correctly modelling well-known problems (e.g. the dining philosophers). Multi-CCS can also express multi-part synchronisation, which is not inherently provided by the DGDL language for communicating agents.[3] Therefore, the latter protocol becomes relatively deficient in the required features to adequately represent concurrent communications. The resulting calculus is defined in Grammar 1, where 0 represents a process in deadlock or terminating, the *prefix* notation denotes the operations that a process performs first, the *strong prefix* defines atomic operations which, jointly with the parallel composition, enable multi-party synchronisation, as well as expressive alternative computations; finally, the restriction operator makes some operations private or enabling multi-process communication. By defining the behavioural semantics of the language, we can express the protocol expressed in such a calculus as a LABELLED TRANSITION SYSTEM (LTS) defined as a tuple $(\mathcal{A}, S, T)$, where S is a set of states, $\mathcal{A}$ is the set of all the possible actions

[3] https://www.arg.tech/index.php/research/dgdl/.

$$
\begin{array}{lcll}
\mathcal{S} & ::= & 0 & \textit{deadlock} \\
 & | & \mathcal{V}.\mathcal{S} & \textit{prefix} \\
 & | & \underline{\mathcal{C}}.\mathcal{S} & \textit{strong prefix} \\
 & | & \mathcal{S}+\mathcal{S}' & \textit{alternative} \\
\mathcal{P} & ::= & S & \textit{all processes} \\
 & | & \mathcal{P}|\mathcal{P} & \textit{parallel composition} \\
 & | & (\nu\mathcal{C})\mathcal{P} & \textit{synch. by restriction} \\
\mathcal{C} & ::= & \alpha, \beta, \ldots & \textit{channel names} \\
\mathcal{V} & ::= & \mathcal{C} \cup \overline{\mathcal{C}} & \textit{visible actions} \\
\mathcal{A} & ::= & V \cup \{\tau\} & \textit{all actions}
\end{array}
$$

Grammar 1 Multi-CCS [10]

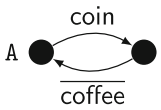

Fig. 3 LTS representation of the simple coffee machine process $\mathtt{A} = \mathsf{coin}.\overline{\mathsf{coffe}}.\mathtt{A}$ [10]

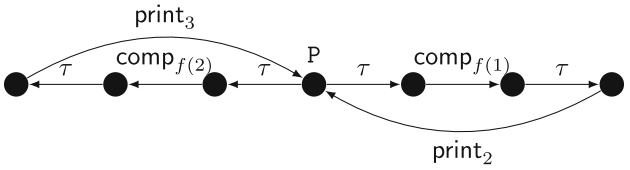

Fig. 4 LTS representing the protocol $P = (\nu\{\mathsf{req}, \mathsf{res}\})(\mathtt{C}|\mathtt{S})$ for $d \in \{0, 1\}$ and $f(x) = x + 1$. Please observe that synchronisation between actions and signed actions leads to silent transitions τ

associable to transitions among states, and $T \subseteq S \times \mathcal{A} \times S$ is the definition of the aforementioned transition. Such triplet can be then graphically rendered as an edge-labelled graph (Fig. 3).

For example, we can describe two processes, a client $\mathtt{C}$ and a server $\mathtt{S}$, where $\mathtt{C}$ sends a request d to $\mathtt{S}$, which in turn computes $f(d)$ and returns its outcome back to the client, which then prints the outcome. The two processes can each be uniquely identified by the current system of process equations:

$$
\begin{cases}
\mathtt{C} = \mathsf{req}(d).\overline{\mathsf{res}}\,\langle y\rangle\,.\mathsf{print}_y.\mathtt{C} \\
\mathtt{S} = \overline{\mathsf{req}}\langle d\rangle.\mathsf{comp}_{y \leftarrow f(d)}.\mathsf{res}(y).\mathtt{S}
\end{cases}
$$

while the synchronisation between the two processes over the request and the response action generates the protocol $(\nu\{\mathsf{req}, \mathsf{res}\})(\mathtt{C}|\mathtt{S})$. Figure 4 shows that the size of the resulting LTS will be directly proportional to the size of the domain D associated with the values parametrised through d.

Process calculi can also be further extended to express message passing of data across processes, thus fully capturing security and encryption protocols fully [11]. This can be seen as a simple extension of the former, where the syntax can be extended by replacing the prefix notation with the message x send and receive operations via a communication channel with a name in $\mathcal{C}$ as follows:

$$\mathcal{S} ::= \ldots \mathcal{C}(x).\mathcal{S} \mid \overline{\mathcal{C}}\,\langle x\rangle\,.\mathcal{S}\ \ldots$$

Furthermore, the authors further extend the alternative process running $\mathcal{S} + \mathcal{S}'$ with a conditional execution branching **if** *cond* **then** $\mathcal{S}$ **else** $\mathcal{S}'$, thus allowing the process to change behaviour depending on some internal or externally received data after associating it to a global variable.

We can easily observe that, when the domain associated with the possible variables' values is infinitely enumerable, this leads to an infinitely enumerable Multi-CCS transformation of such a process; this goes hand in hand with the transformation of any π-calculus algebraic process into LTS. Still, this explosive problem can be easily solved by extending such automata with registries holding the variable's values throughout the computation via the adoption of Fresh-Register Automata in lieu of LTS [12].

2.2 Planners

(Task) planners ideally describe a sequence of actions to be performed over an environment $\mathfrak{E}$ required to achieve a given goal state; such actions can be used to express the removal or substitution of specific operations [13, 14] as well as connecting and disabling components within a service-oriented architecture represented as a software artefact [2]. As a trustworthy technology leading to predictable behaviour of the to-be-automated component, planners are highly used in robotics [15] for performing actions under the meeting of specific preconditions while predicting the possible outcomes of such actions on the environment. Due to the possibility of using such tools for clearly structuring behaviour, such tools are also conveniently used for structuring Non-Player Characters in video games [16].

