Linking Stakeholders’ Viewpoint Concerns and Microservices-based Architecture

Mujahid Sultan†,*

† Department of Computer Science, Ryerson University, Toronto, Canada
* Treasury Board Secretariat, Government of Ontario, Toronto, Canada
mujahid.sultan@ryerson.ca;ontario.ca

ABSTRACT
Widespread adoption of agile project management, independent delivery with microservices, and automated deployment with DevOps has tremendously speedup the systems development. The real game-changer is continuous integration (CI), continuous delivery and continuous deployment (CD). Organizations can do multiple releases a day, shortening the test, release, and deployment cycles from weeks to minutes.

Maturity of container technologies like Docker and container orchestration platforms like Kubernetes has promoted microservices architecture, especially in the cloud-native developments. Various tools are available for setting up CI/CD pipelines. Organizations are moving away from monolith applications and moving towards microservices-based architectures. Organizations can quickly accumulate hundreds of such microservices accessible via application programming interfaces (APIs).

The primary purpose of these modern methodologies is agility, speed, and reusability. While DevOps offers speed and time to market, agility and reusability may not be guaranteed unless microservices and API’s are linked to enterprise-wide stakeholders’ needs. The link between stakeholders’ needs and microservices/APIs is not well captured nor adequately defined.

In this publication, we describe a structured method to create a logical link among APIs and microservices-based agile developments with enterprise stakeholders’ needs and viewpoint concerns. This method enables capturing and documenting enterprise-wide stakeholders’ needs, whether these are business owners, planners (product owners, architects), designers (developers, DevOps engineers), or the partners and subscribers of an enterprise.

KEYWORDS
Microservices, DevOps, Stakeholder Viewpoints, Requirements Engineering, Enterprise Architecture

1 INTRODUCTION
The IT industry has seen many transformations in the Software Development Life Cycle (SDLC) methodologies and development approaches. SDLCs ranging from waterfall to agile, and the development approaches from monolith to microservices. The fundamental difference between the monolith and the microservices-based architectures is the codebase. Monolith applications work of a central and relatively large codebase, whereas microservices have an independent and relatively small codebase. Due to the agile SDLCs, new microservices can be designed very quickly and deployed in production immediately. Agile management practices and microservice designs culminated in a new way of developing software - the DevOps, where Continuous Integration (CI) and Continuous Delivery (CD) play a major role. Most of the medium to large organizations, like public and semi-public entities who remained married to structured methodologies (like enterprise architecture frameworks) of creating enterprise blueprints, find it very difficult to transition to this new software development approaches.

Business process management (BPM) and Enterprise Architecture (EA) disciplines emerged to streamline the business and systems gap. With the adoption of agile methodologies and DevOps, this gap is widening. EA management is a practice to document relationships among businesses and systems. Many EA frameworks have evolved [45], and have received some maturity [48] over the past decade, TOGAF, FEAF, and Zachman, to name a few. A comprehensive review of EA and EA frameworks given in [23, 38, 57]; and evaluating EA frameworks given in [39, 47, 53]. EA frameworks consist of artifacts, which are descriptions of the enterprise from a specific viewpoint† of a group of stakeholders [30, 46]. The stakeholders’ groups are owners, designers (architects), systems engineers, developers and DevOps.

John Zachman introduced the concept of Information System Architecture (ISA) in 1987 [60]. The Zachman framework describes stakeholders’ views focusing on five “wh” interrogatives (what, who, where, why, and when) and one “h” interrogative (how). This focus comes from journalism’s w5h theory [31]. Sultan and Miranskyy [50, 51] applied linguistic findings to establish that the w5h set of interrogatives is not complete and added interrogative which. They denoted this new set of seven interrogatives as w6h. They demonstrated that asking questions, based on the w6h (in the order of precedence described in Figure 1), improves information flow for the description of stakeholders’ viewpoint concerns.

While EA frameworks for monolith applications have matured, there is no structured approach for defining EA for microservices-based developments. In this publication, we extend the w6h EA framework and w6h requirements engineering pattern developed by [50, 51] to the microservices-based developments.

Roadmap: The rest of the paper is organized as follows. In Section 2, we explore the literature on defining EA for microservices-based developments. In Section 3, we describe the technologies which enable microservice architectures. This section level sets the ground for the proposed method - given in Section 5. In Section 6, we

† We use ANSI/IEEE Standard 1471-2000 definitions of stakeholders’ views and viewpoints [28]. A view is a representation of a whole system from the perspective of a related set of concerns. A viewpoint defines the perspective from which a view is taken. A viewpoint is where you are looking from - the vantage point or perspective that determines what you see; a view is what you see.
discuss the proposed framework and its suitability in the creation of EA framework for microservice-based developments. In Section 7 we give some future research directions.

2 RELATED WORK

Thönes [54] and Nadareishvili et al. [43] provided background on microservices. Garriga [33] provided a complete analysis of microservices taxonomy, life cycle, and fitness in an organization. Benchmark requirements for microservices architecture research are given by [19]. Challenges in Documenting microservice-based IT landscape is captured in the survey paper by Kleehaus and Matthes [34] and Soldani et al. [49].

