Abstract—The data mesh is a novel data management concept that emphasizes the importance of a domain before technology. The concept is still in the early stages of development and many efforts to implement and use it are expected to have negative consequences for organizations due to a lack of technological guidelines and best practices. To mitigate the risk of negative outcomes this paper proposes the use of the mask–mediator–wrapper architecture as a driver for a data mesh implementation. The mask–mediator–wrapper architecture provides a set of prefabricated configurable components that provide basic functionalities that a data mesh requires. This paper shows how the two concepts are compatible in terms of functionality, data modeling, evolvability, and aligned capabilities. A mask–mediator–wrapper-driven data mesh facilitates low-risk adoption trials, rapid prototyping, standardization, and a guarantee of evolvability. We demonstrate a mask–mediator–wrapper-driven data mesh by using our open-source Janus system to experimentally drive an exemplified data mesh.

Index Terms—Data mesh, software architecture, data management, mediator–wrapper.

I. INTRODUCTION

The landscape of data management is constantly changing. In just two recent decades, there has been a great procession of technologies, formats, tools, and systems that have shown promise in tackling data management. Data warehouses serving as analytical data stores were the common starting point. The situation became complicated with the introduction of NoSQL, as unstructured schemas, multitudes of data formats, and data federation became commonplace.

In an understandable effort to stay ahead of the competition, the industry more than adapted to these new ideas. The need for velocitous, voluminous, and various data was stamped into the “3 Vs” slogan. With the demand now up to 8 Vs [1], a decade of data streaming, data lakes, and data lakehouses is at a close, with many promising ideas ahead.

These are all exciting ideas to explore, and very few can be considered obsolete. The problem with hitting the mark in data management is that the target is constantly moving, making data management an ongoing struggle susceptible to paradigm shifts [2]. The most recent of these is the introduction of the data mesh [3], leaving leading organizations [4], [5], [6], [7] in a race to extract every ounce of benefit available to them from this new paradigm. The question is if all organizations can implement this shift quickly and stably enough with a favorable outcome, as some might be forced to exhaustively restructure their entire data organization and provision. These organizations run the risk of implementing a half-baked solution they don’t benefit from, ultimately quitting mid-shift with an unmanageable product. There is an implicit expectation that each organization will develop its own custom components to drive a data mesh, restricting inter-system composability, technologist migration among projects, and challenging evolvability. The impact of these problems is undeniable, with just 20 percent of analyses bringing value [8], and the failure rate of data science projects at 87 percent [9]. The pressing research question at hand is: Can a technological surrogate be employed to minimize the risk of introducing a data mesh into an organization?

This paper shows that a data mesh can be driven by the MMW (mask–mediator–wrapper) architecture system acting as a surrogate. The contribution of this paper is an MMW-driven data mesh concept that enables:

- **low-risk adoption trials** for organizations that might not have data exactly at scale;
- **rapid prototyping** to reduce the time required for implementation;
- **standardization** to reduce the cognitive load for technologists when switching projects and making the system composable;
- **a guarantee of evolvability** to alleviate technological changeability.

Sections II and III provide succinct overviews of the data mesh and MMW architecture concepts. Section IV discusses the compatibility of both concepts in terms of their functionalities, modeling abilities, architectural evolvability, and data mesh capability coverage. This is further demonstrated in Section IV-D1 by a case study where a hypothetical data mesh is constructed using the MMW architecture. Section V discusses the benefits that can be gained by using the MMW architecture to drive a data mesh. Section VI introduces experiments utilizing the prototypical MMW system Janus to drive an exemplified data mesh.
II. THE DATA MESH

The data mesh is an enterprise data platform architecture that converges the ideas of [10]:

- distributed domain driven architecture;
- self-serve platform design;
- product thinking with data.

It is an alternative to the centralized data platform approach, where data is centrally managed and served through coupled ETL (extract-transform-load) processes.

The key organizational feature of the data mesh is that it arranges an organization’s data into bounded contexts. This brings domain-driven design from operational systems into analytical systems. With this conceptual data federation in place, teams can manage data in their domains more easily, acquire domain-specific knowledge faster, and handle data and new tasks with greater expertise. This means that responsibilities are distributed among teams along the bounded contexts rather than along mechanical functions (e.g. ingestion, cleansing, aggregation, serving) [10]. Echoing the statement of Richards and Ford [11] that in engineering architectures everything is a trade-off, the division by domains requires teams to be cross-functional.

Each team creates a data pipeline to load domain data from the data infrastructure platform (DIP). Ideally, domains shouldn’t overlap [12], but this is inevitable. In this case, a domain that uses data from another domain doesn’t create its own separate data pipeline from the DIP, rather it correlates the data from an existing domain. Since data is considered a product, its quality is guaranteed by the other domain’s team - teams have data ownership. Verifying and maintaining data quality is simpler with a data mesh since each team manages its own domain.