Given an initial world configuration I (that might also include numerical parameters referred to as *fluents*), the planning problem $\mathcal{P}$ aims at reaching a given goal configuration G by performing actions in Ω described in a planning domain $\mathcal{D}$. Such domain associates the aforementioned actions Ω with a set of propositions P describing any world configuration of interest. Each action $\omega \in \Omega$ is parametrised over some variables α_ω describing the world (e.g. *fluents*), and over which the precondition $\mathsf{Test}_\omega(\alpha_\omega)$ is tested as requirements to possibly perform ω on the environment; this will have as a possible effect $\mathsf{Eff}_\omega(\alpha_\omega)$ an update of the previously bounded parameters α_ω. Furthermore, the overall goal of the planner will be finding the goal solution minimising (or maximising) a specific fluent originally initialised in I, whose value is altered by the execution of actions $\omega \in \Omega$ [13]. Planning algorithms using heuristic functions for selecting the most suitable plan and not necessarily the first one being founded use such fitness functions for ranking the plans being the solution to the planning problem according to the problem domain formulation.

2.3 Data integration

Within the current field of database integration, we might consider two different kinds of approaches: **in-database integration** [3, 17], mainly related to data cleaning and duplicate removal within the same relational database, and **multi-database integration** [18–21], that properly pertains on finding a common data representation across different data sources. The latter can be postulated as the following problem [3]: "*given a global* MODEL G *of reflecting my expected final system and a query* q *written in a* LANGUAGE $\mathcal{L}_G$, *evaluate* q *over multiple data sources* $\mathcal{D} = \{ D_1, \ldots, D_n \}$ *being associated to different* MODEL $\{ M_1, \ldots, M_n \}$ *while providing the results abiding by* G". As the focus of evolutionary architectures does not rely on the design of each specific component rather than finding suitable ways for orchestrating such microservices, we are going to focus on the second type of database integration, as the first will rather occur within each single microservice of interest.

An **ontology** [22] is a semantic interpretation of DATA across MODELs establishing relations among the MODEL's types and defines generic transformation or mapping rules ς for converting objects representable within one ontology concept into another [3]. Ontologies act as semantic domains in denotational semantics [14, 23] and refer to concepts or entities that are independent of language and representation [24], thus appearing to be the ideal candidates for providing a full description of a microservice [14]. We are also interested in assessing the correspondences between different ontologies to reason on the data coming across different domains. In this context, an **ontological alignment** $A(O, O')$ of a source ontology O towards a destination ontology O' [3] is a set of tuples called **correspondences**. Each correspondence $(\delta t, E, t, \varsigma_{\delta t \to t}, s)$ maps a set of types or concepts δt from O into one type or concept t in O' using a transcoding function $\varsigma_{\delta t \to t}$, where E is the explanation for the alignment correctness in terms of Description Logic. Whenever $\varsigma_{\delta t \to t}$ is a bijection, the inverse function $\varsigma_{t \to \delta t} = \varsigma^{-1}_{\delta t \to t}$ is also provided. An uncertainty score s determines the accuracy of the correspondence [19, 25], thus providing a valid fitness score for the overall alignment effectiveness.

At this point, we should ask at which stage we prefer to execute the query q, either on the still-to-be composed microservices and then also after the data integration process (LOCAL AS A VIEW, [26–28]), or always at the end of such composition process (GLOBAL AS A VIEW, [29–32]). These two approaches are interchangeable within the data integration task. The first possible approach for integrating the outcome of distinct databases, known as Local-As-View (LAV) integration, involves performing sub-queries in their original distinct ontology and providing a uniform representation of the results. This is further used for running the rest of the query on an intermediate representation abiding by G [33, 34]. Global-As-View (GAV) integration is an alternative approach that aligns the original sources to a chosen target or hub ontology G before performing the overall query [35, 36]. In both LAV and GAV integration, finding G is crucial in the microservice scenario and shall be determined by dynamically aligning the client app ontology, sending the request across the microservices that might be used to answer the information request. The GAV approach allows each microservice to run in a safer environment without the need for internally running queries coming from a third-party app: on the other hand, the middleware layer will be in charge of reformulating the user query into microservice function calls, where resulting data is then reformulated in terms of the querying user model. We therefore prefer GAV over LAV for our microservice orchestration scenario, which can be then formulated as follows [3]:

Definition 1 (Global As a View) Given a set of DATA(bases) $\mathcal{D} = \{ D_A, \ldots, D_Z \}$ having their MODEL and METAMODEL expressed through ontologies $\mathcal{O} = \{O_A, \ldots, O_Z\}$, a user query p could be run on multiple databases data sources at the end of the data integration steps: after expressing each microservice data view in terms of the hub ontology, we can then aggregate all the entities and relationships using a algorithm clustering similar data together $\nu_{\cong}$ [37]. Such aggregated data is then queried with p. As a result, a GAV-driven query $\mathsf{GAV}_p(\mathcal{D}, \mathcal{O})$ can be defined as follows:

$$p\left(\nu_{\cong}\left(\mathcal{Q}^{A}_{O^{v}_{A},G}(D_A), \ldots, \mathcal{Q}^{A}_{O^{v}_{Z},G}(D_Z)\right)\right) \tag{1}$$

where A expresses the alignment between microservices ontologies and hub ontology G and $\mathcal{Q}^{A}_{O^{v}_{A},G}(D_A)$ represents the transformation of the data provided by each database D_A under its own exposed ontology O_A over the data representation required by the hub ontology G [3]. □

Please observe that, within a distributed environment, the apparent sequential formulation of Eq. 1 can be easily projected to different microservices and cloud nodes within an orchestrated distributed architecture [38, 39].

3 Evolutionary solutions

An evolutionary architecture supports guided and incremental change, describing the building of the software to be tested through fitness functions across multiple dimensions (*performance*, *security*, *code correctness*, *code quality*) [1]. This idea can be, therefore, extended to the orchestration of microservice components. This aims at supporting recent software platforms where single components collaborating to the overall computation within a data pipeline might evolve through time due to updated user requirements [2], while also

providing computational alternatives for the same intended result [40]. Still, a fully dynamic environment should be able to dynamically reconfigure services due to the changed assumptions at the boundaries. This then requires to compose services while guaranteeing that the overall composition satisfies the desired behaviour. We now describe current attempts to address the evolutionary challenge in data integration and service-oriented architecture domains.

