Towards a Resource Slice Interoperability Hub for IoT

Hong-Linh Truong
Faculty of Informatics, TU Wien, Austria
E-mail: hong-linh.truong@tuwien.ac.at

***Abstract*—Interoperability for IoT is a challenging problem because it requires us to tackle (i) cross-system interoperability issues at the IoT platform sides as well as relevant network functions and clouds in the edge systems and data centers and (ii) cross-layer interoperability, e.g., w.r.t. data formats, communication protocols, data delivery mechanisms, and performance. However, existing solutions are quite static w.r.t software deployment and provisioning for interoperability. Many middleware, services and platforms have been built and deployed as interoperability bridges but they are not dynamically provisioned and reconfigured for interoperability at runtime. Furthermore, they are often not considered together with other services as a whole in application-specific contexts. In this paper, we focus on dynamic aspects by introducing the concept of Resource Slice Interoperability Hub (rsiHub). Our approach leverages existing software artifacts and services for interoperability to create and provision dynamic resource slices, including IoT, network functions and clouds, for addressing application-specific interoperability requirements. We will present our key concepts, architectures and examples toward the realization of rsiHub.**

I. Introduction

The increasing, complex IoT applications built from various functionality of IoT, network functions and clouds have created many calls for IoT interoperability. A recent special issue [1] has presented many aspects of and solutions for IoT interoperability. Several works and standardization organizations have started to look at specific IoT interoperability issues. However, most of IoT interoperability solutions are static compared with dynamic changes of underlying systems: IoT interoperability issues are solved in advance by introducing software that are pre-deployed for specific uses. Such software might solve certain interoperability issues but we lack techniques to provision and configure such software for application-specific needs as well as to combine such software with other solutions to provide on-demand interoperability bridges. This can be observed through the use of pre-deployed protocol bridges [2], semantic data conversion hub[3], sensor data syntax and metadata [4], to name just a few.

We look at another perspective: can we solve IoT interoperability at runtime when a client needs a set of IoT resources, network function resources and cloud resources but these resources are not interoperable w.r.t. the client's requirements due to certain reasons. Here the notion of resources in our work is not just about infrastructural ones, like virtual machines with CPU and memory. Resources can be data, firewall, cloud virtual machines, cloud database, streaming processing service, etc. Resources are provided or can be provisioned by different providers and are exposed through services; resources can be services with well-defined interfaces (e.g., based on REST or MQTT). Therefore, with such capabilities resources can be acquired and controlled for on-demand interoperability solutions. Fortunately, such capabilities are the norm nowadays in IoT, edge and cloud systems. In our view, applications (clients), which need IoT resources, mostly data produced by IoT devices and control of such devices, will require various types of resources: in addition to the IoT resources, network functions resources at the edge and cloud resources are also required. Therefore, in the view of such applications, a set of IoT resources, network function resources and cloud resources should be provisioned and managed for the application requirements. In our work, a resource slice is a set of resources established for a specific application in a specific context.

The key issue is how to make such a set of resources interoperable, e.g., for delivering IoT data for the application based on its specific context. This will involve very complex tasks as our approach for IoT interoperability is dynamic: we do not assume that all resources and providers have known existing services for enabling interoperable protocols, data format, etc. At runtime, we expect the interoperability tool to augment the client's resources with new resources to deal with interoperability problems. Previously, we have developed basic services for harmonizing IoT, network functions, and cloud resources [5]. We can interface to various other services to receive information about resources (e.g., Google machines, networks and data service). We can also connect to repositories to download artifacts and provision corresponding resources based on that artifacts, e.g. pull an image from a Docker repository. Leveraging these facilities, this paper proposes a concept of resource slices for IoT interoperability. We introduce Resource Slice Interoperability Hub (rsiHub) which includes techniques and services for gathering metadata about resources and for dealing within on-demand provisioning of slices and interoperability bridges. This paper presents key concepts for rsiHub and examples, paving the way for the realization of rsiHub in the future.

The rest of this paper is organized as follows. Related work is presented in Section II. Section III presents scenarios, models, and rsiHub and its concepts. In Section IV we present core issues of the rsiHub. We illustrate examples in Section V. We conclude the paper in Section VI.

