Functionalities, Challenges and Enablers for a generalized FaaS based architecture as the realizer of Cloud/Edge continuum interplay

George Kousiouris
Department of Informatics & Telematics
Harokopio University of Athens
9, Omirou Str. 177 78, Tavros, Greece
gkousiou@hua.gr

Dimosthenis Kyriazis
Department of Digital Systems
University of Piraeus
Karaoli & A. Dimitriou 80, 18534 Piraeus, Greece
dimos@unipi.gr

***Abstract*— The availability of decentralized edge computing locations as well as their combination with more centralized Cloud solutions enables the investigation of various tradeoffs for application component placement in order to optimize application behavior and resource usage. In this paper, the goal is to investigate key functionalities and operations needed by a middleware layer so that it can serve as a generalized architectural and computing framework in the implementation of a Cloud/Edge computing continuum. As a primary candidate, FaaS frameworks are taken under consideration, given their significant benefits such as flexibility in execution, maturity of the underlying tools, event driven nature and enablement of incorporation of arbitrary and legacy application components triggered by diverse actions and rules. Related work, gaps and enablers for three different layers (application design and implementation, semantically enriched runtime adaptation/configuration and deployment optimization) are highlighted. These aid in detecting necessary building blocks of a proposed generalized architecture in order to enclose the needed functionalities, covering aspects such as diverse service environments and links with the underlying platforms for orchestration, dynamic configuration, deployment and operation.**

Keywords—Function as a Service, Cloud computing, Edge computing, computing continuum

I. INTRODUCTION

The current Cloud computing landscape is characterized by an extreme diversity of offerings and services, incorporating multiple solutions. These include centralized Cloud providers (such as typical VM offerings, dedicated nodes, hardware enhanced resources such as GPUs and FPGAs etc), edge and fog environments, HPC facilities, mobile computing applications etc. implementing the Everything as a Service approach. On the other hand, applications are typically consisted of a multitude of components, others in need of locality and others in need of significant computational resources to fulfil their scope and objectives. These applications portray varying abilities to exploit the underlying services, varying requirements for operation and control as well as technologies and programming structures/languages on which they rely[2].

For achieving a true smart cloud computing continuum, i.e. a unified approach on available resources, one should examine the domain from three main perspectives:

- **Perspective 1:** Continuum in terms of application design and definition, leading to a generalized approach for how one should build their software or service stack. Key aspects would include the ability of the software to react to changing and dynamic conditions as well as handle unexpected loads, failures and adaptations. Furthermore, the software should be adapted to different computing paradigms, as well as integrate functionalities that enable it to exploit the new execution environment. Enabling the combination of edge/cloud resources can significantly enhance aspects such as latency and further application benefits[3]
- **Perspective 2:** Continuum in terms of functionalities, deployment and management approaches that handle all underlying resources in a similar manner, abstracted to the developer and to the majority of the underlying mechanisms. The scale and complexity in the new cloud/edge interplay has become quite complex thus managing dependencies and mechanisms across continuum resources has become a daunting task[4]. Once an application has been defined, can it be seamlessly managed by an underlying framework and distributed based on a given set of goals and constraints? Can the application exploit specific features of the used service, such as GPU processing, multiple cores in the node etc? If yes, which executable version of it and with which parameter set should be used during deployment, without user intevention? In order to achieve that, a relevant set of semantic descriptions should exist that cover and describe the range, type and diversity of the available services, as well as capabilities of application components. Through the combination of these, inference as to which services can be exploited by a given component, including functional adaptation aspects as well as non functional requirements.
- **Perspective 3:** Continuum in terms of the relativity inserted between space (of deployment) and time (of execution). Similarly to the Physics-driven concept of space-time continuum, one cannot separate the two concepts in the eyes of the observer (application end user), who is interested in the final optimized result[5] . Utilizing a centralized service far away from the observer may introduce latency or other factors that render this selection as more time consuming than a localized deployment. The analysis of this trade-off should result in a unified space-time combinatorial approach in order to handle service selection and execution, also taking under consideration other computing space-time continuum distorting factors. Like gravity can be considered as the main distorting force in the space-time continuum, multitenancy and resulting performance interference[6] can be considered as the equivalent in modern, service oriented environments. Therefore, dynamic incorporation of this factor's weight on the final optimization should be measured, included and taken under consideration.

In order to achieve these objectives, suitable middleware frameworks need to be designed, deployed and operated across the continuum. The aim of this paper is to investigate what is the status in some of the key areas mentioned above, in order to highlight existing capabilities as well as according gaps and enablers. As the main middleware platform, (mainly open source) FaaS frameworks are taken under consideration, given their flexibility and distributed nature and are scrutinized under the above 3 perspectives. Section 2 includes the suitability of the FaaS model for the specific purpose as well as investigation of the current status in application design and adaptation (Perspective 1), Section 3 investigates the usage of middleware and semantic technologies (Perspective 2), while Section 4 studies the space-time continuum capabilities (Perspective 3) Finally the paper proposes a high level architectural approach that would enhance the ability of FaaS frameworks to meet this diverse role in Section 5. Finally Section 6 concludes the paper.