Reflections on SOA and microservices are presented by Xiao et al. [58]. Contextual understanding of microservice architecture “current and future directions” is given by Cerny et al. [27]. Microservices architecture enablement with DevOps are investigated by Balalaie et al. [22]. Architectural patterns for microservices are given by [52].

There have been reasonable efforts and systematic approaches to identifying microservices from a monolith’s system functional requirements [55]. Several useful techniques for extracting microservice from monolith enterprise systems are given by [40], Mazlami et al. [41] proposed a formal approach to extract microservices from monolith applications based on the class structure but did not consider stakeholders’ viewpoint concerns. O’Connor et al. [44] presented background on CD and CI and the DevOps enabling technologies but did not talk about the link between EA and microservices.

Kratzke and Peinl [36] presented the anatomy of the cloud-native stack which is a good summary of components of cloud-native developments but it does not address the full set of enterprise stakeholders. A reference architecture for designing microservices is given by Yu et al. [59]. They presented a good representation of the microservice-based reference architecture model but kept themselves at the technology view level. Bogner and Zimmermann [25] investigated mechanisms for integrating microservice architecture and EA and presented a very high-level meta-model without getting into the details of the stakeholders’ viewpoint concerns.

None of these studies considered alignment between enterprise stakeholders’ viewpoint concerns and microservices. Our work gives a structured approach to address these concerns systematically. The need to define a linkage between enterprise stakeholders’ needs and microservices is well established in the literature. Canat et al. [26] in their recent paper titled ‘Agile developments foe or friends with EA’ found that 1) agile development and enterprise architecture can be combined, 2) there are clear communication problems among architects, different teams, and project owners, and 3) there is a lack of system and application reusability.

Our contribution in this work is to address these issues. We propose a framework, a structured approach, and a mind-map to capture enterprise stakeholders’ viewpoint concerns for microservice-based developments. This framework can be used for analysis, reuse and strategic planning, filling the gap between enterprise architects and DevOps.

3 BACKGROUND

The primary advantage, microservice-based architecture (as compared to monolith) offers, is independent development (in a language of choice), independent deployment, and loose coupling or access via the APIs. This paradigm warrants microservices to have their own piece of the data. The significant difference between monolith applications and microservices-based systems is shown in Fig. 2. Similarly, this new way of designing systems puts the events at the center compared to data-centric monolith application architectures. As shown in Fig 3, the enterprise’s source of truth becomes events rather than a central repository.

Microservices are directly related to the business taxonomy of an enterprise and enable business process automation. APIs open an organizations’ business processes for automation within the organization as well as for the business partners. APIs carry business value and the way the rest of the world interacts with an organization and are directly related to the enterprise stakeholders’ viewpoint concerns. Microservices, on the other hand, automate business functions, sponsored by business stakeholders. Therefore, exploring the stakeholders’ viewpoint concerns for designing microservices and APIs is in order.

Microservices are discrete components of a system function or act as a composite application automating one or more business processes. Microservices are stand-alone applications automating business processes. One API can invoke many microservices to deliver a business function representing some part of the business process; this means the APIs can serve as a “contract” between two agents - providing the data or services (the output) in exchange for valid input. These attributes make APIs a valuable tool for business process automation.

To understand the link between Microservices and the Stakeholders’ viewpoint concerns, we briefly describe the enabling technologies which lead to the development of microservices in Section 3.1 through Section 3.7.

A brief description of microservices, containers, and orchestration mechanisms is given below to put the stakeholders’ viewpoints in the perspective of DevOps (where the developers are the ones
3.1 Cloud-Native Application Platforms
With the advent of cloud computing (on-demand availability of computer resources without direct active management by the user), most of the compute services can be offered as a service, for example, infrastructure as a service (IaaS), platform as a service (PaaS) and software as a service (SaaS). The microservices revolution happened with the advent of cloud-native developments on IaaS, which means that the organizations can develop their applications with their own data; the provider manages infrastructure only.

3.2 Microservices
A microservice is a complete application in itself, full filling an entire business function, with its own development methodology and technology stack. Microservices are attributed as radially available and on-demand consumable. Not all microservices are accessible to the outside world; these are called system or interface microservices. Similarly, some microservices are designed for synchronisation or cleaning and are not discoverable or consumable on demand. Therefore, we can classify microservices in three broad categories, a) presentation-level, b) system-level, c) and integrity, or consistency check microservices. Each microservice has a different set of stakeholders’ groups associated with it. Therefore, capturing and aligning their viewpoint concerns is essential for the enterprise.

The most common access mechanism for microservices is API endpoints. The endpoints need to have a registry for internal as well as external use. Externally these are used via the reverse proxy, whereas internally via some discovery and orchestration mechanism. The microservice architecture style’s primary purpose is reusability, necessitating the need to align and link these APIs with the stakeholders’ viewpoint concerns. Different stakeholders pay for different microservices in an enterprise. Therefore, their needs should be accounted for, in the design of the microservices.

One of the core principles of the microservice developments is independent deployment. Independent deployability allows on-demand scaling. Suppose some part of the system experiences high load; in this case, we can re-deploy or move a microservice to an environment with more resources. Without having to scale up hardware capacity for the entire enterprise system, this became possible with the maturity of the container technologies - briefly described below.