II. Related Work

The key difference between our work and existing works for IoT interoperability is the dynamic, on-demand interoperability support. Existing works mainly focus on static protocol and data translation solutions to achieve IoT interoperability . For such solutions, usually interoperability issues are identified in advance and corresponding solutions are developed and deployed. Thus these solutions are hard to reconfigured and they are used in specific deployment. Furthermore, they rarely leverage advanced techniques

The special issue [1] presents various interoperability solutions. All of them can be possibly used in our work as we can discover existing solutions through metadata and determine if they are suitable for runtime interoperability. In [3] the authors present a hub-based approach for IoT Interoperability. This Hub includes various data conversion features. However it is not about dynamic solution of using resource slices like our work. We could assume that this Hub is also one service provider that we can use to deal with the interoperability. In [6] the authors show architectures and use cases for platform interoperability. Their approach is to assume that IoT devices will connect to the right interoperable platforms and services, which are pre-deployed, to ensure interoperability. This is different from our approach in which we select resources and deploy and control resources to support interoperability. In [2] a detailed analysis and proposal for protocols translator has been given. Our work is not focused on providing such protocol translator but collecting information about them and manage them(including artifact) so that when we have a problem, we provision such translator. In this view, we are the user of such translators.

III. Resource Slice Interoperability Hub

A. Running Motivating Scenarios

Scenario 1 – IoT Camera Video Analytics: Consider a city like Da Nang, Vietnam. We have many public and private cameras providing real-time video data. These cameras are exposed through `IoTCameraService` by providers (public organizations or individuals). The video camera as well as its individual video files are resources in our concept. Consider two specific clients need to access camera data for two hours for all possible cameras close to a place (e.g., due to a football event or an accident). Practically, these camera resources are diverse, e.g., w.r.t quality of data and transmission protocols. Furthermore, the clients need data a different ways: $client_1$ needs all data pushing into a storage (e.g., Google) and another component will trigger video analytics using serverless functions listening changes in the storage. $client_2$ needs to obtain data directly from a broker (e.g., `Kafka`[1]) and push video, using `Trigger`, into instances of its Python video analytic application `OpenCVVA` running atop, e.g., docker containers. Since such clients require and process video on-demand with different requirements, we might face several interoperability issues at runtime:

- `IoTCameraService` and cameras might change the data delivery mechanisms, e.g., from pushing video to clients into pulling data from clients.
- Quality of data: near-realtime video records might vary from 1 minute to 5 minutes.
- Clients might need as many as possible videos in a very fast time, instead of latest video from all camera, due to the criticality of the mission.
- Network firewalls and storage might need to be changed due to the situation that the client has to serve.

Many of these problems cannot be solved in advance but at runtime by requiring extra, suitable resources to be provisioned and/or existing resources to be reconfigured.

Scenario 2 – IoT for seaport operations: The above-mentioned scenario deals with the same type of data (video), although data might have different formats, quality, etc. Another scenario is for the operation of the seaport where we have various IoT providers, e.g., vessels in shipping, cranes in port terminals, emergency services, and hauliers, as well as potential edge computing system providers (providing services for network functions and edge analytics, based on edge computing/fog computing models), and cloud services[2]. Each provider might use different platforms, especially for those providing data through IoT technologies. For this scenario, an example of clients which need resource slices could be: an emergency response client requires IoT resources for status of cranes, vessels and hauliers, cloud/edge resources for video analytics of obstacles, traffics and presence, IoT resources for control cranes, and network resources for ensuring network performance and reliability during a specific context (e.g., an accident in the port).