The idea of the data mesh corresponds with the concept of the evolutionary architecture, as presented by Ford et al. [2]. Each domain can be considered an architectural quantum, which is according to the definition of Ford et al. [2] an independently deployable component with high functional cohesion, which includes all the structural elements required for the system to function properly. Since cases of architectural quanta depending on each other are inevitable, this implies that the interfaces between domains must be designed technologically agnostic. Such interfacing is made easier because the data mesh domains have to follow global interoperability principles proscribed by federated computational governance. Dehghani [3] even goes so far as specifying that data products (defined by domains) have input and output data ports as interface points. It can be noticed that the data mesh relies heavily on the ports-and-adapters pattern of the hexagonal architecture [13], which is widely accepted. This transitively gives some promise that the data mesh might be a solution that could also be widely accepted. Even Netflix, the continual proponents of the hexagonal architecture [14], have been quick to admit that they find the data mesh compelling [4], [5].

In this domain-oriented architecture, data is consumed by users (in the form of reporting, visualization, or dashboards), operational systems, and analytical systems (other data products). The inter-connection possibilities are conceptually illustrated in Fig. 1. It is important to note how some data products might depend on analytical data from other data products; following the example of Fig. 1 the purple data product requires analytical data from the green domain. On the other hand, data products never depend on operational data from another domain; the purple data product didn’t directly reach for the operational data from the green domain, rather it reaches for the green domain’s data as a data product.

The DIP is seen as the primary data source location for the data mesh, containing data warehouses, data lakes, or operational data stores as data platforms [16], [17]. On the other hand, [3] mentions that there are other top-level data source archetypes in the data mesh such as collaborating operational systems or other data products. This raises a practical question of whether some data should remain physically close to the data product. In Fig. 2 we present a distilled view on this by Dehghani [3], which proposes that a DPC (data product container) can keep some data locally. This local data is the domain data being served as the data product. The local data must be consumed from the input data ports and transformed into a valuable product.

Summarily, the data mesh as a data platform enables [10]:

- Scalable polyglot big data storage;
- Encryption for data at rest and in motion;
- Data product versioning;
- Data product schema;
Fig. 3. Mask–mediator–wrapper architecture example.

- Data product de-identification;
- Unified data access control and logging;
- Data pipeline implementation and orchestration;
- Data product discovery, catalog registration, and publishing;
- Data governance and standardization;
- Data product lineage;
- Data product quality metrics (collection and sharing);
- In memory data caching;
- Federated identity management;
- Compute and data locality.

III. THE MASK–MEDIATOR–WRAPPER ARCHITECTURE

The MMW architecture (Fig. 3), proposed by Dončević et al. [18] is an extension of the already known MW (mediator-wrapper) architecture. The MW architecture is shown to underperform in terms of data representation, which is heavily leveraged in modern data management, so the MMW architecture proposes a new additional component type - the mask. The work by Dončević et al. [18] presents rules for MMW components, a quantitative flexibility analysis comparing the MMW with the MW architecture, and a case where a legacy store-preserving system could hypothetically be substituted by an MMW system. This paper shows how the MMW architecture accommodates more than just data integration by demonstrating that it can also support other data management purposes.

The MMW architecture is comprised of three component types: mask, mediator, and wrapper. Each of these component types deals with a specific set of tasks and responsibilities:

- The wrapper is used to encapsulate a data source and provide a universal data source interface to the rest of the system. The wrapper allows the data source to be queried for both its data and metadata. The data sources can be a variety of storage solutions: relational and NoSQL databases, data lakes, or even web content.
- The mediator is used to transform and integrate data and metadata from wrappers or lower-tier mediators. Mediators can be composed to work in multiple tiers, each tier raising the level of abstraction or extracting a piece of integrated data and metadata.
- The mask is used to represent the data and metadata it acquires from a mediator in different formats. The mask allows an MMW system to be both materializing and virtualizing. A mask can provide a raw data acquisition interface, a graphical user interface for reporting, or be used as a data warehouse materializer.

The data and metadata that the components translate and exchange are schemas, queries, and data. Schemas represent the organization of data, while queries over schemas are used to specify the data of current interest. The mask, mediator, and wrapper are prebuilt and configurable components [18]. Each component can be considered an architectural quantum, as by definition of Ford et al. [2], since they are independently deployable and cohesive. High cohesion is a product of the refined separation of concerns among component types (set by the component type rules). Independent deployability is intrinsic to the componentization in the MMW architecture; for example, a wrapper doesn’t require other components to function properly, and neither do mediators and masks (although they would present empty schemas and data sets).

A. Janus System

To enable free and open experimentation in data management research, and also prove our claims about the MMW architecture, we have developed a prototypal proof-of-concept MMW system called Janus. The source code for Janus is publicly available in its git repository [19]. Janus uses tabular data and schema models, as well as an SQL-like query and command language model for system operations and management. The choice of a tabular representation for system models is inspired by Kleppmann [20], who suggested that relational models generalize data very well, so they can be useful in a broad variety of cases beyond their originally intended scope of business data processing. Janus’ schema model consists of a data source element that contains multiple schemas. Schemas contain tableaus. Tableaus contain attributes. Janus enables the system format models to be translated into other formats for representation by masks.