3.1 Evolving ontologies

Evolutionary data science literature assumes that each microservice exposes its API with a basic RMI interface [1, 41], which can conveniently be annotated through an ontology (i.e. extended schema) representation [42]. Any schema update of their internal database [43] might also be reflected in an update of the exposed ontology-driven communication interface [1]. Furthermore, the possibility of directly injecting a third-party external component as additional microservices operating within the ecosystem entails that each microservice might describe concepts with different naming conventions and associated properties [44], thus requiring to dynamically align all the exposed and interacting ontologies to form a novel resulting hub ontology [45].

If we also assume that such microservice orchestration architecture will soon become the backbone of futuristic smart-city environments [14] where multiple agents roaming around the city forward real-time requests to microservices available in the cloud [46], we also need to support third-party vendor user applications completely agnostic of the resulting hub ontology within the microservice ecosystem due to privacy purposes. This requirement poses the problem of not having one single target ontology of reference, but rather having a final target hub ontology evolving through time while adapting to disparate user requests.

In this context, ontology evolution is at the heart of data evolution scenarios [47], as we need to characterise the evolution of a MODEL for representing the DATA of interest. To do this, we need a modelling tool that describes not only the concepts or types but also their relationships as well as provides a suitable decidable inference mechanism for better-reconciling types across different representations of our data. This can happen as the data representation might change by extending or reducing the metadata information associated with it [43]. As automated ontology alignments might come with some uncertainty [47] referring to the data similarity measures [21, 44], we can decide upon the best final hub ontology to be used for microprocess composition by ranking those through the aforementioned ontology fitness function.

We can, therefore, extend the usual notion of alignments as follows [3]:

Definition 2 (Multi-Soruce Data Integration System) A **multi-source data integration system** [18, 19] is defined as a triplet $\langle G, \mathcal{I}, \mathcal{O} \rangle$, where G is the *hub* ontology, representing the target of all the ontological alignments $A(O, G) \in \mathcal{I}$ having $O \in \mathcal{O}$ as a source (or local) ontology. When G needs to be found from $\mathcal{O}$, G can be first determined by cross-aligning all the ontologies in $\mathcal{O}$ for then coalescing the alignment operations together in G via a μ operator [45].

3.2 Supporting run-time configuration evolution

By broadening the scale of the use case in Fig. 1 to the context of a smart-city scenario where client requests are expressed by IoT devices forwarding them to an osmotic architecture [46], we can consider the aforementioned middleware orchestration layer as pervasive within the cloud-to-things continuum. As the number of available microservices and potential users forwarding requests to them might dramatically increase, the need for load balancing the computations within the network becomes more pressing. In such contexts, elasticity requirements [8] audit for specific metrics (e.g. minimum/maximum CPU usage, memory usage, data freshness and granularity [48]) to guarantee the meeting of the Quality of Service constraints through user-defined strategies re-distributing processes within the osmotic infrastructure. As multiple metrics are considered within this load-balancing scenario, this can be easily expressed as a multi-objective fitness function [7] ranking the best load-balancing strategy.

Such recently envisioned scenarios then require a huge paradigm shift from Data Lakes [49, 50], where data is at the centre of the service interface, to Data Meshes [51, 52] where both data, data processing algorithms such as database refactoring and data transformation through alignments, as well as better supporting machine learning and business analytics tools, coexist within the same environment. This also postulates that the underlying data representation should be object-oriented[4], so to potentially represent both time series as well as more structured information [36].

3.3 Planning evolutionary architecture governance

Despite the aforementioned elastic data analytics solutions being effective for achieving load balancing flexibility, these, on the other hand, mainly assume that the data collection and integration operations occur within each requesting client. On the other hand, competing approaches assume to have a thin client, where the microservice API middleware takes on the task of performing the overall data analytics operations by composing microservices, providing either data transformation operations or data silos, to satisfy an information need.

[4] https://web.archive.org/web/20140306143204/http://odbms.org/Introduction/history.aspx.

To achieve this, it is also necessary to compose the different services appearing in the cloud and orchestrate their execution in a pipeline similar to already-existing distributed data processing platforms [38, 39].

This section describes two orthogonal approaches in microservice composition and reconfiguration: the first provides a first microservice orchestration and composition by considering both the services' interfaces through their exposed ontologies and the user's requirements (Sect. 3.3.1), the second focus assumes to have an already-existing service configuration to be re-shaped under the updated user's functional requirements (Sect. 3.3.2).

3.3.1 Establishing microservice composition

State-of-the-art literature expresses the service composition problem in terms of a planning problem via an intermediate ontology hub [14]. To do so, authors assume that all services abide by the same Upper Ontology as an Architecture Description Language, thus describing a process and its computations in terms of input–output components. This notion can be further refined by assuming that each input/output data concept describing part of a service interface can subsume others [42] thus assuming a preliminary alignment step across concepts. Given this, the planning problem of choice for establishing an architecture configuration is then formalised by considering the user's provided input concepts and functional requirements as an initial configuration; actions embody the execution of a single microservice remote invocation, where preconditions reflect the accepted input data types and data conditions and the effects describe the output produced by a specific microservice; the overall planning goal reflects the user's expected execution output. As a consequence, the resulting plan reflects the intended orchestration of the overall components as well as the associated *evolutionary architecture governance*, thus identifying which microservices need to be activated and which are the links to be established across components. When the resulting orchestration can be described as a Direct Acyclic Graph due to the missing Iterate operators, the microservice orchestration can be subsequently structured in a layered execution [42] where each layer will contain all the atomic microservices which can be executed contemporarily as they do not have execution interdependencies [53].