B. Models

Let $RS = \{r_i\}$ be a resource slice. We use $RS(c)$ to denote a resource slice specified/needed by a client c, whereas we use $RS_i(c)$ to denote the interoperable resource slice based on $RS(c)$; $RS(c) \subset RS_i(c)$. For interoperability purpose, a resource is represented by $r(DP, CP, MT)$ where $DP = \{dp_i\}$ is a set of data points dp, $CP = \{cp_i\}$ is a set of control points cp that are associated with r, and $MT = \{mt\}$ is a set of metadata. For example, a camera is a resource –r_c, which can offer $DP(r_c) = \{dp_a, dp_c\}$ where dp_a will return all possible videos and dp_c will return the current video (e.g., the latest, near-realtime video). r_c can has cp_p which is used to control the camera (resource) to put a video to a storage (also a resource). Given a management service S (also a resource), S can provide a set of resources $R = \{r\}$ that allows clients to use $r \in R$ by invoking corresponding $DP(r)$ and $CP(r)$. Each $mt \in MT$ is represented by $(name, value)$. We use metadata to capture various information.

We use the above-mentioned notions for all types of resources. For example, network functions and cloud services have control points and data points, although network function

[1] http://kafka.apache.org/

[2] This scenario is from the INTER-IoT project – http://www.inter-iot-project.eu/

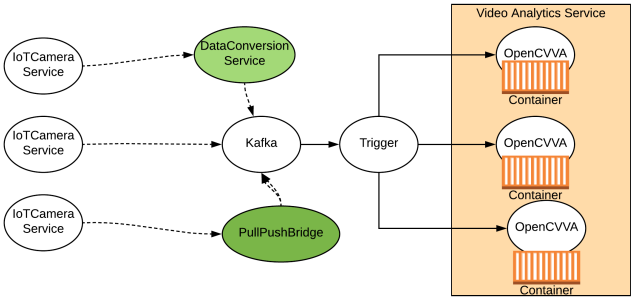

Fig. 1. Typical resources in client slice and new resources (`DataConversionService` and `PullPushBridge`) for interoperability

services might not support various data points like IoT or cloud resources. A client c might require, e.g., dp_c from all possible IoT resources r_c, e.g., all latest video from all cameras managed by `IoTCameraService` to be available in Google Storage. In this case, we need to control the service S to send data to the storage by calling cp_p for all r_c or to pull the video from r_c by using dp_c. Here, for c the slice consists of only r_c, Google Storage and serverless function-as-a-service (FaaS): $RS(c) = \{r_c i, GoogleStorage, FaaS\}$. Another client might have a slice as $RS(c) = \{r_c i, Kafka, Trigger, Container$ in which $r_c i$, $Kafka$, $Container$ are provided by existing providers and $Trigger$ is deployed on-demand.

Since resource slices for an application can be very complex, we are not focusing on automatically building resource slice structures of $RS(c)$. We assume that the structure of $RS(c)$ exists, e.g., provided by the client. Furthermore, we will focus only certain forms of the slice as these forms are typical in IoT applications w.r.t IoT data analytics and controls:

- many-to-one: this is typically for the integration of may data points to a single place (e.g., data broker or client).
- one-to-one: typical one-to-one interaction for data points and control points (e.g., to convert data from one form to another form or to perform a protocol translation)
- one-to-many: it is typical for controlling (e.g., the client controls many resources) and data dissemination (e.g., from the broker to different processing resources).

An $RS(c)$ can have multiple of such forms. In a typical situation, we can ask service providers to provision $RS(c)$. However, we can have different interoperability problems based on analyzing MT_i of $r_i \in RS(c)$, e.g., the data transfer protocol is not suitable. Thus, interoperability bridges – (composite) software artifacts or resources – need to be determined to solve such problems. Assume that we can find an interoperability bridge element, $ibe \in R$ or ibe from software artifacts that can solve an interoperability problem. Then we can augment $RS(c)$ with $IBE(c) = \{ibe_i\}$ to create $RS_i(c)$. Using runtime provisioning and configuration, we can provision $RS_i(c)$. Figure 1 shows an example of a typical slice and a slice with augmented resources for interoperability. Obviously, it does not easy to find out $IBE(c)$ and configure $IBE(c)$ and $RS(c)$ working together but it is a different issue not in this paper.