Currently, Janus supports a virtualizing REST Web API mask, a materializing LiteDB mask, and a materializing SQLite mask. An experimental wrapper is implemented to support SQLite databases as data sources. The Janus source code contains frameworks for implementing mask and wrapper kinds, which enable the system to be open to further development. Mediation is specified through a mediation language inspired by Asano et al. [21]. Mediation specification incorporates specifying a tableau and its attributes and specifying a source query by
which the tableau would be virtually populated with data from the underlying data sources. Tableaus are positioned within schemas by specifying a tableau within a schema specification. The mediator uses the mediation specification to mediate data, schemas, queries, and commands.

Janus defines multiple types of response and request messages for querying, commanding, data exchange, schema acquisition, and infrastructure concerns. The message models are defined without technological coupling, hence component communication can be implemented in various technologies. Janus currently supports exchanges over TCP in multiple binary data formats, of which Avro is primarily used. Due to interfacing strategies in Janus’ design, additional communication technologies can also be implemented without affecting the core message exchange implementation. We are not aware of any such openly-available system at the time of writing this paper.

IV. Compatibility of Concepts

Although at first glance the data mesh and the MMW architecture seem to be concepts from different time periods and paradigms, we consider them compatible. They cover similar problem areas, use similar concepts, and provide a similar service. The MMW architecture is a much more primitive idea than the data mesh. Whereas the data mesh focuses more on solving organizational problems (both human and data-related), the MMW tries to solve functional granularity problems and provide comprehensive reusable architectural quanta for data management. However, Dehghani [3] and Dončević et al. [18] stress that their concepts are domain and technology-agnostic, giving their realizations the ability to prove compatible. The data mesh and the MMW architecture are orthogonal concepts, aiming to cover different problem areas in data management. Together with the promise of agnosticism, this provides the basis for the data mesh and the MMW architecture to be complementary ideas that can produce a greater effect when joined.

A. Consume-Transform-Serve

A data mesh’s DPC is expected to autonomously consume, transform, and serve data Dehghani [3]. This is visible in Fig. 2, with the input ports used to consume data, transformations being done inside the DPC, and the data served through the output ports. In the MMW architecture (Fig. 4), data consumption is driven by the wrappers, transformation is driven by mediators, and data serving is driven by masks. By providing consume-transform-serve capabilities, the components of an MMW architecture are capable of acting as a data product.

B. Data Models

Data models in domain-driven design are instinctively thought of as constructs of classes or structs. This is because domain-driven design is usually observed in operational systems. In analytical systems, this could be a detriment because the consumed data might be defined by different metamodels and it’s expected that the data is served in polyglot form. Dehghani [15] also recognizes this stating: Depending on the nature of the domain data and its consumption models, data can be served as events, batch files, relational tables, graphs, etc., while maintaining the same semantic. Rather, domain-driven design should be considered at the most abstract level. Kleppmann [20] has explained why relational systems have stood the test of time: relational databases turned out to generalize very well, beyond their original scope of business data processing, to a broad variety of use cases. The relational model, with its generalization abilities and time-enduring legacy in analytical systems, should be a good fit for the data mesh. Nothing prevents a domain from being described in a relational model - this is done regularly in operational systems.

A relational model can also be used in an MMW architecture. The use of a tabular relational-like model in mediators has already been demonstrated by Asano et al. [21] and as a system-wide model in Janus. Support for polyglot access would also be significantly easier to research and implement, as there are already numerous implementations of relational mapping to other models. This is the focal point of the MMW architecture’s capability to be domain-agnostic while enabling relational mapping to other models. This is the focal point of the MMW architecture’s capability to be domain-agnostic while enabling relational mapping to other models. The MMW tries to solve functional granularity problems and provide comprehensive reusable architectural quanta for data management. However, Dehghani [3] and Dončević et al. [18] stress that their concepts are domain and technology-agnostic, giving their realizations the ability to prove compatible. The data mesh and the MMW architecture are orthogonal concepts, aiming to cover different problem areas in data management. Together with the promise of agnosticism, this provides the basis for the data mesh and the MMW architecture to be complementary ideas that can produce a greater effect when joined.

C. Evolutionary Compatibility

Dehghani [3] stated that: “it’s only appropriate to [...] leave the specific implementation details and technology to be refined and built over time. [...] any specific implementation design or tooling suggestion will be simply outdated by the time you get to read this book”. Technology is considered capricious in this research area, and it is now common sense to avoid any technological dependency when discussing architectures and concepts in software engineering. Technological changeability
is one of the driving values of evolutionary architectures [2], from which Dehghani [3] derives the suggestions that are made about the data mesh’s hypothetical implementation. Essentially, a data mesh should be evolvable, and the MMW architecture should follow suit if it is to be compatible.

The MMW architecture fits the common dimensions of evolvability as follows:

- **Technical**
  The inner components of the mask, mediator, and wrapper component types are finely grained and their core functionalities can be separated by interfacing from technologically changeable inner components. This allows the MMW components to be adapted to different technologies (e.g., in terms of communication protocols and data formats). This decoupling also allows components supporting newer or previously unsupported technologies to be developed quickly.