3.3 Containers
Though containers have been around for a long time from Linux, Solaris [37], LXC [24], Virtuozzo [8] to Docker, the real change was brought by Docker containers [24]. Docker [42] isolated resources that enable the packaging of applications with all dependencies installed, allowing the apps to run anywhere. Docker containers offer portability, performance, agility, isolation, and scalability of the applications. Despite the technical superiority, the platform’s popularity is mainly due to the image registry (image of a container), where creators can create images, and the consumers can download and use, called Docker-hub. The registry can be public as well as private. Once the organization creates an image, it can be stored in the repository and pulled when and where needed - making this a convenient feature in a microservice architecture.

3.4 Container Orchestration
Once containers are created, these can be deployed anywhere independently; the need arises to orchestrate the deployment of these containers on different infrastructure segments. Many products have emerged in this space, for example Docker Swarm [17], Kubernetes [13], Apache Mesos [10], and Nomad [18]. However, Kubernetes have gained the most popularity and is widely used. Kubernetes (K8s) is an open-source system for automating deployment, scaling, and managing containerized applications. The docker containers are hosted in pods as a basic unit of computations in K8s. The
major components of Kubernetes are a master node and a worker node. Worker nodes run a Docker engine and can host multiple containers. An agent called Kubelet runs at each worker node, and the Kube-proxy handles each node networking. The container-runtime is responsible for running the containers on each worker. The master node components are Kube-apiserver which is a control plane and controls entire workers and pods, Kube-scheduler schedules pods and Kube-controller-manager performs cluster operations and a distributed key-value store called Etcd, which stores the data and maintains states.

3.5 Service Mesh

When the number of microservices grows in number and complexity, the interaction between these becomes challenging - the discovery, load balancing, failure recovery, metrics, and rate-limiting, etc. The Service Mesh - is a networking component that sits on top of the container orchestration layer, and takes care of these issues with custom control planes e.g., Istio [9], Linkerd [7] and Conduit [3], etc. The main components of service mesh are a Data plane - which is a lightweight proxy as “side-car” and a Control plane - which controls configurations of the proxies based on performance metrics.

3.6 Continuous Integration (CI), Continuous Delivery, Continuous Deployment (CD)

CI is a new software engineering practice to speed up software delivery by decreasing integration times. Integrations are verified by an automated build and test [32]. Similarly, Continuous Delivery is an automated build with each check-in of the new code. And Continuous Deployment is a process of releasing changes from the repository to production. CI/CD pipeline is performed via automated tools like Ansible [21], Drone [4], Spinnaker [15], AWS CodePipeLine [2] and Otter [1], etc.

3.7 Complexity and Patterns

As shown in Fig. 3 and Fig. 2 each microservice has its own copy of the data and the events become a source of truth for the data aggregation. Individual microservices can accumulate and modify their own data, but in case of dependency (just like the referential integrity in RDBMS) there are several ways to handle it, and the industry is still evolving and experimenting with different things. The two most prominent patterns to address these issues are briefly described below.

3.7.1 Event Sourcing Pattern. One way to handle the data consistency in a microservices architecture via event sourcing. Event sourcing persists the state of each microservice data as a sequence of state-changing events. When the state of a microservice data changes, a new event is appended to the list of events. Thus by persisting events in an event store becomes a single source of truth for the enterprise.

3.7.2 Side-car Pattern. Side-car pattern name comes from the side-car with a motorcycle, with the same life cycle as the microservice, and is tasked to perform peripheral tasks, such as monitoring, logging and, data integrity checks. Airbnb’s SmartStack [20] and Netflix Prana [11] are some examples.

4 METHODS

To describe and capture the stakeholders’ viewpoint concerns, from microservices perspective, we take Zachman’s definition of stakeholders’ views as given below:

A) Scope (Ballpark View),  
B) Business Model (Owner’s View),  
C) System Model (Designer’s View),  
D) Technology Model (Builder’s View),  
E) Detailed Representations (Subcontractor’s View),

In the following, we define each view and the group of stakeholders who hold this view. Then we briefly describe stakeholders’ viewpoint concerns from each of w66 viewpoints of an enterprise for microservice-based developments. These descriptions constitute EA from a microservices perspective in the order given by Sultan and Miranskyy [51].

4.1 Scope (Ballpark View)

“Ballpark View” sets the scope and puts architecture effort in perspective, and is also called the ‘contextual’ view of the organization and describes the environment and surroundings the enterprise will function in and interact with. The following groups of stakeholders see the organization from this view: business development directors, delivery managers, CIOs, CFOs, and CSOs, etc. Below is the detailed description of their viewpoint concerns from each viewpoint perspective.