C. Integration Requirements

For the work in creating resource slices for interoperability, we have identified basic requirements for us to obtain enough information about possible artifacts and resource providers:

1) Artifacts for interoperability:: Often many types of software artifacts are available for runtime interoperability, for example, data conversion libraries. Such artifacts might be available only for IoT interoperability purposes as well as for general purposes. Furthermore, these artifacts can be deployed as services but due to many reasons they are not deployed at the time the client needs the slice.

2) Instances of resource provider:: Providers are suitable for IoT Interoperability in our concepts; they are IoT, cloud and network function services. They might be also services dedicated for IoT interoperability. For such providers, first, we need `ResourcePublishAPI` for querying all possible resources. Second, we need `ResourceControlAPI` for controlling resources, such as provisioning, configuration, etc. of resources. For example, given an IoTService hosted in `localhost`, `/camera/list` can be used to list all cameras, `/camera//current/now` will be the data point for current video for the resources, whereas `/camera/list` is for all possible camera resources and `camera/:cameraName` will be the API for pushing the video to Google Storage. Given these two sets of APIs, we can query and control resources using data points and control points.

D. Resource Slice Interoperability Hub Architecture

We introduce the conceptual architecture named Resource Slice Interoperability Hub (rsiHub), which includes services, software artifacts and algorithms to ensure resource slice interoperability. Figure 2 describes the current design of rsiHub. Main services are:

- *Local Management Service* is to interface to IoT, network function and cloud providers.
- *Global Management Service* is for the application and other middleware to control IoT devices, networks and services and acquire IoT data.
- *Interoperability Recommendation Service* is used for checking and finding bridges for interoperability and presenting workflows of configuration and provisioning of resources for interoperability.
- *Resource Slice Management Service* is introduced to support slice management by provisioning and configuring slices.

The Local and Global Management Services can be leveraged from HINC [5] by extending information models and query/-control protocols. In the next section, we describe key ideas for recommendation and provisioning on-demand interoperability.

IV. Runtime IoT Interoperability

A. Enriching Client's Resource Slice for Interoperability

1) Interoperability analytics: Interoperability Analytics component will take the metadata of resources and determine

