Semantic Descriptor for Intelligence Services

Edgar Ramos
Ericsson Research
Finland
edgar.ramos@ericsson.com

Timon Schneider
Aalto University
Finland
timon.schneider@aalto.fi

Marie-J. Montpetit
Concordia University
Montreal, Canada
marie@mjmontpetit.com

Ben De Meester
IDLab – imec, Ghent University
Ghent, Belgium
ben.demeester@ugent.be

***Abstract*—The exposition and discovery of intelligence especially for connected devices and autonomous systems have become an important area of the research towards an all-intelligent world. In this article, it a semantic description of functions is proposed and used to provide intelligence services mainly for networked devices. The semantic descriptors aim to provide interoperability between multiple domains' vocabularies, data models, and ontologies, so that device applications become able to deploy them autonomously once they are onboarded in the device or system platform. The proposed framework supports the discovery, onboarding, and updating of the services by providing descriptions of their execution environment, software dependencies, policies and data inputs required, as well as the outputs produced, to enable application decoupling from the AI functions.**

***Index Terms*—Intelligence services, IoT, semantics, data model, taxonomy**

I. Introduction

Artificial intelligence (AI) frameworks increasingly allow for model development. Together with hardware accelerated model execution, this has led to a growing potential for intelligence functionality to be applied in multiple domains [1]. But one of the biggest challenges facing the development of computational intelligence is the lack of interoperability between AI components and their reusability: service and devices usually evolve independently. This is due in part to the combined heterogeneity and fragmentation of the datasets necessary for AI services to be efficient and also to the variety of development frameworks added to AI models' execution runtimes. The lack of interoperability results in friction in AI services including incompatible data types and formats, APIs and platforms required to execute the AI models and the cognitive framework operations. This usually results in the tight coupling of AI solutions to their target application. Currently, this coupling is one of the most challenging aspect of deploying AI in real-world applications. The high, and still increasing, level of coupling of AI solutions to the applications is discussed in detail in [2] and [3]. Coupling basically enforces the provision of one-of and tailor-made AI solutions that fits only or a small number of implementations.

Devices become intelligent as they execute applications, that in a domain or context, makes them look as intelligent entities. A device might execute one or multiple "intelligent" applications and the "perception" of their intelligence comes from how they are able to learn, or process data in a way that seems to resemble human-like reasoning. As it has been defined in [3], "An intelligent application makes use of one or several intelligent services that allows it to fulfill its task which is closely related to a use case". Therefore an application makes use of specific AI or ML models that can be abstracted as " s" (IS). However, as mentioned earlier, the high degree of coupling between the services and the applications can in some cases make them indistinguishable even when it is clear what requirements should a service cover and what are the specific features of the application for theintelligent processing of data. Therefore an IS refers to the execution of a AI or ML model as a function, that is used for an application in the context of its use case. For example, an intelligent car may have a self-driving feature that uses an object recognition model to fulfill its purpose of autonomous driving.

A. Related work

Following an increased scientific interest in the Internet of Things (IoT), researchers in industry and academia as well as the W3C have lead the development of semantic technologies – considered to play an essential role in the ecosystem of IoT for their ability to describe objects and information through open, integrated and machine-interpretable standards [4].

Many specifications were developed to describe semantic web services. Prominent ones are OWL-S [5], WSML [6], SAWSDL [7] or Hydra [8]. However, these are mostly successful in describing device-provided services. They are also tightly coupled with their technology stack (HTTP, SOAP or REST) and forego the possibility of local service execution. They are essentially web and cloud-based frameworks. And while they comply to the requirements of backoffice services for IoT they are incomplete to respond to the demands of the local intelligence services demanded by applications at the edge (autonomous systems, industy 5.0 etc.)

Several state-of-the-art projects are aimed at semantically describe data mining and machine learning algorithms and experiments in order to improve their interoperability and support model selection. Well-known ones include Onto-DM [9], DMOP [10], Exposé [11] and Mex [12]. ML-Schema combined the efforts and created one standard [13]. These frameworks as well as the solution presented in this paper have a common goal of overcoming the lack of interoperability in machine learning and increase the reusability of experiments and algorithms.

Over time, the Function Ontology has taken a more technology independent approach [14]. This data model allows

the description of functions independently of the technology used to implement them, making a positive step towards unambiguous description of services.

Other steps have been taken to decouple AI from applications through using RESTful APIs to remotely execute models(e.g., [15] and [16]). However, [2] and [3] argued that for many IoT devices this approach does not suffice because API updates for the targeted devices will be heavily complicated due to compatibility issues. Also, the semantic tools outlined here does not target the particularities of the device heterogeneity and their context might only be restricted to very few domains of deployment or use case.

This paper presents a novel solution to address both the fragmented dataset and the application/AI decoupling problem together by improving on data description. The proposed descriptors allow to unify data description in a comprehensive manner that can be generically addressed and utilized by the cognitive solutions. This is essential step to achieve interoperatiblity and enable a life cycle management of intelligence in a distributed environment. An example will show how the descriptors can address variable data fields and combine data, metadata, and policy features as inputs to the AI models. Following this introduction, the requirements and motivation to utilize intelligence services descriptions are provided considering multiple aspects related to discovery and exposition of intelligence features and functionality. In chapter III, the proposed taxonomy and data model are defined and mapped to the requirements and functionality outlined in the previous chapter, and an example of IS description for object detection is provided. Finally, the conclusion presents a summary of the work and avenues for future work.