These stakeholders need to capture the contextual information of APIs and Microservices (to be provisioned by the organization).

who (people):
This stakeholders’ viewpoint concern captures the list of businesses an enterprise interacts with and may expose microservices to. The list of the organizations an enterprise might interchange information via APIs. This viewpoint concern is used in planning for the business to business (B2B) and business to consumer (B2C) relationships.

what (data):
This stakeholders’ viewpoint concern captures a list of things an enterprise needs to conduct business with other businesses and organizations - used in the design of microservices and may be required to offer APIs.

which (selection):
In this viewpoint concern, we need to capture and select the list of organizations important to the enterprise - the target audiences of microservices and APIs important to business or vice versa - select microservices and APIs to conduct business.

where (network):
This viewpoint concern captures locations where the enterprise conducts its business - the candidate locations for microservices. The list of partner locations is also of interest form this viewpoint for planning purposes.

how (function):
It is essential to define business processes required for microservices and APIs at the contextual level. This allows for
strategic planning and setting future goals at the highest level of the enterprise hierarchy.

why (motivation):
It becomes important to link the vision and business reasons for the provision of microservices and APIs at the contextual level. Without this linkage, the IT may provide services and offer APIs without business need.

when (time):
Only the enterprise owners and planners can think and provide guidance on the business cycles important to the business: the target cycle of microservices and APIs renewal, retirement, or up-gradation is captured in this viewpoint concern.

Capturing these viewpoint descriptions from the microservices perspective adds to the enterprise’s contextual view and enables planning for DevOps, re-user, vision and strategic direction.

4.2 Business Model (Owner’s View)

“Business Model” describes the owner’s view of the enterprise. What the enterprise does and why is captured in the business value propositions.

The following groups of stakeholders see the enterprise from each viewpoint perspective to see what API and Microservices will be provisioned by the organization, when, and why.

who (people):
This viewpoint concern identifies workflows required to expose microservices to internal or external audiences (organizations) via APIs

what (data):
This viewpoint concern describes the data elements (at semantic level) required for the design of microservices and APIs

which (selection):
This viewpoint concern deals with prioritization and, selection of the important business processes, business scenarios, business functions required to provide APIs and microservices.

where (network):
This viewpoint concern captures locations (geographic locations) where specific microservices may reside (for example, the data center location for latency or specific service offering), and APIs may be offered.

how (function):
This viewpoint concern captures and defines at the business function level, which business functions will be automated with what microservice. Similarly, this viewpoint concern also defines which business functions will be exposed via APIs.

Table 1 below captures this viewpoint concern in tabular form.

| Business Taxonomy | μ-service {1..n} | API {1..n} |
|-------------------|-----------------|-----------|
| Business Function | x               | x         |
| Business Function | x               |           |

why (motivation):
It is imperative to capture and define the motivation and reasons for the provision of essential business services which are the target for microservices and APIs. This viewpoint concern captures and documents these.

when (time):
This viewpoint concern captures business cycles for which a specific API or microservice is provided, modified, updated, or expired.

4.3 System Model (Designer’s View)

The systems model is the view of an enterprise from the system designer’s perspective (automation perspective). Where the business processes (candidates for automation) are described in microservice terms. At this stage, the designers (system engineers/architects) identify business concepts (entities) on which the microservices will work (use-cases, application components, etc.) or the entities used by microservices and APIs. API naming conventions and naming guidelines are documented. The CD and CI pipelines are decided, and tools are selected.

The following groups of stakeholders see the organization from this view: enterprise architects, requirements engineers, project managers, security architects, privacy specialists, internal and external regulators, auditors (internal and external), business continuity & disaster recovery planners, etc. Below is the detailed description of their viewpoint concerns from each viewpoint perspective.

who (people):
This viewpoint concern captures interface requirements for APIs and microservices. A Decision which microservices will be system-level microservices and which ones will be user-level microservice is decided in this viewpoint concern. Reverse proxy requirements and considerations from the access perspective may also be captured in this viewpoint concern. This can be captured in a table like a grid, as shown in Table 2.

Table 2: Interface requirements for μ-services and APIs

| Organization → | internal / external | B2B | B2C |
|----------------|---------------------|-----|-----|
| μ-service {1..n} | x                   |     |     |
| API {1..n}      | x                   |     |     |
| reverse proxy   | x                   |     |     |

what (data):
This viewpoint concern captures logical data elements to be used by each microservice and exposed by APIs. This is
the point the system designers may decide the microservice patterns ("event sourcing," "side-car" pattern, etc. [52]) to be used for persistent data, as shown in Table 3.

Table 3: Persistence requirements of μ-services

| Data elements | Pattern     | Persistence |
|---------------|-------------|-------------|
| μ-service\textsubscript{1..n}       | side-car    | x           |
| API\textsubscript{1..n}            | event-sourcing |             |

**which (selection):**

Selection and prioritization based on the business value is captured in this viewpoint concern - Which business processes, business scenarios, business functions and logical data elements are required for which APIs and microservices. Please note that which quantifies the selection as described in detail by [50].

**where (network):**

This viewpoint concern captures decisions for the provision of microservice at different network locations. The decisions to group services in different Kubernetes pods and clusters are captured in this viewpoint concern. This can be represented in tabular form as shown in Table 4 below. The API Gateway rules are also defined by this viewpoint concern, as shown in Table 5.