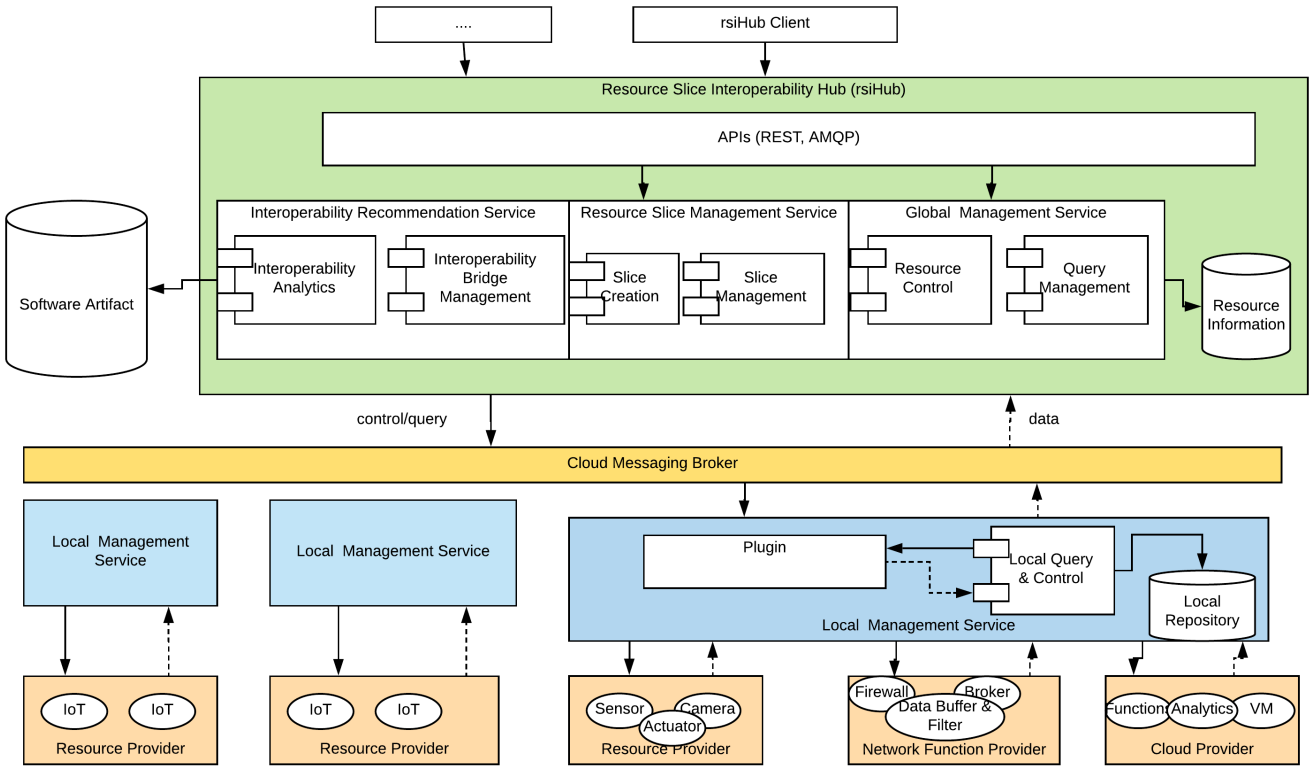

Fig. 2. Overview of the Resource Slice Interoperability Hub

interoperability issues. Examples of issues could be: (i) existing an IoT resource whose data format differs from the one required by the client, (ii) disparate data delivery frequencies between IoT resources, or (iii) incompatible data transferring protocols among resources. In supporting interoperability analytics and suggestions, we will rely on resource metadata and existing software artifact metadata and a catalog of possible situations that can be checked for interoperability. Information in the catalog can be simple, e.g., the information about software/resource that can be used to map one data source to another data. In the catalog, complex information about solutions will be presented by (i) a set of provision tasks to provide missing resources based on existing software artifacts and (ii) requests of new sources from existing providers, and (iii) reconfiguration tasks of existing resources. For this, we will obtain metadata from existing resources as well as from providers and from *Software Artifact Repository*. Here it is important to note that software artifacts should be annotated with metadata indented for supporting interoperability. We define an Interoperability Bridge Pipeline (IBP) as a composition of resources which are not required by the client in its resource slice, but are needed to deal with interoperability problems. For a client's resource slice $RS(c)$, we have a set of $ibp \in IBP$, each ibp will be established by provisioning suitable software artifacts and/or resources from existing providers; each ibp includes a set of interoperability bridge elements ibe.

2) Metadata: For interoperability, we consider various types of metadata, not just about syntax and semantics of IoT data for using data conversion and alignment services. The following types of metadata are important for interoperability:
Data quality: examples of data quality are accuracy, timeliness, up-to-dateness [7]. Such metadata characterize the data that IoT devices/services provide to the client. We need to consider them, as two sources of data might not be interoperable if their quality is not compatible. In this view, we apply data quality check for a set of data points. Given a set of $dp_i \in DP$ that have the same types of data, e.g. video, and dp_i connect to the same type of receivers (e.g. brokers or client applications), we will check the quality of data from dp_i.
Data delivery frequency: this is related to how data would be delivered. For example, some IoT temperature sensors might deliver data within 1 minute, whereas others would be within 5 minutes. They might create some interoperability problems when a client receives data from IoT sensors of different frequencies. Given a set of data points dp_i of a resource and connected to the same place, e.g. the message broker, we will check whether their frequencies are compatible.
Data compliance regulations: certain resources will deliver data based on some regulations (such as, data has to be within Europe). In our scenario, due to the situation, resources in Google may be used for analytics, thus network functions for firewall should allow sending data to Google. Otherwise, normally, it would be no image of humans will be sent (creating slice should not violate, e.g., by using US-based cloud services in the resource slice).
Data transfer protocols: given a set of dp_i and possible

software artifacts and existing resources for data transfers, there is a protocol mismatch. Given certain data formats and protocols, it is possible to deploy components to bridge different protocols. This might invoke not only on-demand services for communication, but also data transformation. Note that in our generic work, protocols might be low-level, such as MQTT, but also high-level, such as pull or push of video. Thus important information about protocols should be captured.