II. Intelligence service description

A semantic description of services allows automated third-party management services to automatically discover and execute exposed functions without human intervention when needed. Its deployment methodology follows the principles of content negotiation inherited from the semantic web [17].

A. Addressing the problem space

The main aspect to consider when describing a IS is clearly define *what problem it is trying to solve*, because this will indicate what it is attempting to do. The problem-solving dimension is inherently linked to *intelligence*. In practice, machine learning algorithms (as a subset of AI techniques) are composed of various mathematical or logical functions that are adapted over time (trained) using data from previous experiences, by co-relating data sets with new data or by experimenting with outcomes. It is an iterative process modifying the handling of the input until the desired outcome likelihood is maximized. Therefore, the main objective of the intelligent functions is to provide a suitable solution to a particular problem, which fits the premise of applied reason.

If the concept of IS is reduced to a minimum expression, e.g., a simple linear predictor, the service is consisting of the simple function F(x) = y (where F(x) = m * x + a for the linear predictor). This is a very generic function that has an almost infinite amount of uses. The key issue in this case is the context in which the problem is laid out. The simple function may provide a solution in a domain, e.g., a real estate prices domain, when applied to the right dataset. Generic algorithms can solve many types of problems; therefore one generic IS may have many problem descriptions. In consequence, the input of the service needed to solve a specific problem is important as well, and should be outlined in a domain context . In particular, the input domain may differ from the problem domain, as long as the definitions are compatible. However, the output context description is strongly coupled to the problem domain as it expresses the type of solution provided by the *transformation function* that in practice is an IS. Of course, the output may have additional semantic descriptors, from other domains than the problem domain, to increase the understanding accuracy of the output's multiple characteristics.

Some problems may be simple to express semantically, while others require a more complex description and specification, even when the actual IS implementation may be quite simple. For example, in some domains, a term from a vocabulary is enough to define a problem, such as "human recognition". Certainly, the problem definition is incomplete without defining proper inputs and outputs. An input describing a type of data, e.g., an array of bits representing a bitmap of an image, is in some cases insufficient to define a solution-space depending of the actual implementation of the service: the algorithm might require that the data is free of color information or has a minimum resolution or image size. Additionally, it might only work (or work better) with specific type of images, e.g., colored with heat maps or equalized to black and white. If it is specifically designed to differentiate humans from animals, the image should not have a manikin or other type of dummy representations of humans or it may result in a erroneous recognition. All these pre-conditions need to be semantically described to provide better understanding of the limitations and capabilities of the services used by automation systems.

In the context of automated vehicles, recognizing a red light by a computer vision system is be critical for such a system to stop at an intersection, or to brake if the vehicle in front has also activated the brakes. The problem formulation in that case can be quite complex, if specified as "detecting a red light activated from the taillight of a vehicle when braking from an image". There are multiple entities and relationships between them that need to be expressed. Even though vocabulary could reduce such semantic construction to a single term, it might not be able to capture the whole semantic meaning of the expression. Questions like "how is a vehicle defined", "does it include radio controlled toys", "what kind of hue is considered red and differentiated from dark orange", "what shape does a tail light have" will stay unanswered. These issues may have varying degrees of relevance according to the domain context , and it is expected that these kinds of important semantic constructions must be available for each