- **Data**
  Data shared across an MMW system is both metadata and data, described in a generalized manner. Changes to the core metamodels are expected to be infrequent, but to minimize the effects of changes, the definitions can be placed in a shared library.

- **Security**
  Security related to authentication, authorization, confidentiality, and data integrity for communication requirements can be interfaced alongside communication protocols and data formats. Security in terms of logical operations (e.g., sharing confidential system information or sending unsafe data) can be assured through a common library for each component type as well as defining a standard set of inter-component exchanges.

- **Operational/System**
  Since each MMW system component is an architectural quantum, mapping the components to existing infrastructure is very flexible.

  The evolvability of the MMW architecture allows it to uphold the data mesh’s set of conventions that promote interoperability between different technologies, which guarantees its longevity alongside the data mesh concept.

### D. Capability Coverage

A set of capabilities the data mesh offers was mentioned in Section II. The MMW can cover these fifteen capabilities as follows in Table I.

#### 1) Constructing a Data Mesh Platform:

An example of an MMW-driven data mesh for a fictitious music streaming service is shown in Fig. 5. The music streaming service is separated into four domains. The music domain deals with data concerning music information; artists, albums, and tracks. The customers domain is concerned with data about customers that use the music streaming service; names, usernames, and contact information. The listening domain deals with data concerning the listening histories of customers; tracks and the media type of the track that was listened to. The subscriptions domain is concerned with customer subscription plans; the billed customer for a plan, and the customers assigned to those plans as beneficiaries. As an example of a possible topology, each domain contains an operational system serving operational data. The DPC from each domain serves analytical data. DPCs consume data from their domain’s operational system and the DIP.

A DIP should solve the need for duplicating data pipelines, storage, and streaming infrastructure [3], making it the platform for uniform data access. This platform can be fittingly driven by multiple wrappers. Each wrapper covers a single data source. This provides standardized access to data sources and

| Scalable polyglot big data storage | The MMW architecture allows usage of heterogeneous data source and heterogeneous representation. |
|-----------------------------------|--------------------------------------------------------------------------------------------------|
| Encryption for data at rest and in motion | MMW components can interface their communication with encryption protocols. Encryption of static data comes down to the encryption of local data stores. |
| Data product versioning | Data products can be versioned over a metamodel. As a hypothetical example over a relational model, tables are grouped in schemas, and schemas represent specific versions of data products. |
| Data product schema | A data product schema is provided by default in each component because queries are declared over schema. |
| Data product de-identification | De-identification can be done during data transformation in mediators, or via specialized instructions for wrappers. |
| Unified data access control and logging | Data access can be controlled in all MMW component types. Masks can also provide access control via their applications. |
| Data pipeline implementation and orchestration | Data pipelines are implemented by composing the mask, mediator, and wrapper components to consume, transform, and serve data. |
| Data product discovery, catalog registration, and publishing | Data product discovery is enabled by examining schemata provided by components. This process is simplified since all component types have common interfaces. |
| Data governance and standardization | Data governance is federalized because MMW components can work in separate groups. Standardization is provided by the MMW components’ standard interfaces. |
| Data product lineage | Lineage can be overseen by looking into mediator transformations, and mask and wrapper translations. |
| Data product monitoring/alerting/log | Each MMW component can be deployed along with a monitoring application. Logging is expected to be component-level in all component types. Alerting interfaces can also be put in place as part of the core component or the monitoring application. |
| Data product quality metrics (collection and sharing) | Quality metrics can be shared as a part of the schema metadata. This allows quality metrics to be defined for each data product version separately. |
| In-memory data caching | In-memory data caching can be implemented for each component type to optimize query response times. |
| Federated identity management | Identity management is a part of the infrastructure capabilities, but the MMW architecture doesn’t prohibit or discourage such cases. |
| Compute and data locality | Data can be locally transformed in mediators and placed in local stores using materializing masks; then to be consumed by a wrapper on request and served (see Section IV-D1). |
Fig. 5. MMW architecture used to drive a data mesh for an exemplified music streaming service.

The exemplified music streaming service from Fig. 5 will be used in the experiment presented in Section VI, with some minor alterations.

2) Constructing a Data Product With Localized Storage: Corresponding to the product container presentation in Fig. 2, Fig. 6 displays how a DPC can be driven by MMW components if it contains a local domain data store. The DPC’s MMW components are grouped as either ingress or egress components. Ingress components are used to acquire and prepare data for further serving. Egress components are used to finalize and serve data (as the data product). These two component groups can work asynchronously. Data in a DPC is acquired either as a data product from another domain directly by an ingress mediator or by consumption of the domain operational data via an ingress wrapper. The ingress mediator transforms the data in preparation for it to be stored as a prefabricated data product in the domain data store. An ingress mask then materializes the data in the domain data store. The persisted data is extracted for further use by an egress wrapper. The egress mediator uses the egress wrapper and ingress mediator to acquire data for finalization and serving. Using the egress wrapper enables the egress mediator to acquire the prefabricated data product while using the ingress mediator allows the egress mediator to acquire the most recent data. As will be shown in Section VI, although the prefabricated data might not contain the most recent data,
acquiring it is less computationally demanding than acquiring the most recent data. This is a trade-off to consider. The egress mediator is used to finalize the data product and to provide a data port for other DPCs (via their own ingress mediators). The egress mask acquires data from the egress mediator and represents it with an API that was prescribed by the data mesh governance.