In this domain, the algorithm used for searching the most appropriate plan describing the achieved services and their connections can be structured as either an A$\star$ multisource and multi-layer search, where each node is a service and each edge is a possible connection across components [42], or as a genetic algorithm where each chromosome reflects a candidate plan [14], and each gene reflects an activated scheduled microservice. While for the former the overall fitness function F is inversely proportional to the cost of traversed paths considering the number of overall active services and number of steps for carrying out the computation, in the latter F reflects both the feasibility of combining the services as well as the suitability over the user's requirements. This characterisation allows the planner to choose the specific evolutionary architecture governance solution that maximises the user requirements.

3.3.2 Dynamic architectural reconfiguration

Evolutionary architectures require changing their architecture configuration to adapt to new user requirements. This requires the machine the ability to problem-solving and, across all of the possible solutions, to find the one abiding by novel computational requirements, thus potentially requiring restructuring the link across the components. This avoids the need to pre-determine the target service composition while still requiring the full knowledge of the current target architecture configuration. Differently from the aforementioned approaches where microservices are mainly assumed to be simple input/output functions, the Architecture Description Language of choice (Wright#, Listing 1) models both microservices and connectors across those via a specialised process calculus with message passing. This dramatically differs from the aforementioned approach, where most microservices are assumed to be "*atomic and indecomposable*", thus "*directly satisfying a single intention*". Still, this calculus does not formalises the interfaces of these components and links in terms of ontology concepts, thus not being possibly used for data-driven process composition while still providing an orthogonal approach to the former.

After describing each microservice as a specific software engineering component within an *evolutionary topology* while connectors bridge and link different (active) components, authors in [54] showed for the first time the possibility of determining the operations required to transform one network configuration to another through a planning problem where the actions for (re/dis)connecting and (dis)activating components within the architecture; the initial configuration describes the state of activation of each microservice and the linking between communication channels, allowing the information to flow for a specific client request; finally, the planning goal is derived from the functional user requirements over the overall architecture functional requirements in LTL_f.

To alleviate the overbearing programmer task of manually reconfiguring the whole infrastructure due to the changes of the fitness function F [55], authors in [2] envision a new approach where F determines the total number of functional [2] or security [56] requirements being met expressed in LTL_f, where the planning algorithm is set-up to prefer the minimum amount of architecture restructuring operations. F is then computed as follows: after providing a multi-trace

representation of the resulting protocol as per the previous subsection (Sect. 2.1), an LTL_f satisfiability solver evaluates the resulting process resulting over the new functional requirements to be met, while potentially enforce a further restructuring of the architecture until all the desired new requirements are met.

4 Challenges and future works

4.1 Data science

4.1.1 Evolutionary architectures for machine learning pipelines

Concerning the data science field, given the resemblance of such architectures' fitness functions with Machine Learning ones determining the satisfactory training/fitting of a model, evolutionary principles were merely applied in the context of neural network architecture search [57] and hyperparameter search, where services are referring to specific neural network layers, which services provide only one functionality (computing an output over an input data representation), and where data is just represented in terms of vector embeddings [58] – and not in complex object-oriented representation. Although later approaches also attempted to consider tabular CSV data that might undergo transformation operations, these merely consider data operations over one single data source, such as missing values imputation and one-hot-encoding transformations [59]. When considering multiple data sources, authors assume already reconciled data with a compatible schema (i.e., an ontological specification), where single relationships (i.e., data tables) might be easily joined into a universe relationship through natural joins [60]. This does not adequately reflect the generic architecture proposed by full data mesh and evolutionary database architectures [43], as it also needs to support schema evolution on the services' side [41] resulting in database schema restructuring operations. Still, the possibility of performing data integration operations in current machine learning pipelines is unchallenged, thus requiring the explicit injection of pre-existing data integration and schema alignment solutions for automating data reconciliation and integration processes [3]. As a matter of fact, data integration pipelines and transformation pipelines [61] often rely on massive manual intervention for specifying correct record linkage and specifying the actual dimension of interest for carrying out the final analysis.

Please consider that, to the best of our knowledge, current microservices platforms such as Jolie [62] or even distributed data pipelines such as PlinyCompute [38] or Apache Flink [63] are not natively supporting automated data integration procedures, as they assume the direct intervention of the programmer to conduct data transformations. Still, this section will show how this is a required feature for orchestrating different microservices together with the aim of reducing the task of user query rewriting for any future user information need. Similar considerations are also given in the current elastic architecture framework, where the major information being currently collected mainly involves time series [48] or numerical data (e.g. images [8]) rather than structured data as the one encompassed by the use case scenario from this paper. This then postulates that current distributed data pipelines or microservices languages are not yet ready to automate evolving data integration functionalities fully. Therefore, further work towards this direction is supported by reusing prior tools as a substantial building block for the envisioned microservice middleware.

4.1.2 Data integration as a microservice middleware

To exploit the aforementioned distributed processing facilities, we need, as previously mentioned, to exploit the projection of a sequential computation within a distributed environment so to support microservices better acting as external system to be called (e.g. interoperation between Apache Flink and Apache Kafka[5]), also supporting the composition and pipelining of intermediate computations. To do this, we might attempt at directly express the service composition task for satisfying a user information need as a GAV query rewriting in Eq. 1, while each microservice exposes both data and the access to model facilities as a restful API, also representable as a query over a distributed database D_A. Furthermore, to adapt to the high variability of the data, unsupervised approaches for ontology alignments are a desirable feature [44, 64]. Due to the similarity of this set-up to CORBA and pre-existing middleware, we can freely assume to adopt the MODEL OBJECT FACILITY for representing data, models, and meta-models [3], thus abiding to the object-oriented representation of data also currently assumed in modern evolutionary architecture literature [41].