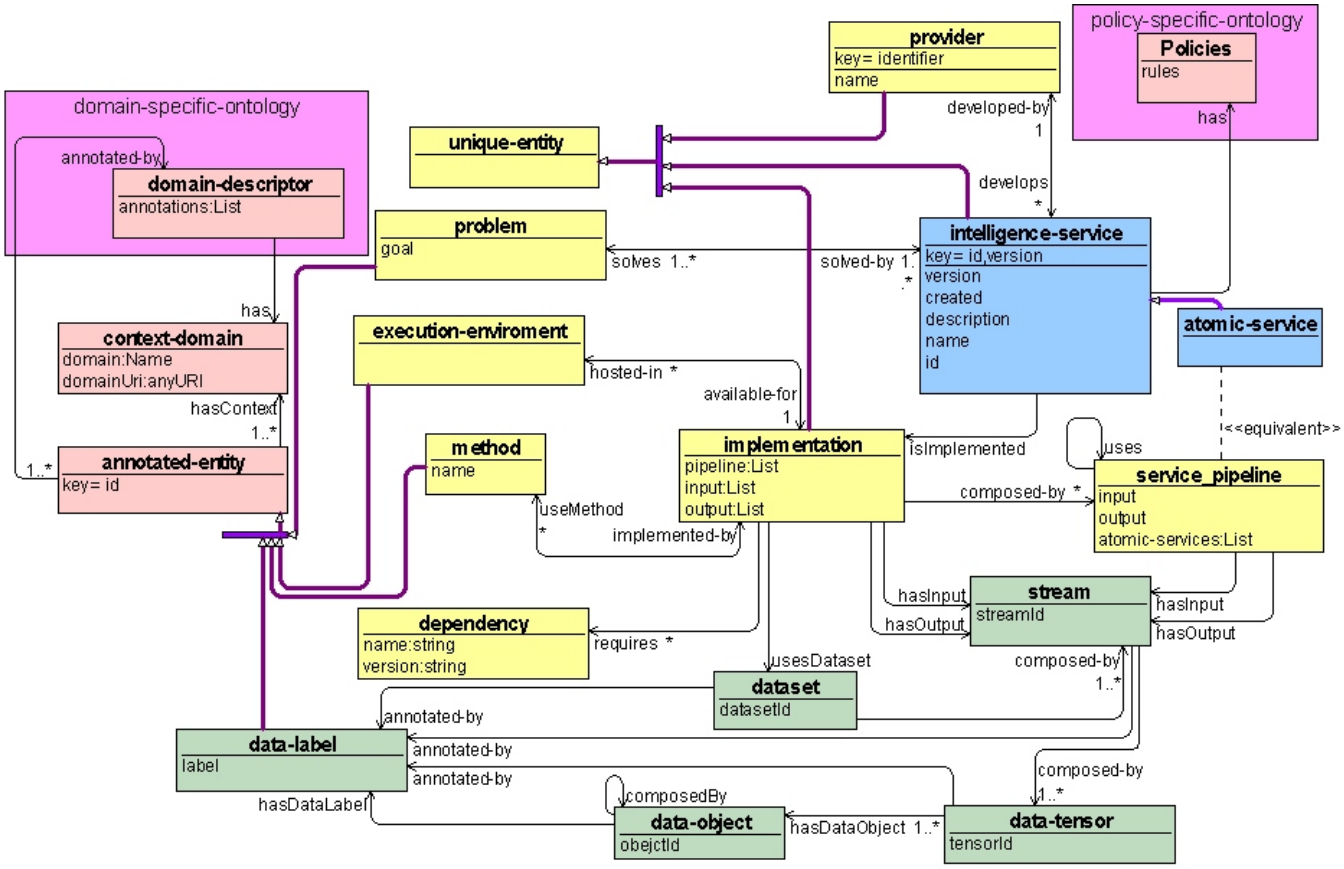

Fig. 1. Intelligent services description taxonomy

domain accordingly to the increasing need of them to exist and describe the intelligence addressing the domain problems requiring automation. This is a process that will go hand in hand and advance together with the availability of intelligent solutions addressing specific problems.

B. Limitations and Requirements

Intelligent devices are often heterogeneous in nature. They may or may not include, acceleration hardware, on-device user interface, continuous connectivity, battery operation, non-volatile memory storage, etc. In consequence, services that are executed and exposed for applications in such devices must be able to express requirements and take in account the advantages or limitations that the hardware or the execution environment may impose to the services. These may have an effect on the performance metric of a service, e.g., latency, available memory, CPU load and free cycles etc. They may have an impact on the correct application behavior or on other applications executed concurrently in the same device. Even when a service is expected to be executed remotely (in the cloud or edge node), the client application is executed in the device. This means that connectivity must be in place to enable communication between the service and the application. A device with sporadic connectivity will not be able to have access to networked services which will further impact the application execution.

The minimum device hardware capabilities available for a service at the time of discovery and selection are important. They provide an indication of the expected performance of any algorithm in terms of execution, taking into account specialized hardware and in some cases software support (i.e., libraries, interpreters, or even an specific operating system) that may be required by the service. The service itself, being executed in a runtime environment, imposes requirements on the device to support such runtime environment or, in case of being remotely executed, to have the capability of reaching the service remotely using the network stack and access protocols supported by the publisher of the service.

C. Implementation and Life-Cycle Management

A specific problem may possibly be addressed by nultiple ISs. This means that it can be specified in a domain while being dealt with in different manners using different input data or processing methods and produce, in some cases, different output. The particular processing, operations, and the approach used to deal with the problem requirements constitute the implementation of the service which may be directly responsible for the limitations and requirements of the service. It may also affect the performance of the service from the application perspective (e.g., accuracy, precision, error rate, etc.). Using the same example as previously, that of a the humans recognition service, it might be possible to recognize

humans from various inputs: in some implementations the recognition may use audio, others may use an image, and it is even possible to imagine an implementation based on smell (or detected components in an air sample). The input of the first two may be the same, a video feed that includes ambient sound in addition to the image frames. Some implementations may even have the same input and produce the same output but using a different computational technique. For example, a classification IS may be implemented using a naive Bayes classifier, a simple decision tree, a random forest, or a neural network among others. Some of those implementations may be more memory hungry, others more processing intensive, some require extensive training data to produce good results and some are more suitable for small number of classification categories while others scale better when the number of category labels is high.

Another aspect related to IS implementation is versioning. A service being used by an application with a specific implementation may be updated with a different implementation version over time. In some occurrences, the update may require changes in the performance or feature of the hardware (ex., requires a GPU unit for acceleration). In the best of cases a new version fine-tunes the learning parameters or extends the service operation domain to cover additional cases previously not able to be processed. The implications of the implementation version changes may impact some applications' performance with respect to previous versions of the same service. This introduces the need for the life cycle management of the services in devices to consider not only discovery and onboarding of the ISs but also updates and version handling.