The aforementioned cases presented in Fig. 5 and Fig. 6 demonstrate how the MMW components can drive the data mesh boilerplate. The cases also illustrate that custom-built adapters are not required for the MMW components to fit the data mesh’s functionality.

V. WHAT ARE THE BENEFITS?

Using the MMW architecture to drive a data mesh creates benefits regarding an organization’s adoption of the data mesh and its technical maintenance. We propose that the MMW architecture has four major benefits when utilized to drive a data mesh: low-risk adoption trials for organizations whose data might not be at the appropriate scale, rapid prototyping to expedite implementation time in the adoption process, evolvability to enable the longevity of the system, and standardization to lower the cognitive load on technicians and facilitate composability.

A. Low-Risk Adoption Trials

The data mesh promises getting value out of data at a scale. Determining that scale currently remains a rule of thumb, and it’s unlikely that a metric will be proposed until a larger number of organizations try to adopt the data mesh - failing or succeeding. The point remains that the data mesh is intended for organizations that store various and voluminous data. Would a small local convenience store or a local accountant’s office benefit from a data mesh? Probably not. Would a streaming service or an online market service benefit from a data mesh? Very likely.

Those organizations that are considered among the two exemplified groups are at the most risk. Adopting a new analytical system, in a novel architecture, and built from scratch is not a simple undertaking. Many human and economic factors can lead the adoption astray. The system might be too complex to use in a simpler business environment, it might increase latency without providing any tangible analytical flexibility, and it might not even be developed to the product level, leaving the analytical capabilities of an organization in disarray. Even worse, if engineers unwittingly use the elephant migration anti-pattern [22], they might find themselves with an unfinished data mesh and a partially dismantled legacy system. This is all without mentioning the time, human, and financial resources invested. These cases are expected to be common, as just 20 percent of analytic insights are expected to deliver business outcomes through 2022 [8], and 87 percent of data science projects never make it into production [9].

Organizations can greatly reduce the risk of adopting a data mesh by setting up a trial run using the MMW architecture. The use of the MMW architecture lowers the risk of:

- **Development failure**
  MMW components require development only if specialized components are needed. This is an edge case if wrappers and masks are not developed for certain data source types or data representations. Components are primarily expected to just be acquired, deployed, and configured. There is no extensive coding required.

- **Loss of large resource investment**
  Since there is no extensive coding required when using the MMW architecture, a small technical or development team could prepare a demonstrative data mesh in a short time. The financial investment can boil down to the price of additional infrastructure (if needed), and that could be constrained to a cloud service so no additional hardware acquisition is required.

- **Deteriorated business usability**
  The MMW architecture doesn’t require the legacy analytical system to be dismantled, so a possible deterioration of service is limited to the time frame of the adoption trial run. If the data mesh is found unsuitable for the organization, then it can be easily dismantled and the analytical system reintroduced.

B. Rapid Prototyping

The MMW architecture allows the data mesh to be rapidly prototyped when initiating analytical capabilities in an organization. The system is to be realistically set up. The initial step is to deploy and configure MMW components to drive a prototypical data mesh. When requirements are fully distilled over the prototype, the MMW-driven data products can then be substituted piecemeal with permanently developed ones. The DIP is migrated incrementally as permanent data products are developed and deployed.

Rapid prototyping via the MMW architecture allows the data mesh to be expeditiously deployed in an organization’s environment, so the benefits of a data mesh can be reaped as soon as possible (e.g., merging data silos). It enables early problem detection, faster business processes, and organizational structure alignment, as well as bringing an increase in business product value sooner.
C. Evolvability

It is expected that components driving the data mesh form an evolvable architecture. In reality, the software development industry is known to omit beneficial architectural and design system properties for the sake of reaching a minimum-viable product and creating profit quickly. While the consideration of evolvability is seen as obvious, it is questionable how many software development projects will continue to follow evolvability principles throughout their life cycles. Another problem is that senior developers, designers, or architects tend to stack technologies instead of components and quanta when working on systems.

It was shown in Section IV-C that the MMW architecture is evolvable. A composition of evolvable architectural quanta transiently makes their composition evolvable, hence MMW components can be used as a set of building blocks for a data mesh to guarantee evolvability.

D. Standardization

Since the data mesh concept proposes no explicit implementation or use of technology, it’s expected that organizations will implement data meshes as custom-built platforms. According to Conway’s law [23], organizations will implement the data mesh according to their specific knowledge base; using specific design patterns and technologies used by their technologists. Because of this, it is expected that each data mesh implementation will be significantly different from another. The discrepancy will lead to a lack of standardization, which can cause: a lack in the composability of multiple data meshes in case of organizational mergers and acquisitions, an increased learning curve for newer technologists, and a significant learning curve for experienced technologists switching projects (this diminishes their existing skill-set and makes it largely unusable), or an implementation of a data mesh far removed from the original concept with detrimental effects on the organization.