Table 4: μ-services deployment schemes and polices

| Deployment | POD\textsubscript{1..n} | Network\textsubscript{1..n} | Service Mesh |
|------------|------------------------|----------------------------|--------------|
| μ-service\textsubscript{1..n}     | yaml files             | yaml files                 |              |
| Container\textsubscript{1..n}    | yaml files             | x                          | yaml files   |

**how (function):**

This viewpoint concern captures business process models for provisioning APIs and microservices. As each microservice can be designed using a different technology stack, this can be captured as a grid, as shown in Table 6 below. Similarly, the design parameters - the output, the input and success criteria, and error methods along with the tools used to design the APIs can be captured in this viewpoint concern as shown in Table 7.

Table 6: μ-services technology stack

| Tech Stack | Node | Python | GO |
|------------|------|--------|----|
| μ-service 1 | x    | x      |    |
| μ-service 2 | x    | x      |    |
| μ-service n | x    | x      |    |

**why (motivation):**

This viewpoint concern captures business rules specific to microservices and APIs. And can be represented in tabular form, or a business rules engine can be used.

**when (time):**

This view captures business cycles on which a specific API or microservice is provided, modified, updated, or expired and can be captured in tabular form, see Table 8.

Table 8: μ-services and API life cycle

| Business Cycles | Provide | Delete | Update |
|-----------------|---------|--------|--------|
| μ-service\textsubscript{1..n} | business cycle | x | date |
| API\textsubscript{1..n} | date | time | x |

4.4 Technology Model (Builder’s View)

"Technology Model" is the description of microservices from a technology, data, and infrastructure perspective. The technology stack and the design decisions are captured in this viewpoint. These decisions play a major role in the CD and CI pipelines. For example, the configuration files (YAML files for containers and Kubernetes); the Software Development Kits (SDKs), and code samples are described in this viewpoint concern. The data transfer mechanisms between microservices (XML, JSON, Protocol Buffers known as gRPC [6], etc.) are also decided and captured in this viewpoint concern.

The following groups of stakeholders see the organization from this view: developers, programmers, designers, DevOps engineers, network engineers, SRE engineers, etc. Below is the detailed description of their viewpoint concerns from each viewpoint perspective.

**who (people):**

This viewpoint concern captures the detailed design decision of the APIs and microservices. The decision in which microservices will be system-level microservices and which ones will be user-level microservice is decided. Reverse proxy rules and considerations from the access perspective is also captured in this viewpoint concern.

**what (data):**

This viewpoint concern captures details of physical data elements to be used by each microservice and/or exposed by APIs. This is the point the builders provide details on how the data between the microservices will be interchanged (JSON, XML, or gRPC).

**which (selection):**

This viewpoint concern deals with the selections and prioritization related to physical data elements. Low-level design decisions for the design of microservices and APIs are captured in this viewpoint concern. For example, the choice of
containers available for a specific microservice and the priority given to a specific container (for its light-weightness or availability in the private repositories).

where (network):

This viewpoint concern captures the detailed design of network locations where microservices are deployed. This viewpoint concern captures system networking requirements, such as Kubernetes clusters and the container networking requirements and needs, as shown in Table 9.

Table 9: K8s clusters Networking

| POD 1...n | POD CNI | IP 1...n |
|-----------|---------|---------|
| µ-service1...n | IP | IPS |
| Container1...n | IP | IPS |

how (function):

This viewpoint concern captures system-level design decisions of the APIs. The technology stack for each microservice is captured and the design parameters of the APIs - the access, authorization, and authentication mechanisms, are captured in this viewpoint concern.

why (motivation):

This viewpoint concern captures business rules implementations for microservices and APIs.

when (time):

This viewpoint concern captures implementation details for the expiration and deletion of microservices and APIs at specific business cycles and events.

4.5 Detailed Representations (Consumer’s View)

This viewpoint describes how external consumers see the enterprise from outside. The stakeholders in this group include the business owners of external enterprises, the architects, and the developers community outside the enterprise. These stakeholders’ viewpoint concerns are how to conduct business with the enterprise and how the systems-level integration will work.

The US-based technology giants, also known as FANG (Facebook, Amazon, Netflix, Google) [56] offer SDKs and code examples that directly target this group of stakeholders. The terms like “developers experience” comes from providing this viewpoint concern. Most of the industry leaders in technology, provide detailed descriptions of their APIs and issue SDKs of their services. Their target audiences are the designers, architects, and developers in consuming organizations (these audiences, in turn, steer the partner organizations’ business leaders to buy services from the enterprise). In this viewpoint concern, the needs of external stakeholders are captured and defined. These views and viewpoint concerns of stakeholders, enable other organizations to consume what the enterprise offers.

This view focuses on the detailed representation of the organizations’ microservices and APIs - API documentation (Swagger, OpenAPI, GraphQL [5]); and the Microservices SDKs and code samples. These viewpoint concerns can be captured in the form given in Table 10.

| Consumers View | SDK | Code Samples | Documents |
|----------------|-----|--------------|-----------|
| µ-service1...n | details | details | details |
| API1...n       | details | details | details |

5 THE PROPOSED FRAMEWORK

Based on the technologies described in Section 3 and the descriptions given in Methods (Section 4), we organize viewpoint concerns in tabular form, as shown in Table 11. The columns in Table 11 represent the ordered viewpoints and the rows are stakeholders’ groups drawing from work by Sultan and Miranskyy [50, 51]. Each cell in Table 11 contains viewpoint concerns of a given stakeholders’ group from a specific viewpoint. The order and precedence of w6h viewpoints enables capturing holistic EA of the enterprise [51].