In our rsiHub, the above-mentioned types of metadata can be described either in metadata associated with resources or in specific data contracts.

3) Interactions: a slice consists of IoT resources, network function resources and cloud resources. Such resources are managed by individual providers and made available for the client by rsiHub. *Interoperability Recommendation Service* checks interoperability and suggests possible Interoperability Bridge Pipelines (IBPs). Figure 3 describes steps in supporting the construction of IBPs:

- *Data Transformation*: such as if the client requests the data delivery in JSON, but the data is in CSV. In this case we can utilize data pipeline techniques to transform data. For implementation, various existing services as well as tools, such as Logstash[3], can be used. This task will be done at the slice creation.
- *Protocol Translation for Data Delivery*: there are different delivery protocols and we need some protocol bridges. This can be achieved by using multi-protocol data delivery platforms but might need middleware to bridge different protocols. For example, an IoT resource provides data through MQTT, but the client needs data through REST. In this case, we can deploy an MQTT broker and ingest components (e.g., an ingestion service takes data from MQTT to Google BigQuery[4]). We will perform this task at the same time of slice creation.
- *Data Quality*: different IoT data sources have incompatible data quality. To make them interoperable, certain tasks might need to be invoked, such as removing bad data. This task might be done at runtime, by leveraging complex IoT data monitoring.

The above-mentioned tasks are just a few selected ones. Other tasks could be involved into guaranteeing reliability of networks using network functions or complex event processing to filter bad data.

B. Provisioning and Configuration

We distinguish two situations of provisioning and configuration of resource slices. First, there exist software artifacts for performing the interoperability but the resource has not been deployed and no service for that. For example, we have an ingest client that can take data images from `Kafka` and move the images into Google Storage but this ingest client has not been started, while `Kafka` is already available. Second, existing services have capabilities to support interoperability but their resources have not yet provisioned or need to be reconfigured. For example, a middleware has a plug-in for performing protocol translation, like in [2], but the plug-in has not provisioned because it has no request.

1) Step 1 – Provisioning: Provisioning is carried out when resources for ibp is not available from existing providers. In this case, $rsiHub$ will perform the deployment for resources of ibp by utilizing available infrastructures. This requires $rsiHub$ to interface to existing cloud providers, edge providers and network function infrastructure providers. In our prototype, these providers have REST APIs for us to provision containers and VMs. The provisioning will be two steps: first provisioning the infrastructural resources and second provisioning the software services resources.

2) Step 2 – Orchestrating services: When a r is required by a client, the corresponding provider of r will deploy r for the client. Through data points and control points we can request the provider to reconfigure r or we can directly reconfigure r.

Currently, for configuration and deployment, we rely on state-of-the-art deployments using Docker with Kubernetes and Docker Swarm . We are using popular tools like Chef and Ansible for configuration. Furthermore, to orchestrate control of multiple resources, we use Airflow[5].

V. Towards Implementation and Examples

We are currently implementing $rsiHub$. In this section we give some examples towards the realization of our ideas. We build our examples by developing and utilizing various IoT, network functions and cloud providers[6].

A. BTS Slice

We consider the Base Transceiver Stations (BTS) analytics in which a client wants to receive certain types of alarms from BTS and analyze. In our current work, a simple resource slice will include data points of (virtual) BTS sensors, MQTT brokers, ingest clients, and cloud database. Let us consider that the BTS monitoring is outsourced for different maintainers. It means their data formats are different. Furthermore, different vendors might use different data serialization mechanisms to save bandwidths. Figure 4(a) describes an example of possible slices in normal cases. However, as we have different types of IoT data formats (JSON, CSV and binary data), in order to support interoperability at runtime, we will need to add (i) `DeseralizedBinDataIngestClient` – a runtime deserialization library provided by corresponding providers, (ii) `JSONDataIngestClient` – a data pipeline conversion using Logstash, leading to the new interoperable resource slice shown in Fig 4(b)[7]. Such new sources can be determined by checking metadata of data points of BTS sensors.