Dehghani [3] stresses the importance of lowering the cognitive load of developers by using experiences, languages, and APIs that are easy to learn as a starting point. With the MMW architecture, a step further can be taken, and standardization of architectural quanta created with which data meshes are regularly developed. This further lowers the cognitive load, allowing technologists to be calmly migrated between projects. Standardization benefits the composability of systems, the composers use the same interfaces to function. It can be stated that standardization lowers degrees of implementation freedom, but it brings the beneficial effect of keeping projects close to the original data mesh concept.

VI. EXPERIMENTS

A. Functional Experiment

We have prepared a functional experiment to support our claims that the MMW architecture can drive a data mesh. The goal of the experiment was to emulate the data mesh illustrated by Fig. 5. We used Docker to containerize Janus’ components for simplification and standardization of their deployment. We specified a data mesh deployment by combining multiple container images through a Docker Compose specification. Multiple SQLite databases were used to act as the DIP data sources, as well as the domain data store within a DPC. The databases were created and populated with fake representative data by using the Faker library in Python. Janus’ SQLite wrapper components were used for wrapping the SQLite databases. SQLite materializing masks were used for materializing the DPCs’ domain data stores. REST Web API masks were used to represent the serving interfaces of DPCs, enabling their access via HTTP requests or the Swashbuckle Swagger GUI. The components are configured to execute their startup operations automatically; registering remote components, loading schemas, applying mediations, materializing databases, and starting REST Web API instances. The Music, Customers, Listening, and Subscriptions DPCs were specified and deployed for this experiment. The operational systems were omitted from this experiment because they provide additional levels of complexity to the experiment without adding any valuable insight. The operational systems might as well have been represented by some of the DIP wrappers.

Fig. 7 represents the environment of the MMW-driven listening DPC. The two DIP wrappers used in this case deal with raw listening data and media-type instances of tracks. The customer and general music data are provided as data products by the Music and Customers DPCs. The aforementioned data is acquired by the Listening DPC’s ingress mediator, which constructs a ListeningIngressData schema. This schema is loaded into the ingress mask and, along with the acquired data, is used to materialize the domain data store as an SQLite database. The egress wrapper infers the domain data store’s schema and exposes the ListeningDomainData schema and its data to the egress mediator. The egress mediator mediates the ListeningIngressData and ListeningDomainData schemas by representing them as Fresh and Stable schemas, respectively. When the Fresh schema is queried, then the data is acquired directly through the ingress mediator and its connected components. When the
Stable schema is queried, then the egress wrapper is used to acquire the data from the domain data store. The dual ListeningData schema is singularly represented by Janus’ data source schema model element, and the Fresh and Stable schemas are represented by separate schema elements in the data source. The egress mask loads the ListeningData schema from the egress mediator and represents it as a REST Web API. The dual schemas are represented as separate “/Fresh” and “/Stable” route prefixes, with each tableau in those schemas being an individual resource route. Tableaus are mapped to DTOs as resources. The schema management strategy is presented in Fig. 7 by white rectangles which represent individual schemas next to their respective components.

The required databases and specification files for the experiment are located in the Janus git repository [24]. The experiments can be easily reproduced using these files through Docker Compose.

B. DPC Data Acquisition Experiments

To show the general acquisition time of the Janus-driven DPCs, and to provide evidence that a domain data store is a useful mechanism we conducted a time-measuring experiment over the Fresh and Stable schemas of the Customers and Subscriptions DPCs. Due to the constraints in the computing resources of the accessible equipment, we opted to use these two DPCs instead of the more demanding Listening DPC. The Fresh and Stable schemas of the Subscriptions DPC contain 3 tableaus: SingleUserSubscriptions with 330 entries, DualUserSubscriptions with 10 entries, and MultiUserSubscriptions with 18 entries. The Customers DPC contains a Customers tableau in both the Fresh and Stable schema with 400 entries. A Python script was utilized to send 100 HTTP GET requests to each URL and measure their response times. The Docker Compose with the required components was executed on two machines. Machine A has an 11th Gen Intel® i7-11700F processor and 32 GB of RAM. Machine B has a 7th Gen Intel® i5–7300HQ processor and 16GB of RAM. We used machines of objectively different computing power to show that the trends we measured can be taken as general rules. Results are summarized in graphs of Fig. 8 and Fig. 9.

To quantify the difference in response times between tableaus of the Fresh and Stable schemas we used the following formula:

\[ P = \frac{\sum_i^4 \text{Med}_{fresh} - \text{Med}_{stable}}{4} \times 100[\%] \]  

(1)

The measurements taken from machine A show that the Stable schema has an average reduction in response time of approximately 28% in comparison with the Fresh schema. The measurements on machine B show an even more pronounced reduction of approximately 64%. We have confirmed that the reduced acquisition time from a domain data store is a global trend across machines and can be taken as a general rule.