In this exercise, we can also overcome the limitations in current microservice literature by freely assuming that each microservice can be considered as a library serving disparate functions (i.e., queries) to be called via RMI (e.g. Apache Kafka). Each method exposed by a microservice A through its public interface might be considered a query $q_i^A(d)$ parameterizable over some input argument values d and run over an internal database D_A. We can, therefore, represent A as a finite collection of parameterizable queries $A = \left\{ q_1^A(\star), \ldots, q_n^A(\star) \right\}$; as each of such queries over D_A provides a view over such database definable by finitary extending the all the possible inputs d described in D_A, we

[5] https://nightlies.apache.org/flink/flink-docs-release-1.4/dev/stream/operators/asyncio.html.

can then describe D_A in terms of the global database exposed through an exhaustive search through the input parameters, thus defining a function $\delta_A(d) = \cup_{q_i^A \in A} q_i^A(d)$ for each $d \in D_A$.

Given that δ_A might accept a limited amount of parameters and return only certain entities from the database, we might expose, for external documentation purposes, only the concepts/types being accepted as parameters of the microservices and being returned by it, thus obtaining a restricted ontology O_A^v describing the interface exposed by each microservice, thus providing preliminary mapping conventions for the local data so to abide by the microservice platform data schema G [24]. As we might consider data models generated by AI algorithms as also data features, future works will address the challenge of extending microservices' Upper Ontologies to support the description of training machine learning algorithms, thus allowing the usage of machine learning models throughout the microservice middleware. This will allow the effective implementation of data meshes pipelines.

Last, this approach will also permit encoding the connectors between microservices as transformation functions ς, which is currently not explicitly supported by service-oriented literature.

If considering a LAV approach instead, each client request p can be decomposed through sub-queries p' of p, where each p' reflects the exposed interface of a microservice.

4.2 Certified fitness functions

At the time of the writing, current evolutionary architecture literature prescribes the adoption of continuous benchmarks and fitness function testing for assessing the algorithms' adequacy over each novel hardware and network configuration, as well as assessing the change related to microservices update and reconfiguration.

On the other hand, by restricting the expressive power of the programming languages in use within the architecture by guaranteeing the coding of always-terminating procedures, we can extract from all programmes relevant properties through static code analysis, thus limiting the need for dynamically testing the architecture to the bare essentials. This will also help maximise automation concerning the overall infrastructure fitness while minimising human intervention in code development and deployment.

4.2.1 Certified computational complexity

Static code analysis can estimate the clock cycles each part of the code will take after being compiled over a specific architecture [65]. Still, a reasonable estimate of algorithmic performance can only be achieved through time complexity analysis, on which the algorithm's scalability can be assessed independently from the data load being used in the current set-up. In order to achieve this through code static analysis, it is therefore required to restrict the expressive power of the programming language so that it is always guaranteed to terminate, therefore requiring a language that is not Turing complete. These theoretical considerations are in line with the W3C specifications, prescribing programmers to use the least-powerful programming language for a given purpose (*Rule of Least Power* [66]), and with hard real-time computing requirements, where developers aim for subroutines that are guaranteed to both finish and doing so before a given deadline while providing insight of a worst-case scenario analysis [67].

The C compilers designed for the Certified Complexity (CerCo) projects [67] showed that it is not only possible to estimate the clock cycles for running the code on a given architecture but also determine its overall time and space complexity when a programme does not contain unbounded iterations. In fact, if those were allowed, we might end up with a programme using the fully Turing-complete set-up of the language, over which it is impossible to have a decidable halting problem (as we also need to check the terminating condition within the code if any through static analysis otherwise). This further corroborates that any language for supporting evolutionary architectures should definitively be a total language and, therefore, not Turing complete.

Future works should also attempt to deliver novel methodologies for certified distributed computational complexity assessment by estimating the order of magnitude of the overall number of messages being exchanged within a distributed environment controlled by the microservice middleware.

4.2.2 Decidable Calculi for osmotic computations

Notwithstanding the expressiveness of the previously referenced calculi adequately capturing process synchronisation and data communication, most of these are Turing-complete: some property (such as process equivalence based on the indistinguishability of the programmes on their actions, e.g. *bisimulation*) is semidecidable. As decidable programming languages lead to an extensive expressive power for deciding on specific programme properties, we also seek fragments of such calculi adequately capturing communication of remote processes while being decidable.

One of the first contributions in this regard was the *polyadic* π*-calculus* [68], which achieved such a goal by preventing the parallel composition of processes within the recursive definition of processes. Concerning Fig. 1, each microservice might only run one process (or thread) at a time, thus completely excluding communications between sub-processes within each "recursive" microservice. On the other hand, walking on the footsteps of software with hard

real-time computing [65], we can easily achieve decidable computations by bounding the number of active *ambiences*, i.e. named collections of running processes that can be also nested [69]. This is a fairer assumption, as we can always both estimate the number of processes and resources required to carry out the computation in realistic cloud and edge scenarios, thus including the ones considering real urban settings [46]. Future works should address these challenges by designing a novel ADL language supporting both cloud and edge computing environments while also including the specificities of elastic smart-city scenarios within such environments.

Future research directions should also aim to extend such process calculi with ontological concepts referring to data input and output (e.g. communication variables): this might be achievable by decorating existing ADL languages with notions from Upper Ontologies. This should be the prelude to the definition of a holistic architecture description language, fully encompassing the description of data properties with process behaviour specifications through explicit data-driven technologies. At the time of the writing, Apache Kafka does not support ontological alignment within their service discovery process. As a minor note, such novel ADLs should also support describing data transformations ς through services via connectors as required by originated client requirements for data integration purposes.

4.2.3 Certified code and orchestration correctness

The constant addition of new technologies and services makes the code-correctness guarantee a necessity, as ensuring such data-processing correctness will also guarantee the correctness of the provided results. As current studies remark that most OpenSource platforms are severely affected by such bugs, this need becomes even more pressing in current pipeline architectures [70]. So, in the best of the possible worlds, we would like to detect bugs before the code goes into production. This section discusses which are the desired features of a programming language supporting the automation of fitness functions so as to provide correctness guarantees. We show that this boils down to providing current data science practitioners with adequate support for theorem proving through total and non-Turing complete languages (or, alternatively, with termination checking support) providing full support for *dependant types*.