[3]https://www.elastic.co/products/logstash

[4]https://cloud.google.com/bigquery/

[5]https://airflow.apache.org/

[6]The basic components – IoT Units – and services – utilizing units – are available at https://github.com/rdsea/IoTCloudSamples

[7]We have sample of BTS normal slice under https://github.com/rdsea/IoTCloudSamples/tree/master/examples/simpleBTS. However, the example of enhanced resource slice is currently being developed.

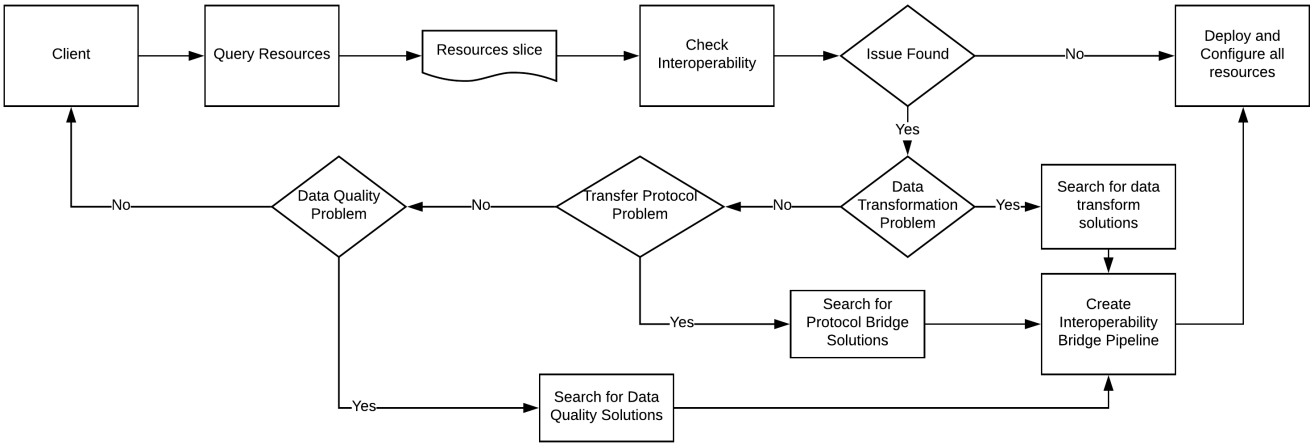

Fig. 3. Interactions in solving interoperability

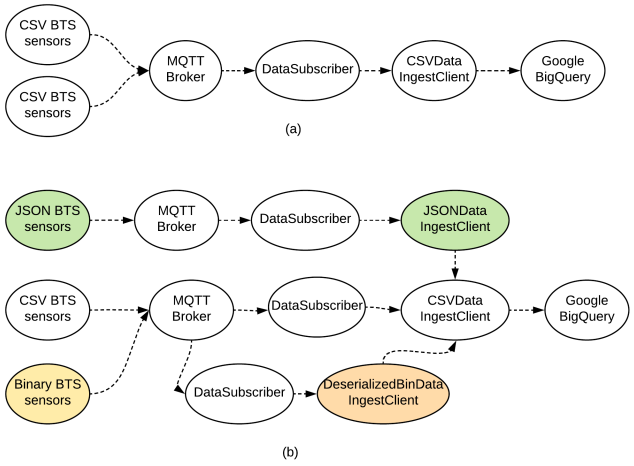

Fig. 4. BTS example: (a)with typical resource slice, (b)interoperable resource slice due to different data formats

B. Video Camera Slice

This example, we consider resource slices for video processing in Scenario 1. Given a situation, e.g., fire, various clients e.g., the traffic management, nearby hospital, and police, want different resource slices. Figure 5 gives an example of a resource slice $RS(c)$ with also interoperability bridge elements. Typically, we have IoT providers with cameras that can push data into `Kakfa` and `Trigger` can subscribe data from `Kafka` and invoke `OpenCVVA` – OpenCV[8] Video Analytics. `IoTCameraProviders` have interfaced to our systems so that we have information about metadata, data points, and control points. All of these services and artifacts – `Kafka`, `Trigger` and `OpenCVVA` – are used often and pre-deployed. However, due to the changes of new `IoTCameraService` and cameras, for many data points about video, the `IoTCameraService` will not push the video to any other system but one has to pull (download) the video. In this situation, the interoperability problem is due to change of data delivery mechanisms. We detect that