The required databases, specification files, and scripts for the experiment are located in the Janus git repository [24]. The experiments can be easily reproduced using these files through Docker Compose and running the scripts in Python.

VII. Conclusion

The data mesh is a promising concept, but it is still in its infancy. As with any such new concept, it lacks concerted best practices and standards. These will undoubtedly arrive as real-world development experience is gathered by early adopters. Potential data mesh adopters may come from a multitude of business areas and come in different organizational sizes, so it would be beneficial for them to run a trial data mesh before committing to full adoption. Standardization of a data mesh allows the system to be composable with other data mesh implementations and allows technologists to migrate from project to project with a minimal cognitive load. The use of an MMW architecture to drive a data mesh addresses all of those concerns.

We have proposed that the MMW architecture and data mesh are orthogonal concepts - the data mesh concerns itself with the organization of data management, while the MMW architecture solves functional granularity problems for quanta in data management. Our findings are that these concepts are compatible in terms of consume-transfer-serve functionalities, data modeling, evolvability, and coverage of capabilities. The compatibility and orthogonality of the data mesh and MMW architecture concepts mean that they can be used side-by-side.
The benefits of an MMW-driven data mesh we have proposed are the ability to run low-risk adoption trials, the ability to develop a data mesh using rapid prototyping, guaranteed evolvability, and standardization. Both early and late adopters can find these benefits useful. Organizations at the edge of managing large-scale data can run trials to see if a data mesh suits them. Organizations that require a data mesh to be developed quickly can use the MMW architecture to rapidly prototype it. Organizations concerned with standardization and evolvability can adopt the MMW-driven data mesh as a complete solution.

We have conducted reproducible experiments using an existing MMW system to prove that a data mesh can be driven by MMW components. The system used in the experiments was the proof-of-concept MMW system of our own creation, called Janus. The experiments have shown that an MMW-driven data mesh is achievable. The experiments have also demonstrated the use of a DPC with domain data store, and its benefit of acquisition time reduction.

Looking to future research, the MMW architecture might also be used to drive other data management architectures (e.g., data hub, data fabric, data spoke). Additionally, this would mean that our Janus system might not just be a data source integration system, but an adaptable data management platform in the making. If these suspicions are shown to be true, as in the case of the data mesh, further research might also explore the ability of the MMW architecture to drive migrations to other architectures. This includes future state-of-the-art architectures. The MMW architecture might prove to be a long-standing driver that enables technologists and their organizations to execute generational migrations between data management architectures with reduced risk. This could provide standardized and low-risk adoption of state-of-the-art architectures while allowing the technological environment to evolve.

REFERENCES

[1] C. González García and E. Álvarez Fernández, “What is (not) Big Data based on its 7Vs challenges: A survey,” Big Data Cogn. Comput., vol. 6, no. 4, Dec. 2022, Art. no. 158. Accessed: May 31, 2023. [Online]. Available: https://www.mdpi.com/2504-2289/6/4/158

[2] N. Ford, R. Parsons, and P. Kua, Building Evolutionary Architectures: Support Constant Change, 1st ed. Beijing, China: O’Reilly Media, 2017.

[3] Z. Dehghani, Data Mesh: Delivering Data-Driven Value at Scale, 1st ed. Beijing, China: O’Reilly Media, 2022.

[4] A. Nguen et al., “Data movement in Netflix Studio via data mesh.” Netflix. Accessed: Aug. 15, 2022. [Online]. Available: https://netflixtechblog.com/data-movement-in-netflix-studio-via-data-mesh-361dcee61059

[5] B. Lei, G. Pires, J. Shao, K. Chatterjee, S. Jain, and V. Sydorenko, “Data mesh—A data movement and processing platform @ Netflix.” Netflix. Accessed: Aug. 15, 2022. [Online]. Available: https://netflixtechblog.com/data-mesh-a-data-movement-and-processing-platform-netflix-1280c5cb2873

[6] D. Joshi, S. Pratik, and M. P. Rao, “Data governance in data mesh infrastructures: The Saxo Bank case study,” in Proc. 21st Int. Conf. Electron. Bus. (ICEB). Nanjing, China, Dec. 3–7, 2021, pp. 599–604.

[7] Databricks. Data Mesh in Practice: How Europe’s Leading Online Platform for Fashion Goes Beyond the Data Lake. Jul. 28, 2020. Accessed: Aug. 9, 2022. [Online Video]. Available: https://www.youtube.com/watch?v=eiUtVYYnVUC

[8] A. White. “Our top data and analytics predicts for 2019.” Gartner. Accessed: Sep. 1, 2022. [Online]. Available: https://blogs.gartner.com/andrew_white/2019/01/03/our-top-data-and-analytics-predicts-for-2019/

[9] VentureBeat. What the Heck Does It Even Mean to “Do AI”? | Business AI Integration | VB Transform 2019. Jul. 15, 2019. Accessed: Sep. 1, 2022. [Online Video]. Available: https://www.youtube.com/watch?v=Ezm7ZHIo-EI

[10] Z. Dehghani. “How to move beyond a monolithic data lake to a distributed data mesh.” martinfowler.com. Accessed: Mar. 31, 2022. [Online]. Available: https://martinfowler.com/articles/data-mesh-monolith-to-mesh.html

[11] M. Richards and N. Ford, Fundamentals of Software Architecture: An Engineering Approach, 1st ed. Sebastopol, CA, USA: O’Reilly Media, 2020.