The properties sought for such languages are ones that cannot be just determined through model checking but that strictly rely on the structural properties of the programming language and their behaviour. In practice, let us assume that we must test the algorithm's ability to effectively return a sorted list after providing an arbitrary list as an input. Current *theorem provers* allow to test these properties over safe iterative fragments of programming languages [71] and to implement recursive algorithms guaranteeing to terminate by only allowing recursive calls over "subsets" of the data received as an input [72]. These theorem provers can also verify the abidance of the specific programme to a given communication protocol after encoding the latter within the programming language's type system [73]. It would be then interesting to adopt such technologies to guarantee the overall correctness of the planned orchestration, thus in terms of the communication protocol and over the type transformation required for composing third-party microservices exposing differently shaped types.

Future work should also consider enabling the interoperation between legacy microservices and novel ones [1], as well as guaranteeing that microservices abide by formal specifications as addressed by the client's requests [2, 14].

References

1. Ford N, Parsons R, Kua P, Sadalage P (2022) Building evolutionary architectures: support constant change, 2nd edn. O'Reilly Media Inc
2. Chondamrongkul N, Sun J (2023) Software evolutionary architecture: automated planning for functional changes. Sci Comput Prog 230:102978. https://doi.org/10.1016/j.scico.2023.102978
3. Bergami G (2018) A new nested graph model for data integration. Ph.D. thesis, University of Bologna, Italy, pp 119-155 (2018). https://doi.org/10.6092/unibo/amsdottorato/8348
4. Chondamrongkul N, Sun J, Warren I (2019) PAT approach to Architecture Behavioural Verification. In: The 31st International Conference on Software Engineering and Knowledge Engineering, SEKE 2019, Hotel Tivoli, Lisbon, Portugal, July 10-12, 2019, ed. by A. Perkusich (KSI Research Inc. and Knowledge Systems Institute Graduate School, 2019), pp 187–252. https://doi.org/10.18293/SEKE2019-041
5. Petermann A, Junghanns M, Müller R, Rahm E (2015) FoodBroker: generating synthetic datasets for graph-based business analytics. Springer, Cham, pp 145–155. https://doi.org/10.1007/978-3-319-20233-4_13
6. Bergami G (2019) A framework supporting imprecise queries and data
7. Geeta K, Prasad VK (2023) Self-improved algorithm for cloud load balancing under SLA constraints. Serv Oriented Comput Appl 17(4):277–291. https://doi.org/10.1007/S11761-023-00366-8