[8]https://opencv.org/

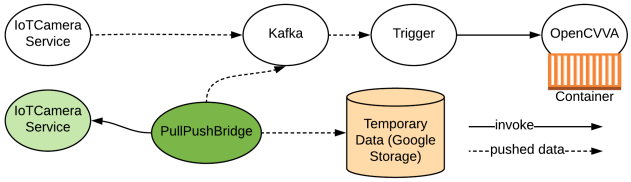

Fig. 5. Camera Example

we need a `PullPushBridge` for such data points. Suitable `PullPushBridge` is then deployed to allow us to get the data from such data points. All of this can be done on-demand at runtime. In Figure 5, we have `PullPushBridge` implemented using Google Storage as backend but one can also implement `PullPushBridge` using a simple Airflow with pull and push operators[9].

VI. Conclusions and Future Work

Our main idea for this paper is that we need to support dynamic interoperability solutions for IoT. This can be achieved by leveraging the concept of resource slices which are built atop existing software artifacts and resources. In this paper we have presented the Resource Slice Interoperability Hub (rsiHub) for IoT interoperability. We described the architecture, key components and examples. Our current work is to focus on two aspects: the prototype of slice recommendation and slice provisioning. The implementation work is continued to be updated in https://github.com/SINCConcept/HINC.

Acknowledgments: This work is partially supported by the INTER-IoT project through the subproject INTER-HINC. We are grateful to Lingfan Gao and Bunjamin Memishi for prototype implementation and discussion. Duc-Hung Le and Nanjangud Narendra provide useful ideas about HINC used for rsiHub implementation. The-Vu Tran provides information about cameras in Da Nang.

[9]The `IoTCameraService` is available at https://github.com/rdsea/IoTCloudSamples/tree/master/IoTCloudUnits/IoTCameraDataProvider. However, we have not made the `PullPushBridge` available.

REFERENCES

[1] G. Fortino, M. Ganzha, C. Palau, and M. Paprzycki, "Interoperability in the internet of things," https://www.computer.org/web/computingnow/archive/interoperability-in-the-internet-of-things-december-2016-introduction, December 2016.

[2] H. Derhamy, J. Eliasson, and J. Delsing, "Iot interoperability: On-demand and low latency transparent multiprotocol translator," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1754–1763, Oct 2017.

[3] M. Blackstock and R. Lea, "Iot interoperability: A hub-based approach," in *2014 International Conference on the Internet of Things (IOT)*, Oct 2014, pp. 79–84.

[4] M. Milenkovic, "A case for interoperable iot sensor data and meta-data formats: The internet of things (ubiquity symposium)," *Ubiquity*, vol. 2015, no. November, pp. 2:1–2:7, Nov. 2015. [Online]. Available: http://doi.acm.org/10.1145/2822643

[5] D. Le, N. C. Narendra, and H. L. Truong, "HINC - harmonizing diverse resource information across iot, network functions, and clouds," in *4th IEEE International Conference on Future Internet of Things and Cloud, FiCloud 2016, Vienna, Austria, August 22-24, 2016*, M. Younas, I. Awan, and W. Seah, Eds. IEEE Computer Society, 2016, pp. 317–324. [Online]. Available: https://doi.org/10.1109/FiCloud.2016.52

[6] P. Pace, R. Gravina, G. Aloi, G. Fortino, k. Fides-Valero, G. Ibanez-Sanchez, V. Traver, C. E. Palau, and D. C. Yacchirema, "Iot platforms interoperability for active and assisted living healthcare services support," in *2017 Global Internet of Things Summit (GIoTS)*, June 2017, pp. 1–6.

[7] C. Batini and M. Scannapieco, *Data Quality: Concepts, Methodologies and Techniques*, 1st ed. Springer Publishing Company, Incorporated, 2010.