In some cases, some aspects of the life cycle management of the service may be handled or triggered by the application: a service discovery in a device, which results in an unsuccessful match, could trigger a wider search for a suitable service from a remote repository to onboard the service in the device. In addition, the application may refuse to switch to an updated version of the service if it considers that it does not fulfills its requirements anymore. Enabling an automatic management of the life cycle of a service requires that the characteristics associated to changes in the implementation are also exposed and semantically meaningful for applications. Hence the relevant semantic properties supporting it are necessary to be present in the description of the IS.

D. Intelligence services composition and pipelining

Another characteristic of the ISs is its high potential for composition. A service implementation can, in many cases, be decomposed in more atomic independent functions that in combination of their inputs and outputs produces the intended result of the service. In its simplest form, this resembles a cascading serial processing and, in other more complex compositions, is comparable to a pipeline of parallel processes with local junctions and splitters that finally converge in a final outlet. The service reusability principle promotes this type of design pattern. It makes sense for some implementations to separate a problem into sub-problems that are addressed first separately, and their outcome then combined to provide a result to the more complex problem. From the perspective of a IS provider it is reasonable to reuse already available solutions in a device and integrate them as part of another service when and if their functionality is useful. In consequence, a service implementation may be composed with other more atomic services, with their own semantic properties, and become part of the composed service elements. As a result, support for the semantic integration of atomic service semantic properties must also be present in the description of a composite intelligence service.

E. Associated datasets

Most intelligence functions have an intrinsic learning component that is characterized by data. In many cases, some datasets are associated to the services supporting an inference process, as part of a training process or to map or transform inputs and outputs from one domain to other, e.g., to support normalization. In addition, the service's own generated data can be used in a close-loop fashion to optimize an objective function or goal, in the way that reinforcement learning does. The generated data might require to be stored in non-volatile memory, logged, or sent for further processing by a different application, execution instance, device or computational network domain.

The handling of all this associated data is in many cases left to end-user-agreements or policies dictated mainly by the service providers. But one of the interesting properties of exposing services directly on devices is that it allows a higher degree of control of the data that flows through the APIs used to execute the services. In addition, in the case of the services executed in the device itself, control is required to enforce polices that are dictated upfront by the intelligence service provider, in relation to the data associated to inference that can be considered proprietary. Meanwhile, the data provided by the device or by the user of the device may follow another set of policies that needs to be enforced as well in the same device. In some cases, a negotiation process and agreements may introduce new policies or override default behaviors. The conception, negotiation, conflict resolution and enforcing of the set of policies is a topic in its own, but several efforts to express them semantically have been proposed [18][19] and exchanged using digital service level agreement. An application discovering a fitting intelligence services may need to understand the impact on the data handling in the device that might embody the requirements from the intelligence provider. An example is the restriction on keeping certain data on non-volatile memory without encryption or the policy to not allow forwarding of data to other devices, or even the to other non-authorized applications running in the same device. Equally, the device can have its own local policies that requires a selection or negotiation process with services that may not have fully compatible or conflicting rules for example if the device does allow any input data to be sent to cloud based services or if the data requires anonymizing before sending it further.

III. IS DATA MODEL AND TAXONOMY

Based in the reasoning provided in the previous sections, it is possible to define what elements and relationships are needed to semantically describe an intelligence service. A schema with semantic relationships could easily be represented by classes and their connectors.

A. Classes

The classes of the IS schema are semantic entities with properties and components. Each class may be instantiated several times but still be part of the same IS descriptor. This approach provides freedom to describe a service from multiple domains' perspective instead of a single description element, and by linking them it enriches the description further, even when an initial version was already provided. Also, some of class instances may be reused by several IS descriptors, e.g., the provider of one service could be the provider of other services. The model is based in four abstract classes providing properties that are used as baseline for the others parts.

unique-entity. Objects instances that are unique and if compared to other object of the same class must always be consider different. This may be achieved by using an unique identifier in the context of the class that extends it. Classes such as ISs, atomic services, provider and implementation inherit datatype properties from this class.

annotated-entity. The especial property of an annotated-entity is that they belong to a context domain and the semantic annotation may vary according to what domain they have been defined. Therefore, an annotated-entity has a context-domain and a domain-descriptor with a set of annotations related to the context-domain. A data-label, method, problem, execution-environment are examples of subclasses from annotated-entity. An annotated-entity has a context-domain (see next) and may be annotated by a domain-descriptor that specified elements that are derived or defined from the correspondent domain.

context-domain. The domain provides a context in which the annotated-entity operates. The domain is established by a pointer or reference to an ontology, vocabulary, data model or be a placeholder for a customized/proprietary description.

domain-descriptor. Provides description of an annotated-entity accordingly to the vocabulary and semantic constructions available from the context-domain reference.