8. Murturi I, Dustdar S (2022) Decent: a decentralized configurator for controlling elasticity in dynamic edge networks. ACM Trans Internet Technol. https://doi.org/10.1145/3530692
9. Milner R (1989) Communication and concurrency. PHI Series in computer science. Prentice Hall
10. Gorrieri R, Versari C (2015) Introduction to Concurrency Theory - Transition Systems and CCS. Texts in Theoretical Computer Science. An EATCS Series. Springer. https://doi.org/10.1007/978-3-319-21491-7
11. Fournet C, Abadi M (2003) Hiding names: private authentication in the applied pi calculus. In: Okada M, Pierce BC, Scedrov A, Tokuda H, Yonezawa A (eds) Software security: theories and systems. Springer, Heidelberg, pp 317–338
12. Tzevelekos N (2011) Fresh-register automata. SIGPLAN Not 46(1):295–306. https://doi.org/10.1145/1925844.1926420
13. Bergami G, Maggi FM, Marrella A, Montali M (2021) Aligning data-aware declarative process models and event logs. In: Polyvyanyy A, Wynn MT, Van Looy A, Reichert M (eds) Business process management. Springer, Cham, pp 235–251
14. Daosabah A, Guermah H, Nassar M (2023) User's intention and context as pertinent factors for optimal web service composition. SOCA. https://doi.org/10.1007/s11761-023-00380-w
15. He K, Lahijanian M, Kavraki LE, Vardi MY (2015) Towards manipulation planning with temporal logic specifications. In: IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015 (IEEE, 2015), pp 346–352. https://doi.org/10.1109/ICRA.2015.7139022
16. De Pellegrin E, Petrick R (2022) Plan simulation with pdsim. CEUR Workshop Proceedings 3065
17. Leser U, Naumann F (2007) Informationsintegration. dpunkt.verlag
18. Groß A, Hartung M, Kirsten T, Rahm E (2011) Mapping Composition for Matching Large Life Science Ontologies. In: ICBO, CEUR Workshop Proceedings, vol. 833. CEUR-WS.org
19. Hartung M, Groß A, Rahm E (2013) Composition methods for link discovery. In: BTW, LNI, vol 214 (GI, 2013), pp 261–277
20. Euzenat J, Shvaiko P (2007) In Ontology Matching. Springer
21. Aligon J, Gallinucci E, Golfarelli M, Marcel P, Rizzi S (2015) A collaborative filtering approach for recommending olap sessions. Decis Supp Syst 69:20–30. https://doi.org/10.1016/j.dss.2014.11.003
22. Allemang D, Hendler J (2011) Semantic web for the working ontologist: effective modeling in RDFS and OWL, 2nd edn. Morgan Kaufmann Publishers Inc., San Francisco
23. Saeki M, Kaiya H (2006) On Relationships Among Models, Meta Models, and Ontologies. In: Proceedings of the 6th OOPSLA workshop on domain-specific modeling
24. Henderson-Sellers B (2012) On the Mathematics of Modelling, Metamodelling, Ontologies and Modelling Languages. Springer Briefs in Computer Science. Springer, pp I–IX, 1–106
25. Euzenat J, Shvaiko P (2013) Ontology matching, 2nd edn. Springer, Heidelberg
26. Manolescu I, Florescu D, Kossmann D (2001) Answering XML queries on heterogeneous data sources. In: VLDB. Morgan Kaufmann, pp 241–250
27. Nadal S, Romero O, Abelló A, Vassiliadis P, Vansummeren S (2017) An integration-oriented ontology to govern evolution in big data ecosystems. In: EDBT/ICDT Workshops, CEUR workshop proceedings, vol 1810. CEUR-WS.org
28. Sint R, Stroka S, Schaffert S, Ferstl R (2009) Combining Unstructured, Fully Structured and Semi-Structured Information in Semantic Wikis. In: 4th Semantic Wiki Workshop (SemWiki 2009) at the 6th European Semantic Web Conference (ESWC 2009), Hersonissos, Greece, June 1st, 2009. Proceedings. http://ceur-ws.org/Vol-464/paper-14.pdf
29. Magnani M, Montesi D (2004) A unified approach to structured, semistructured and unstructured data. Tech. rep., in education. Inf Process Manag 29
30. Magnani M, Montesi D (2006) A unified approach to structured and XML data modeling and manipulation. Data Knowl Eng 59(1):25–62. https://doi.org/10.1016/j.datak.2005.06.004
31. Lu JJ (2006) A data model for data integration. Electron Not Theor Comput Sci 150(2):3–19
32. Botoeva E, Calvanese D, Cogrel B, Rezk M, Xiao G (2016) OBDA beyond relational DBs: a study for mongodb. In: Description Logics, CEUR Workshop Proceedings, vol. 1577. CEUR-WS.org
33. Holubová I, Contos P, Svoboda M (2021) Multi-model data modeling and representation: state of the art and research challenges. In: IDEAS 2021: 25th international database engineering & applications symposium, Montreal, QC, Canada, July 14-16, 2021 (ACM, 2021), pp 242–251. https://doi.org/10.1145/3472163.3472267
34. Holubová I, Contos P, Svoboda M (2021) Categorical management of multi-model data. In: IDEAS 2021: 25th international database engineering & applications symposium, Montreal, QC, Canada, July 14-16, 2021 (ACM, 2021), pp 134–140. https://doi.org/10.1145/3472163.3472166
35. Halevy AY (2001) Answering queries using views: a survey. VLDB J 10(4):270–294. https://doi.org/10.1007/s007780100054
36. Bergami G, Zegadło W (2023) Towards a generalised semistructured data model and query language. SIGWEB Newsl. https://doi.org/10.1145/3609429.3609433
37. Saeedi A, Peukert E, Rahm E (2017) Comparative evaluation of distributed clustering schemes for multi-source entity resolution. ADBIS
38. Zou J, Barnett RM, Lorido-Botran T, Luo S, Monroy C, Sikdar S, Teymourian K, Yuan B, Jermaine C (2018), PlinyCompute: A Platform for High-Performance, Distributed, Data-Intensive Tool Development. In: Proceedings of the 2018 International Conference on Management of Data (Association for Computing Machinery, New York, NY, USA, 2018), SIGMOD '18, pp 1189–1204. https://doi.org/10.1145/3183713.3196933
39. Junghanns M, Petermann A, Teichmann N, Gomez K, Rahm E (2016) Analyzing extended property graphs with apache flink. SIGMOD workshop on Network Data Analytics (NDA)
40. Zhang T, Subburathinam A, Shi G, Huang L, Lu D, Pan X, Li M, Zhang B, Wang Q, Whitehead S, Ji H, Zareian A, Akbari H, Chen B, Zhong R, Shao S, Allaway E, Chang S, McKeown KR, Li D, Huang X, Sun K, Peng X, Gabbard R, Freedman M, Kejriwal M, Nevatia R, Szekely PA, Kumar TKS, Sadeghian A, Bergami G, Dutta S, Rodríguez ME, Wang DZ (2018) GAIA: a multi-media multi-lingual knowledge extraction and hypothesis generation system. In: Proceedings of the 2018 Text Analysis Conference, TAC 2018, Gaithersburg, Maryland, USA, November 13-14, 2018 (NIST, 2018). https://tac.nist.gov/publications/2018/participant.papers/TAC2018.GAIA.proceedings.pdf
41. Kleppmann M (2016) Designing data-intensive applications: the big ideas behind reliable, scalable, and maintainable systems. O'Reilly. http://shop.oreilly.com/product/0636920032175.do
42. Gamha Y (2023) A framework for rest services discovery and composition. Serv Oriented Comput Appl 17(4):259–275. https://doi.org/10.1007/s11761-023-00376-6
43. Ambler SW, Sadalage PJ (2006) Refactoring databases: evolutionary database design. Addison-Wesley Professional
44. Melnik S, Garcia-Molina H, Rahm E (2002) Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching. In: Proceedings of the 18th international conference on data engineering, San Jose, CA, USA, February 26 - March 1, 2002, ed. by R. Agrawal, K.R. Dittrich (IEEE Computer Society, 2002), pp 117–128. https://doi.org/10.1109/ICDE.2002.994702