[12] E. Evans, Domain-Driven Design: Tackling Complexity in the Heart of Software 1st ed. Boston, MA, USA: Addison-Wesley Professional, 2003.

[13] A. Cockburn. “Hexagonal architecture.” cockburn.us. Accessed: Aug. 14, 2022. [Online]. Available: https://alistair.cockburn.us/hexagonal-architecture/

[14] D. Svirat and S. Makagon, “Ready for changes with hexagonal architecture.” Netflix. Accessed: Aug. 14, 2022. [Online]. Available: https://martinfowler.com/articles/ready-for-changes-with-hexagonal-architecture-b315ec967749

[15] Z. Dehghani. “Data mesh principles and logical architecture.” martinfowler.com. Accessed: Mar. 29, 2022. [Online]. Available: https://martinfowler.com/articles/data-mesh-principles.html

[16] K. Waechner. “The heart of the data mesh beats real-time with Apache Kafka.” Accessed: Aug. 2, 2022. [Online]. Available: https://www.kai-waechner.de/blog/2022/07/28/the-heart-of-the-data-mesh-beats-real-time-with-apache-kafka/

[17] V. Desai, T. Fountaine, and K. Rowshankish. “Manage data like a product to unlock full value.” McKinsey & Company. Accessed: Aug. 15, 2022. [Online]. Available: https://www.mckinsey.com/business-functions/quantumblack/our-insights/how-to-unlock-the-full-value-of-data-manage-it-like-a-product

[18] J. Dončević, K. Fertalj, M. Brečić, and A. Krajna, “Mask–mediator–wrapper: A revised mediator–wrapper architecture for heterogeneous data source integration,” Appl. Sci., vol. 13, no. 4, Jan. 2023, Art. no. 2471. Accessed: Apr. 5, 2023. [Online]. Available: https://www.mdpi.com/2076-3417/13/4/2471

[19] J. Dončević. “Janus.” GitHub. Accessed: May 29, 2023. [Online]. Available: https://github.com/JurajDoncevic/Janus

[20] M. Kleppmann, Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems, 1st ed. Boston, MA, USA: O’Reilly Media, 2017.

[21] Y. Asano et al., “A view-based programmable architecture for controlling and integrating decentralized data,” 2018, arXiv:1803.06764.

[22] N. Ford, M. Richards, P. Sadalage, and Z. Dehghani, Software Architecture: The Hard Parts: Modern Trade-Offs Analyzed for Distributed Architectures, 1st ed. Sebastopol, CA, USA: O’Reilly Media, 2021.

[23] M. E. Conway, How Do Committees Invent?, New York, NY, USA: F. D. Thompson Publications, Inc., 1968, p. 4.

[24] J. Dončević. “Janus—Data mesh experiments.” GitHub. Accessed: May 29, 2023. [Online]. Available: https://github.com/JurajDoncevic/Janus

[25] Juraj Dončević (Graduate Student Member, IEEE) received the Ph.D. degree in computer science from the Faculty of Electrical Engineering and Computing (FER), University of Zagreb. He is a Teaching Assistant with the FER. His research interests include software architectures, data integration, and bidirectionalization.
Krešimir Fertalj (Senior Member, IEEE) is a Full Professor with the Faculty of Electrical Engineering and Computing (FER), University of Zagreb, Croatia, where he lectures a couple of computing courses on undergraduate, graduate, specialist, and doctoral studies. His professional and scientific interests include automated software engineering, complex information systems, project management, and software security. He led several scientific and research projects and a few dozen development projects. He was a mentor to students for over 250 bachelor and graduate theses, nine M.Sc., and eleven Ph.D. theses. He has published nearly 200 scientific and technical papers. He is the Founder of the Laboratory for Special Purpose Information Systems and of the Postgraduate Specialist Study “Project Management” with the FER. He is a Full Member of the Croatian Academy of Engineering (HATZ). He served as a Head of Department with the FER, a Secretary with the Department of Information Systems, HATZ, and was one of the founders and members of the management board of the PMI chapter in Croatia.

Mario Brčić (Member, IEEE) received the Ph.D. degree in computer science from the Faculty of Electrical Engineering and Computing, University of Zagreb, in 2015. He is currently an Assistant Professor with the University of Zagreb. His research interests include the crossroads of artificial intelligence and operations research. He is a member of ACM and AGI Society.

Mihael Kovač (Graduate Student Member, IEEE) received the master’s degree from the Faculty of Electrical Engineering and Computing (FER), University of Zagreb. He is working toward the Ph.D. degree with the FER. His research interests include compilers, heterogeneous systems, and deep learning.