The abstract classes are used as very basic building blocks by the classes belonging to the data model that describe an IS. It is then addressed by the elements and their relationships depicted in figure 1. The elements' semantic definition and their relationships aims to address the requirements discussed in II. A mapping table between the requirements and the elements of the taxonomy are presented in table I. The classes belonging to the taxonomy's data model used to describe a service include:

intelligence-service This class represents an IS and its metadata. An IS takes inputs, executes a set of calculations, and follows processing according to one implementation of a method and returns an output that matches the solution for a problem. The service is hosted in an execution environment (although it might be available for more than one), it also may have software dependencies and policies that affects the data and processing handling. An IS can be uniquely addressed by the combination of its version and its identifier. It may also have some information elements, such as date of creation, a human readable description and a name.

provider. ISs are handled and managed by an intelligence provider. The provider may have ownership of the model or just facilitates its availability. It is considered the responsible entity for the publication, updates and changes of the specific service described.

execution-environment. The execution environment exposes information of the type of software platform or system especially required to execute a service-specific implementation, and if required it may specify hardware components. The execution environment can be described according to a domain-context, focusing on different domain-dependent aspects. Different implementations of the same service may have different execution-environment requirements.

dependency.It describes specific software modules necessary to execute the implementation of the service within an execution environment.

problem. It describes the applicability of the service to a specific domain. It expresses semantically the goal (or intent) that the service attempts to fulfill in order to resolve it. Therefore, a problem is defined by what it desires to solve. An IS might address several problems with the same implementation, or different versions of a service may map to different problem solving.

method. The method implemented to achieve the goal of the IS is outlined here. A method may be known by different names or level of detail according to the descriptive domain.

implementation. An IS implementation deals with the actual solution to the problem solved by the service. It might follow one step or multiple steps to accomplish the service goal(s). An implementation may pipeline other IS's (atomic-services) or point to methods or models that may be standalone or combined with each other. Each version of a service may have a different implementation. Also, even when two separate ISs attempt to solve the same problem description, their implementations may be different. An implementation processes an input an produce an output in the shape of stream.

atomic-service. It abstracts an element of a pipeline that is used to compose an IS from other ISs previously defined and may be individually invoke-able from the main service. A pipeline of atomic services can be aggregated into one more complex IS.

service-pipeline. It allows to define a sequential pipeline of atomic-services and sub-sets with their input and outputs, allowing more complex constructions from baseline services. This is especially appealing for services that may provide a library of functions. Semantically a service-pipeline is equivalent to an atomic-service even it adds additional elements that wrap-up the atomic-service own definition of inputs and outputs. A service-pipeline can use in its elements another service-pipeline or atomic-services.

Semantic classes	Purpose - Intelligence Service requirement addressed
intelligence-service	Defines uniquely a service. Identifies and aggregates all the data related to a service using fixed navigable relationships
provider	Defines uniquely a service provider. Certification of origin for the service
execution-environment	Describes and addresses the needs for platforms, hardware or runtime environment required by the implementation to be executed.
dependency	Describe the requirements of software modules, libraries, etc. required for the implementation to be executed.
problem	Describes in multiple domains the goal of the service. Matching of domain applications intents requirements to a service
method	Describes what techniques were used to provide a solution in the implementation of the service. Preferences on specific techniques and algorithms can be discovered and specified
atomic-service	Used in the creation of intelligence pipelines to abstract other services as part of a composed service. It allows to use services as construction blocks for other services. It allows to solve problems that requires multiple services to operate together without need of generating a new service targeting a specific platform and requirements
implementation	Ties the particularities of one implementation of the service to a service version, and its requirements, and provides information about input required and output generated by the service. Also, it indicates if the service is implemented by a pipeline of multiple services connected.
service-pipeline	It abstracts atomic-services so a subset of inputs and outputs from services can be used to match inputs and output of other atomic-services. Also, to create parallel pipelines of services that later are input for final pipeline that produce the output of the service. It enables more complex pipelining than sequential processing of services.
stream, data-tensor, data-object, dataset	Data input/output semantics. Directly used to discover a service API and adapt the application and device data to fit the requirements of the service and understand the way of deploying it.
data-labels	Provides additional annotation to describe the data classes that can be mapped to specific domains semantics (ontologies, vocabularies, data models, etc.) that an application is able to understand.
policies	Semantic constructions that provide rules or specific instructions on the data and processing aspects of the services. The policies may have multiple origins (device, application, user, service provider, o&m, etc.) and in the deployed version might resemble a digital Service Level Agreement (SLA) that it is enforced by the signing parties in their respective domain of operation (device, platform, application, etc.).

TABLE I
TAXONOMY CLASSES MAPPING TO DISCUSSED NEEDS AND REQUIREMENTS

stream. It represents a set of data with a correlation (source, time, applicability, etc.). E.g., data from sensor of one device or domain, or weather data from an online service.

dataset. Contains the information about the dataset that is used by the implementation. In some cases, the dataset information gives an idea of the training data used for the model implemented. In other cases, it represents information that is needed by the service, in addition to the input, for service delivery. In that case, it is expected to be provided together with the service or at least provide the means to acquire it. A data package or data access API might be delivered together with the service, and the semantic description of the data and their sources is presented by this class. The dataset may also be used as auxiliary input for inference or as table for a mapping function of the output, for example.

data-tensor. It provides metadata in reference to one tensor from a stream. Multiple data-tensors conforms one stream. A tensor has at least one dimension and it correlates information from one object, and therefore a lower granularity than a stream. E.g., information about one picture, a video, or a genome sequence.

data-object. A data-tensor may be composed by several data-objects representing labels, features or categories. A data object might be composed by other simpler but semantically meaningful data-objects. The data-object specific semantic characteristics can be described with this class. An example of data-object can be a temperature value from a sensor, a timestamp, or a composed coordinates of an device position.

data-labels. It presents additional metadata about a data related class. A data-label can be described by a domain and semantic descriptors from the domain's taxonomy and annotates dataset, stream, data-tensor and data-object.

policies. It represents a set of rules and instructions related to the treatment of processes and data interacting with the service. The rules and interaction with the entities involved can be expressed by one or multiple policy specific ontology or semantic constructions.