6 DISCUSSION

In this publication, we described the fundamental components of microservices-based cloud-native application developments. We presented a structured methodology to capture and define stakeholders’ viewpoint concerns from microservices and APIs perspective. We used well-established viewpoints to gather the full-some views of the entire enterprise. This structured approach enables capturing and defining each stakeholder’s needs. Once the full-some needs and viewpoint concerns are captured, these enable engineering and analysis for strategic planning, alignment, reusability, and agility.

It is well established in the industry that there is a widening gap between EA and DevOps teams. The methodology we provided in this publication fills this gap by enabling DevOps teams to understand the stakeholders’ viewpoint concerns. At the same time, this enables enterprise architects to capture and define microservices-based DevOps developments in enterprise architecture constructs.

The methods described in Section 4 can be used to create a full blueprint of the entire enterprise from stakeholders’ viewpoint concerns perspective for microservices-based developments. This framework establishes a mind-map to start the architectural effort for microservice-based developments. The order of viewpoints enables asking and capturing the viewpoint concerns in the right order without missing necessary and required information. With the advent of microservices-based DevOps developments, organizations are facing another challenge, known as BiModel IT3. Most legacy applications may fulfill their functional requirements; these are not scalable or interoperable with the modern microservices-based architectures. The proposed framework enables dealing with these challenges by providing tools (the ordered stakeholders’ viewpoints concerns) to bridge the gap between these two inherently disjoint worlds. In summary the proposed framework can help organizations:

- Scale microservices based on stakeholders’ needs;
- Reduce complexities of microservices based developments;
- Link stakeholders’ needs for increased sponsorship;
- Perform analysis for strategic planning and future growth;

3Gartner defines BiModal IT as “the practice of managing two separate, coherent modes of IT delivery, one is focused on stability and the other on agility’’
Table 11: The proposed framework: Rows are the views - how stakeholders view an organization, and the square brackets list key stakeholders who hold this view. Columns are stakeholders’ viewpoints (in the order proposed by [50, 51]). The cells are high-level stakeholders’ viewpoint concerns from a specific viewpoint, based on discussions in Sections 3 and 4.

| Microservices (µ-services) | (1) People (who) | (2) Data (what) | (3) Selection (which) | (4) Network (where) | 2,3/2,4 ⇒ (5) Function (How) | 2,5 ⇒ (6) Motivation (Why) | 4,5 ⇒ (7) Time (when) |
|---------------------------|-----------------|----------------|----------------------|--------------------|-----------------------------|-----------------------------|------------------------|
| Scope (Ballpark View):    | list of business the enterprise may develop µ-services for, and expose or consume APIs from | List of things required for each µ-service and/or API | Which: prioritized list of organizations, µ-services and APIs each candidate for reuse | List of locations for hosting µ-services and related technologies | List of business processes for design each µ-service | List of business goals/strategies for each µ-service and API | List of cycles significant for µ-service and APIs |
| Business Model (Owners’ View): | e.g., ű-services workflow models | e.g., ű-services semantic models | e.g., Which business process, data elements are required for ű-services and APIs | e.g., Datacenter locations where specific ű-service reside | e.g., Business taxonomy and linkage between business processes with µ-services and APIs, as shown in Table 1 | e.g., Business goals related to ű-services and APIs | e.g., Business cycles when an API or ű-service is provided, modified, updated, or expired. |
| System Model (Designer’s View): | e.g., Interface requirements for APIs and ű-services - Table 2 | e.g., Logical data elements to be used by each ű-service and exposed by APIs - Table 3 | e.g., Selection and prioritization of business processes/logical data elements required by API and ű-services | e.g., Distributed ű-service and container architecture - Table 4, Gateway and reverse proxy rules - Table 5 | e.g., ű-service technology stack and API design mechanisms - Table 6 & Table 7 | e.g., ű-service business rules | e.g., Automated deletion/provisioning of ű-services at pre-specified time |
| Technology Model (Builder’s View): | e.g., ű-services architecture (system vs user level ű-services) and patterns | e.g., ű-service physical data elements; data interface specifications (JSON, XML or gPRC) | e.g., Selection and prioritization of low-level design components, like containers, and pods etc. | e.g., ű-service and container deployment locations, locations of Kubernetes clusters and configuration files (.yaml) | e.g., ű-service technology stack, API access, authentication and authorization mechanisms | e.g., Business rules implementations for ű-services and APIs | e.g., Zero-touch deployment scripts for auto provision/deletions of ű-services |
| Detailed Representations (Consumer’s View): | SSDKs, APIs and Code Samples; See Table 10 | e.g., (curl -v -X GET https://api.sandbox.paypal.com/v1/payment-experience/web -profiles/XP-8YTH-NNP3-WSVN-3C76 -H "Content-Type: application/json" -H "Authorization: Bearer Access-Token"); |

- Promote and increase re-use of microservices;
- Bridge the gap between DevOps and EA teams;
- Create EA for microservices-based developments.