45. Henderson-Sellers B (2012) On the mathematics of modelling, metamodelling, ontologies and modelling languages. Springer Briefs in Computer Science. Springer, pp I–IX, 1–106
46. Almutairi R, Bergami G, Morgan G, Gillgallon R (2023) Platform for energy efficiency monitoring electrical vehicle in real world traffic simulation. In: 25th ieee conference on business informatics, CBI 2023 - Volume 1, Prague, Czech Republic, June 21-23, 2023 (IEEE, 2023), pp 1–8. https://doi.org/10.1109/CBI58679.2023.10187450
47. Thor A, Hartung M, Groß A, Kirsten T, Rahm E (2009) An Evolutionbased Approach for Assessing Ontology Mappings - A Case Study in the Life Sciences, in Datenbanksysteme in Business, Technologie und Web (BTW 2009), 13. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS), Proceedings, 2.-6. März 2009, Münster, Germany, LNI, vol. P-144, ed. by J.C. Freytag, T. Ruf, W. Lehner, G. Vossen (GI, 2009), pp 277–286. https://dl.gi.de/handle/20.500.12116/20452
48. Laso S, Berrocal J, Fernandez P, García JM, García-Alonso J, Murillo JM, Ruiz-Cortés A, Dustdar S (2022) Elastic data analytics for the cloud-to-things continuum. IEEE Internet Comput 26(6):42–49. https://doi.org/10.1109/MIC.2021.3138153
49. Liu P, Loudcher S, Darmont J, Noûs C (2021) ArchaeoDAL: a data lake for archaeological data management and analytics. In: Proceedings of the 25th international database engineering & applications symposium (association for computing machinery, New York, NY, USA, 2021), IDEAS '21, pp 252–262. https://doi.org/10.1145/3472163.3472266
50. Giebler C, Gröger C, Hoos E, Schwarz H, Mitschang B (2019) Modeling Data Lakes with Data Vault: Practical Experiences, Assessment, and Lessons Learned. In: Conceptual Modeling - 38th International Conference, ER 2019, Salvador, Brazil, November 4-7, 2019, Proceedings, Lecture Notes in Computer Science, vol. 11788, ed. by A.H.F. Laender, B. Pernici, E. Lim, J.P.M. de Oliveira (Springer, 2019), pp 63–77. https://doi.org/10.1007/978-3-030-33223-5_7
51. Dehghani Z (2022) Data mesh: delivering data-driven value at scale. O'Reilly Media, Inc
52. Machado IA, Costa C, Santos MY (2022) Advancing Data Architectures with Data Mesh Implementations. In: Intelligent Information Systems - CAiSE Forum 2022, Leuven, Belgium, June 6-10, 2022, Proceedings, Lecture Notes in Business Information Processing, vol. 452, ed. by J.D. Weerdt, A. Polyvyanyy (Springer, 2022), pp 10–18. https://doi.org/10.1007/978-3-031-07481-3_2
53. Bergami G, Appleby S, Morgan G (2023) Quickening data-aware conformance checking through temporal algebras. Information. https://doi.org/10.3390/info14030173
54. Méhus J, Batista TV, Buisson J (2012) ACME vs PDDL: support for dynamic reconfiguration of software architectures. CoRR arXiv:1206.0122
55. Eilertsen AM (2020) Refactoring operations Grounded in manual code changes. In: ICSE '20: 42nd International Conference on Software Engineering, Companion Volume, Seoul, South Korea, 27 June - 19 July, 2020, ed. by G. Rothermel, D. Bae (ACM, 2020), pp 182–185. https://doi.org/10.1145/3377812.3381395
56. Fionda V, Greco G, Mastratisi MA (2021) Reasoning about smart contracts encoded in LTL, in AIxIA. Springer, Cham, pp 123–136
57. Elsken T, Metzen JH, Hutter F (2019) Neural architecture search: a survey. J Mach Learn Res 20:551–5521
58. Liang J, Meyerson E, Hodjat B, Fink D, Mutch K, Miikkulainen R (2019) Evolutionary Neural AutoML for Deep Learning. In: Proceedings of the Genetic and Evolutionary Computation Conference (Association for Computing Machinery, New York, NY, USA, 2019), GECCO '19, pp 401–409. https://doi.org/10.1145/3321707.3321721
59. Chen S, Tang N, Fan J, Yan X, Chai C, Li G, Du X (2023) Haipipe: combining human-generated and machine-generated pipelines for data preparation. Proc ACM Manag Data. https://doi.org/10.1145/3588945
60. Grafberger S, Groth P, Schelter S (2023) Automating and optimizing data-centric what-if analyses on native machine learning pipelines. Proc ACM Manag Data. https://doi.org/10.1145/3589273
61. Bertini F, Bergami G, Montesi D, Veronese G, Marchesini G, Pandolfi P (2018) Predicting frailty condition in elderly using multidimensional socioclinical databases. Proc IEEE 106(4):723–737. https://doi.org/10.1109/JPROC.2018.2791463
62. Safina L, Mazzara M, Montesi F, Rivera V (2016) Data-driven workflows for microservices: genericity in jolie. In: 2016 IEEE 30th international conference on advanced information networking and applications (AINA) (2016), pp 430–437. https://doi.org/10.1109/AINA.2016.95
63. Papp S (2016) The definitive guide to apache flink: next generation data processing, 1st edn. Apress, USA
64. Sagi T, Gal A (2012) Non-binary evaluation for schema matching. In: Atzeni P, Cheung D, Ram S (eds) Conceptual modeling. Springer, Heidelberg, pp 477–486
65. Simon DE (1999) An embedded software primer, 1st edn. Addison-Wesley Longman Publishing Co. Inc
66. Berners-Lee T, Mendelsohn. The rule of least power. https://www.w3.org/2001/tag/doc/leastPower
67. Amadio RM, Ayache N, Bobot F, Boender JP, Campbell B, Garnier I, Madet A, McKinna J, Mulligan DP, Piccolo M, Pollack R, Régis-Gianas Y (2014) C. Sacerdoti Coen, I. Stark, P. Tranquilli, Certified Complexity (CerCo). In: Dal Lago U, Peña R (eds) Foundational and practical aspects of resource analysis. Springer, Cham, pp 1–18
68. Dam M (1997) On the decidability of process equivalences for the π-calculus. Theoret Comput Sci 183(2):215–228. https://doi.org/10.1016/S0304-3975(96)00325-8
69. Charatonik W, Gordon AD, Talbot JM (2002) Finite-control mobile ambients. In: Le Métayer D (ed) Programming languages and systems. Springer, Heidelberg, pp 295–313
70. Tang X, Wu S, Zhang D, Li F, Chen G (2023) Detecting logic bugs of join optimizations in dbms. Proc ACM Manag Data. https://doi.org/10.1145/3588909
71. Filliâtre JC, Magaud N (1999) Certification of Sorting Algorithms in the Coq System. In: 12th International Conference of Theorem Proving in Higher Order Logics. TPHOLs), Emerging Trends
72. Asperti A, Ricciotti W, Sacerdoti Coen C, Tassi E (2009) A compact kernel for the calculus of inductive constructions. Sadhana 34(1):71–144. https://doi.org/10.1007/s12046-009-0003-3
73. Brady E (2017) Type-Driven Development with Idris, 1st edn. Manning Books