B. Service description example

Mobilenet [20] is a pre-trained object recognition model based on the ImageNet [21] dataset that is relatively light weight in processing and memory requirements. The model classifies an image within the 1000 classes of ImageNet by providing the probability of the object in the image matching each of the categories. The model used in this article example is encoded using ONNX[1] and it is part of the ONNX ZOO[2]. The model requires the input to be a matrix of dimensions N x 3 x H x W. These values stand respectively for the number of batches of images, the number of RGB-channels, the height of the image in pixels and the width of the image in pixel. Also the input image RGB channels are expected to be normalized. Since the model used by this example is encoded in ONNX format, it can then be executed only in runtimes supporting ONNX native execution. An example service description is

[1]https://onnx.ai/

[2]https://github.com/onnx/models

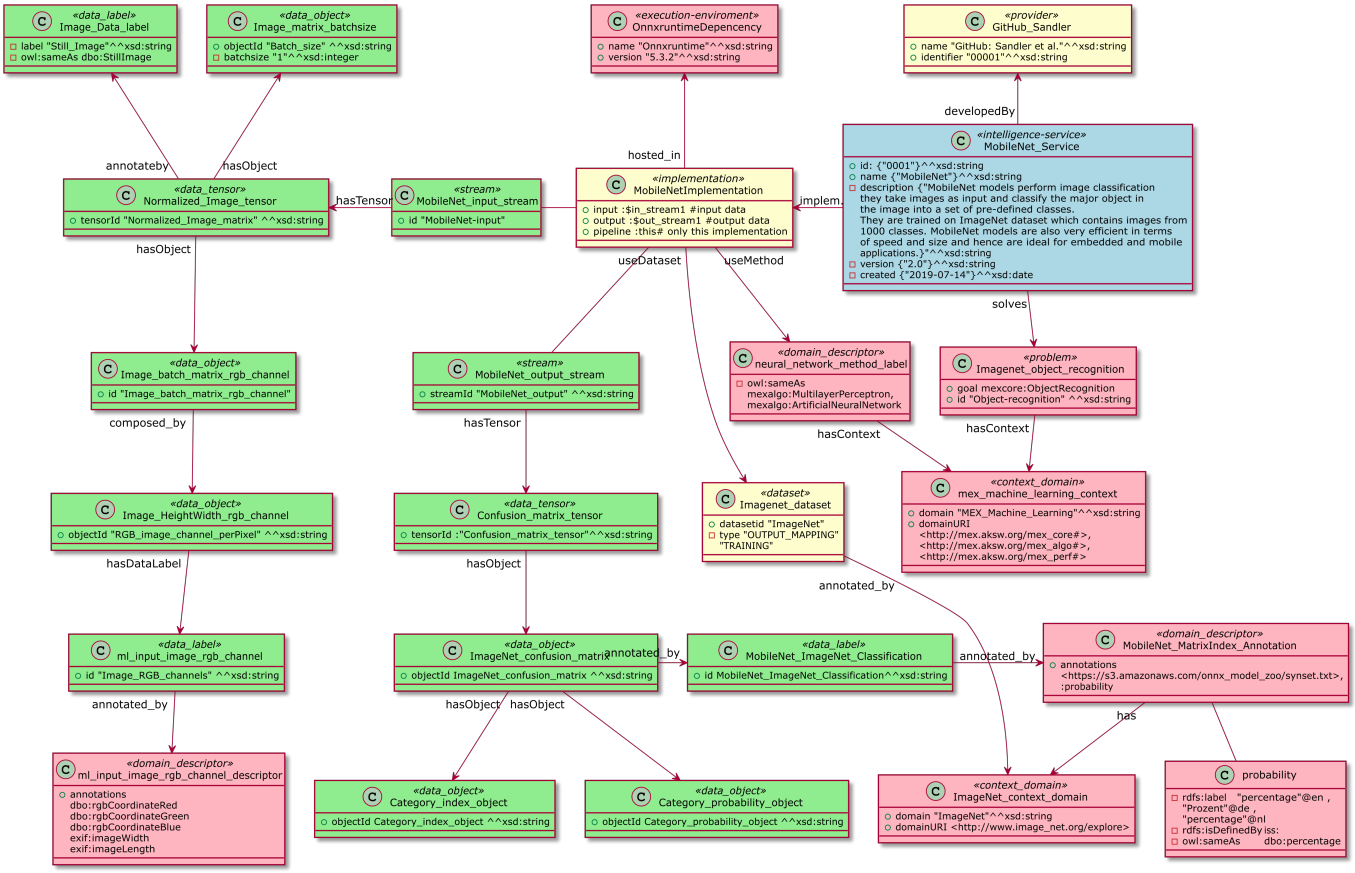

Fig. 2. MobileNet semantic service description visualization using UML class notation

depicted by figure 2. In this example it can be seen multiple context domain descriptions, such as the confusion matrix or the RGB channel from images. The domain descriptors use vocabulary, classes and objects already defined by other ontologies and are expected to be known and handled by the application deploying such model. Also the application (or even the device itself) may be able to update the knowledge of an ontology by acquiring support during the on-boarding process.