Disclaimer: The opinions expressed in this paper are those of the authors and not necessarily of the Government of Ontario.

7 FUTURE RESEARCH

We intend to use graph theory to model and cluster stakeholders’ viewpoint concerns and microservices. For example, to retire weak microservices, which does not carry much business value, we may cluster a weighted graph of stakeholder’s viewpoint concerns and
work with data collected from their enterprises. These findings will enable novel work in stakeholders’ viewpoints and microservice-based developments.

REFERENCES

[1] 2020. Accelerate DevOps on Windows. (sep 2020). Retrieved September 2, 2020 from https://msdn.com/

[2] 2020. AWS Codepipeline. (sep 2020). Retrieved September 2, 2020 from https://aws.amazon.com/codepipeline/

[3] 2020. Conduit. (sep 2020). Retrieved September 2, 2020 from https://conduit.io/

[4] 2020. Drone CI LÅŚ Automatic Software Testing and Delivery. (sep 2020). Retrieved September 2, 2020 from https://drone.io/

[5] 2020. GraphQL. (sep 2020). Retrieved September 2, 2020 from https://graphql.org/

[6] 2020. gRPC. (sep 2020). Retrieved September 2, 2020 from https://grpc.io/

[7] 2020. Homepage. (sep 2020). Retrieved September 2, 2020 from https://linkerd.io/

[8] 2020. Istio. (sep 2020). Retrieved September 2, 2020 from https://istio.io/

[9] 2020. Mesos. (sep 2020). Retrieved September 2, 2020 from http://mesos.apache.org/

[10] 2020. Netflix/Prana. (sep 2020). Retrieved September 2, 2020 from https://github.com/Netflix/Prana

[11] 2020. OpenAPI. (sep 2020). Retrieved September 2, 2020 from https://raml.org/

[12] 2020. Production-Grade Container Orchestration. (sep 2020). Retrieved September 2, 2020 from https://kubernetes.io/

[13] 2020. RAML. (sep 2020). Retrieved September 2, 2020 from https://raml.org/

[14] 2020. SmartStack: Service Discovery in the Cloud. (Dec 2016). https://medium.com/airbnb-engineering/smartstack-service-discovery-in-the-cloud-4b8a008de19

[15] 2020. Spanmaker. (sep 2020). Retrieved September 2, 2020 from https://spanmaker.io

[16] 2020. Swagger. (sep 2020). Retrieved September 2, 2020 from https://swagger.io

[17] 2020. Swarm mode overview. (Aug 2020). https://docs.docker.com/engine/swarm/Online; accessed 1 September 2020.

[18] 2020. Workload Orchestration Made Easy. (sep 2020). Retrieved September 2, 2020 from https://www.nomadproject.io/

[19] Carlos M Aderaldo, Nabor C Mendonça, Claus Pahl, and Pooyan Jamshidi. 2017. Benchmark requirements for microservices architecture research. In 2017 IEEE/ACM 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering (ECASE). IEEE, 8–13.

[20] AirbnbEng. 2016. SmartStack: Service Discovery in the Cloud. (Dec 2016). https://medium.com/airbnb-engineering/smartstack-service-discovery-in-the-cloud-4b8a008de19

[21] 2020. Istio. (sep 2020). Retrieved September 2, 2020 from https://istio.io/

[22] 2020. Production-Grade Container Orchestration. (sep 2020). Retrieved September 2, 2020 from https://kubernetes.io/

[23] 2020. RAML. (sep 2020). Retrieved September 2, 2020 from https://raml.org/

[24] 2020. Spanmaker. (sep 2020). Retrieved September 2, 2020 from https://spanmaker.io

[25] 2020. Swagger. (sep 2020). Retrieved September 2, 2020 from https://swagger.io

[26] 2020. Swarm mode overview. (Aug 2020). https://docs.docker.com/engine/swarm/Online; accessed 1 September 2020.

[27] 2020. Workload Orchestration Made Easy. (sep 2020). Retrieved September 2, 2020 from https://www.nomadproject.io/

[28] 2020. Production-Grade Container Orchestration. (sep 2020). Retrieved September 2, 2020 from https://kubernetes.io/

[29] 2020. Production-Grade Container Orchestration. (sep 2020). Retrieved September 2, 2020 from https://kubernetes.io/

[30] 2020. Production-Grade Container Orchestration. (sep 2020). Retrieved September 2, 2020 from https://kubernetes.io/

[31] 2020. Production-Grade Container Orchestration. (sep 2020). Retrieved September 2, 2020 from https://kubernetes.io/

[32] Martin Fowler and Matthew Formmel. 2006. Continuous integration. (2006).

[33] Martin Garriga. 2017. Towards a taxonomy of microservices architectures. In International Conference on Software Engineering and Formal Methods. Springer, 203–218.

[34] Martin Kleehaus and Florian Matthias. 2019. Challenges in Documenting Microservice-Based IT Landscape: A Survey from an Enterprise Architecture Management Perspective. In 2019 IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC) IEEE, 11–20.