The MobileNet model requires a normalization of the images before the actual inference to avoid errors and increase efficiency. In a more elaborate example, the normalization function could be described as a separate atomic service. In that case, the atomic service becomes part of the pipeline together with the MobileNet service described earlier. Figure 3 shows an example of such composed service targeting an Android platform. The input descriptor differs from the ones used to describe the MobileNet service to accommodate the input required by the first service in the chain (the normalization python function). Additionally, new dependencies are now provided. Python and MXNet are required to run the preprocesing function and therefore are needed in the composed service. Since this case is a sequential pipeline of atomic services, there is no need to define a pipeline-service class and the first service output (the pre-processing function) becomes the second service input (MobileNet service).

The pre-processing function should also be described as a service in order to allow the service processing layer in the device to map them into the device execution environment and processing. It can be noted that the problem definition for both examples is the same. The MobileNet model requirement of normalization could be documented either as a domain specific description that specialize the input with the parameters required for the image or as part of the policies that should be applied to the data. In this article the policies are not yet being considered and therefore is not visible in the example, but the difference on the input (from a file in one case to a input tensor matrix in the other case) provides a guidance to the application on what service to consider using base in what it can handle. Certain applications would be able to produce the matrix required by the MobileNet service example, but some others would only be able to produce a picture in a JPEG format, therefore the selection of model will match what the application or device can handle.

The framework is being applied in an vertical agriculture use case. In there, complex relationship between sensors, cameras and growth parameters need to be considered together to diagnose problems but also predict yields and ”program” the best environment for a specific crops based on resulting size, color and weight. By using the descriptors from this paper is

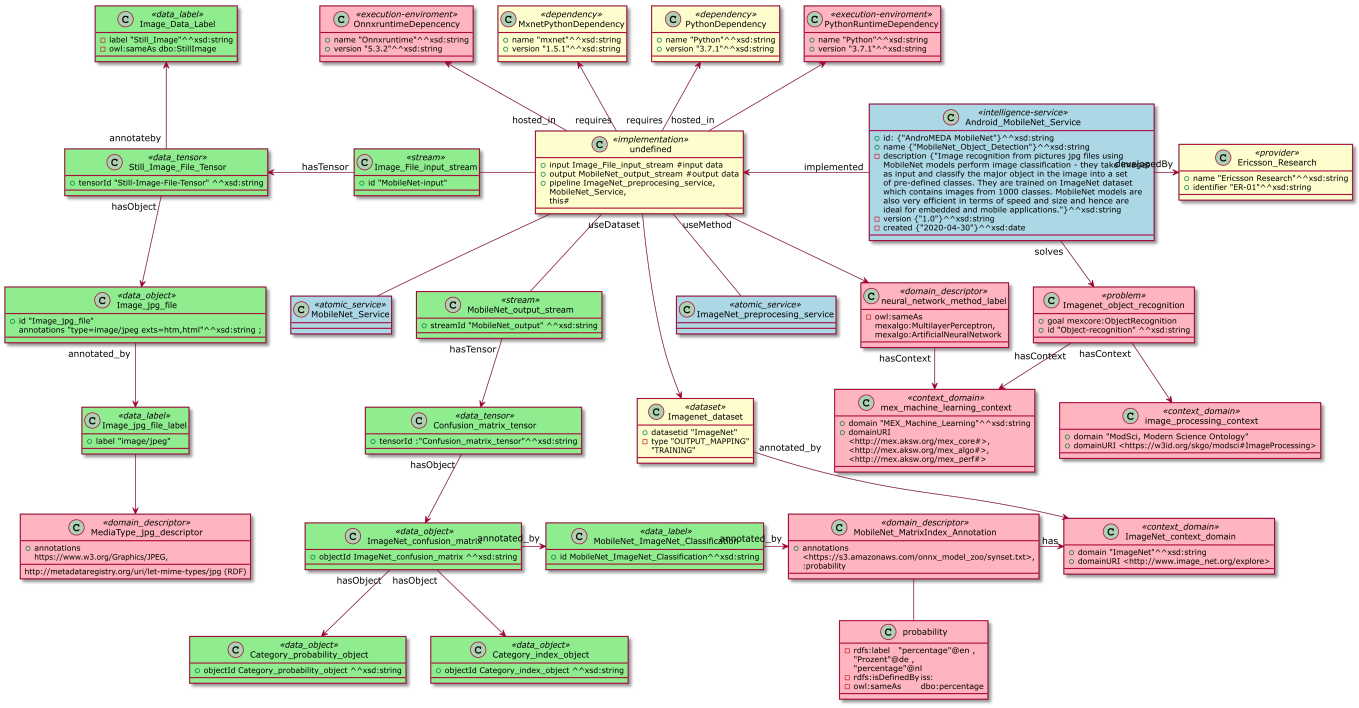

Fig. 3. Description of a composed-service including MobileNet using UML class notation

it possible to re-use the basic intelligence function when the crop requirements change.

IV. Conclusion

This article presents a framework for emerging data-driven and distributed ISs. The framework consists on a taxonomy of semantic descriptors which provide a solution to two data problems in emerging systems: dataset fragmentation and the AI/Application coupling. Indeed, one characteristic of novel system in IoT and automation is the fragmentation nature in type and location of the data itself and its non uniform use in local and web-based processing. The inter-related functional elements of the growing number of modern applications requires tailored but yet flexible solutions. Data related descriptors resolve some of the issues related to the current coupling of AI functionality to specific applications, discouraging AI application migrating from cloud-bases implementation to distributed systems. Because of this coupling, data cannot be re-used locally to create more advanced services or meta-applications. Data related descriptors allow the data to be readily available to local functions hence for those functions to discover the data they need. The article does not present a new data description language, instead it leverage on the use of existing descriptors in an expanded context. It is envisaged that the proposed approach will allow the creation of both targeted and flexible distributed ISs in the future especially for next generation IoT and control systems.

References

[1] M. S. Mahdavinejad, M. Rezvan, M. Barekatain, P. Adibi, P. Barnaghi, and A. P. Sheth, "Machine learning for internet of things data analysis: a survey," *Digital Communications and Networks*, vol. 4, no. 3, pp. 161–175, 2018.

[2] E. Ramos, R. Morabito, and J.-P. Kainulainen, "Distributing Intelligence to the Edge and beyond [Research Frontier]," *IEEE Computational Intelligence Magazine*, vol. 14, no. 4, 2019.

[3] E. Ramos and R. Morabito, "Intelligence stratum for IoT. Architecture requirements and functions," in *Proceedings - IEEE 17th International Conference on Dependable, Autonomic and Secure Computing, IEEE 17th International Conference on Pervasive Intelligence and Computing, IEEE 5th International Conference on Cloud and Big Data Computing, 4th Cyber Scienc*, 2019.

[4] T. Kamiya, V. Charpenay, S. Käbisch, M. Kovatsch, and M. McCool, "Web of Things (WoT) Thing Description," W3C, Recommendation, 2020, https://www.w3.org/TR/wot-thing-description/.

[5] D. Martin *et al.*, "OWL-S: Semantic Markup for Web Services," W3C, Member Submission, 2004, https://www.w3.org/Submission/OWL-S/.

[6] J. de Bruijn, D. Fensel, M. Kerrigan, U. Keller, H. Lausen, and J. Scicluna, *Modeling Semantic Web Services: The Web Service Modeling Language*, 1st ed. Springer Publishing Company, Incorporated, 2008.

[7] J. Farrell and H. Lausen, "Semantic Annotations for WSDL and XML Schema," W3C, Recommendation, 2007, https://www.w3.org/TR/sawsdl/.

[8] M. Lanthaler, "Hydra core vocabulary," Google, Unofficial Draft, 2020, http://www.hydra-cg.com/spec/latest/core/.

[9] P. Panov, L. Soldatova, and S. Džeroski, "Ontology of core data mining entities," *Data Mining and Knowledge Discovery*, vol. 28, pp. 1222–1265, 2014.

[10] C. Maria Keet, A. Ławrynowicz, C. d'Amato, A. Kalousis, P. Nguyen, R. Palma, R. Stevens, and M. Hilario, "The data mining optimization ontology," *Journal of Web Semantics*, vol. 32, pp. 43 – 53, 2015.

[11] J. Vanschoren, H. Blockeel, B. Pfahringer, and G. Holmes, "Experiment databases. a new way to share, organize and learn from experiments." *Machine Learning*, vol. 87, pp. 127–158, 2012.

[12] D. Esteves, D. Moussallem, C. Neto, T. Soru, R. Usbeck, M. Ackermann, and J. Lehmann, "MEX Vocabulary: A Lightweight Interchange Format for Machine Learning Experiments," in *Proceedings of the 11th International Conference on Semantic Systems*, 2015, pp. 169–176.

[13] G. C. Publio, D. Esteves, A. Ławrynowicz, P. Panov, L. Soldatova, T. Soru, J. Vanschoren, and H. Zafar, "ML-Schema: Exposing the

Semantics of Machine Learning with Schemas and Ontologies," *arXiv preprint arXiv:1807.05351*, 2018.

[14] B. De Meester, T. Seymoens, A. Dimou, and R. Verborgh, "Implementation-independent function reuse," *Future Generation Computer Systems*, vol. 110, pp. 946–959, 2020.

[15] C. Olston, F. Li, J. Harmsen, J. Soyke, K. Gorovoy, L. Lao, N. Fiedel, S. Ramesh, and V. Rajashekhar, "TensorFlow-Serving: Flexible, High-Performance ML Serving," in *Workshop on ML Systems at NIPS*, 2017.

[16] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A Low-Latency Online Prediction Serving System," in *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, 2017, pp. 613–627.

[17] R. T. Fielding and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content," RFC 7231, 2014. [Online]. Available: https://rfc-editor.org/rfc/rfc7231.txt

[18] L. Kagal, T. Finin, and A. Joshi, "A policy based approach to security for the Semantic Web," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2003.

[19] S. A. Chun, V. Atluri, and N. R. Adam, "Policy-based Web service composition," in *Proceedings of the IEEE International Workshop on Research Issues in Data Engineering*, 2004.

[20] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.

[21] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.