[35] Jean-Pierre Koenig. 2004. Any questions left? Review of Ginzburg & Sag’s Investigative investigations. Journal of linguistics 40, 01 (2004), 131–148.

[36] Nane Kratzke and René Peinl. 2016. Clouns-a cloud-native application reference model for enterprise architects. In 2016 IEEE 20th International Enterprise Distributed Object Computing Workshop (EDOCW) IEEE, 1–10.

[37] Nane Kratzke and René Peinl. 2016. Clouns-a cloud-native application reference model for enterprise architects. In 2016 IEEE 20th International Enterprise Distributed Object Computing Workshop (EDOCW) IEEE, 1–10.

[38] Nanne Lamgen and Sun Client Solutions. 2005. Solrini containers/What they are and how to use them. Sun BluePirns OnLine (2005), 819–2679.

[39] Marco Tulio Valente. 2016. Towards a technique for extracting microservices from monolithic enterprise systems. arXiv preprint arXiv:1605.03175 (2016).

[40] Michael Cysouw. 2004. Interrogative words: an exercise in lexical typology. In António Ferreira Pereira and Pedro Sousa. 2004. A method to define an Enterprise Architecture using the Zachman Framework. In Proceedings of the 2004 ACM symposium on Applied computing. ACM, 1366–1371.

[41] Robert Winter and Ronny Fischer. 2006. Essential layers, artifacts, and dependencies of enterprise architecture. In Proceedings of the 33rd Annual ACM Symposium on Applied Computing. ACM, 640–647.

[42] Michael Cysouw. 2004. Interrogative words: an exercise in lexical typology. In António Ferreira Pereira and Pedro Sousa. 2004. A method to define an Enterprise Architecture using the Zachman Framework. In Proceedings of the 2004 ACM symposium on Applied computing. ACM, 1366–1371.

[43] Robert Winter and Ronny Fischer. 2006. Essential layers, artifacts, and dependencies of enterprise architecture. In Proceedings of the 33rd Annual ACM Symposium on Applied Computing. ACM, 640–647.

[44] Robert Winter and Ronny Fischer. 2006. Essential layers, artifacts, and dependencies of enterprise architecture. In Proceedings of the 33rd Annual ACM Symposium on Applied Computing. ACM, 640–647.

[45] Robert Winter and Ronny Fischer. 2006. Essential layers, artifacts, and dependencies of enterprise architecture. In Proceedings of the 33rd Annual ACM Symposium on Applied Computing. ACM, 640–647.

[46] Robert Winter and Ronny Fischer. 2006. Essential layers, artifacts, and dependencies of enterprise architecture. In Proceedings of the 33rd Annual ACM Symposium on Applied Computing. ACM, 640–647.

[47] Robert Winter and Ronny Fischer. 2006. Essential layers, artifacts, and dependencies of enterprise architecture. In Proceedings of the 33rd Annual ACM Symposium on Applied Computing. ACM, 640–647.

[48] Robert Winter and Ronny Fischer. 2006. Essential layers, artifacts, and dependencies of enterprise architecture. In Proceedings of the 33rd Annual ACM Symposium on Applied Computing. ACM, 640–647.

[49] Robert Winter and Ronny Fischer. 2006. Essential layers, artifacts, and dependencies of enterprise architecture. In Proceedings of the 33rd Annual ACM Symposium on Applied Computing. ACM, 640–647.

[50] Robert Winter and Ronny Fischer. 2006. Essential layers, artifacts, and dependencies of enterprise architecture. In Proceedings of the 33rd Annual ACM Symposium on Applied Computing. ACM, 640–647.

[51] Robert Winter and Ronny Fischer. 2006. Essential layers, artifacts, and dependencies of enterprise architecture. In Proceedings of the 33rd Annual ACM Symposium on Applied Computing. ACM, 640–647.

[52] Robert Winter and Ronny Fischer. 2006. Essential layers, artifacts, and dependencies of enterprise architecture. In Proceedings of the 33rd Annual ACM Symposium on Applied Computing. ACM, 640–647.

[53] Robert Winter and Ronny Fischer. 2006. Essential layers, artifacts, and dependencies of enterprise architecture. In Proceedings of the 33rd Annual ACM Symposium on Applied Computing. ACM, 640–647.

[54] Robert Winter and Ronny Fischer. 2006. Essential layers, artifacts, and dependencies of enterprise architecture. In Proceedings of the 33rd Annual ACM Symposium on Applied Computing. ACM, 640–647.

[55] Robert Winter and Ronny Fischer. 2006. Essential layers, artifacts, and dependencies of enterprise architecture. In Proceedings of the 33rd Annual ACM Symposium on Applied Computing. ACM, 640–647.

[56] Robert Winter and Ronny Fischer. 2006. Essential layers, artifacts, and dependencies of enterprise architecture. In Proceedings of the 33rd Annual ACM Symposium on Applied Computing. ACM, 640–647.

[57] Robert Winter and Ronny Fischer. 2006. Essential layers, artifacts, and dependencies of enterprise architecture. In Proceedings of the 33rd Annual ACM Symposium on Applied Computing. ACM, 640–647.