II. FAAS CHARACTERISTICS AND APPLICATION DESIGN (PERSPECTIVE 1)

A. Why FaaS? Key FaaS Characteristics of Interest

One of the main benefits of the FaaS model is the fact that it is built around the most sophisticated variation of the pay-as-you-go concept, the **pay-as-you-execute model**, thus only charging when the application code (or specific function) is actually executed (including other billing factors such as function runtime and memory size). Therefore not even the idle time of an application counts towards its costs. What is more it alleviates from other aspects such as server environment maintenance. Break down into functions enables easier scalability and elasticity of the applications, thus better ability to exploit elastic resources and services, as well as software modularity and maintenance. Therefore it strengthens the benefits of a migration towards a cloud/edge service environment. This function-based break-down also enables the easier distribution of tasks between centralized and edge resources available.

This is further strengthened by the fact that application structure is fed in the underlying FaaS platform (such as Openwhisk) which handles many of the deployment, configuration and orchestration needs. FaaS has been in the recent years among the highest growth public cloud service types while the need to optimize cost savings from cloud services is the top priority for cloud users in 2020[1].

One of the main abilities of FaaS platforms such as Openwhisk is to include diverse components and behaviours and adapt them to event driven sequences and workflows, a feature known as **polyglot ability**. Its operators include notions such as Actions (application code that may include pure function code enforced on input message data, legacy non-FaaS components, arbitrary and diverse executables of any programming language in docker containers) and Rules (used to associate one event trigger with one or multiple actions, therefore enabling the definition of complex workflows). One key aspect is that the event sources can be anything i.e messages arriving on Message Queues, changes in databases, web interactions, service APIs etc. This enables bridging the FaaS model with other popular existing approaches such as microservices, REST services, legacy web applications or any arbitrary legacy component through encapsulation of existing workflows and components in FaaS wrappers and executable containers. Therefore the operational migration and combination with newly introduced FaaS based functionalities is much easier for existing applications.

Functions can typically range from small and lightweight to larger and more computationally demanding, therefore very suitable for the cloud/edge interplay scope and offloading trade-offs investigation, depending on available hardware on the edge, latency considerations etc. However careful consideration should be given to aspects such as the distribution of the load, auto-scaling mechanisms, operational tasks and function limitations[6]. A thorough analysis of numerous open and closed source FaaS frameworks[7] indicates the fact that in many cases there are misconceptions around characteristics and special features of these frameworks. Costs may also differ from the initial considerations due to function invocation and resource usage characteristics.

Other benefits of a FaaS platform middleware include the abstraction of the underlying resources and infrastructure services used. Each platform may span across different providers, resource types etc in a manner that is agnostic to the end user. Locally in each service resource used, the FaaS platform (through its distributed managers and agents) handles aspects such as deployment of a specific functional part of the application workflow, discovery between components (on the same provider or remote ones), enabling seamless federation and distribution of the components.

The need to offload computation exploiting various tradeoffs and capabilities as well as transform monolithic applications in order to be decomposed in smaller components, that may in turn be executed separately and on the most suitable resources is proposed by the FuncX framework[10]. Furthermore, approaches at an architectural level have started to emerge in order to scale and distribute FaaS platforms across different providers in multi-cloud or hybrid cloud scenarios. DisOpenFaaS[11] is such an approach, indicating the design principles for privately deployed FaaS platforms to perform auto-scaling of virtual machines in a distributed infrastructure, while considering the scenario where the users of such platforms are scattered around the globe. This allows the execution of requests in servers located as close as possible to the client.

Other approaches such as A3-E[12] enable applications to choose to execute parts of their logic on different infrastructures that constitute the continuum, with the goal of minimizing latency and battery consumption and maximizing availability. A3-E also exploits the FaaS model to bring computation to the continuum in the form of microservices while it selects where to execute a certain function based on the specific context and user requirements, indicating significant improvements.to latency and other aspects such as battery consumption in energy constrained devices. Cross-site orchestration has been investigated in the context of the AWS Lambda service in GlobalFlow[13]. Other frameworks based on e.g. WebAssembly[14] have emerged , in an effort to reduce container performance overheads in environments with need for low-latency response or hardware platforms with limited

resources, such as those served by edge computing environments, however their tool support is not near the maturity of platforms such as Openwhisk or OpenFaaS. Thus they can be considered as supplementary solutions but not the main driving platform solution.

Identified Gap

While FaaS frameworks portray a number of promising characteristics in terms of execution on demand, improved cost, ease of placement and inherent/direct parallelization achieved, along with a long list of open source tooling and approaches maturing, they come also with a number of shortcomings that should be addressed. Tooling availability related to **deployment and function reuse**, remains a major difficulty, in current FaaS systems[15]. Furthermore, abstractions and programming models for building non-trivial FaaS applications are limited. Only 11% of related works come from industry-only environments, indicating that while FaaS is maturing and advancing, it is time to start looking at actual industrial applications and achieved benefits[16]. **This is linked to the ease with which existing applications can be migrated to the FaaS model or combine existing functionality with FaaS enabled extensions**. Currently frameworks imply the need for full porting of the application to the FaaS model, thus the redesign of their execution model around short-lived functions, leading to potential need for extensive application rebuilding. Another challenge is the handling of state in the FaaS model. The latter primarily targets at stateless functions that do minimal I/O and communication. Frameworks such as CloudBurst[17] have tried to extend the scope to a broader range of applications and algorithms, while incorporating key-value stores for state sharing between functions. However new challenges emerge when **functions operate at a distributed and federated cloud-edge environment**, with data consistency, locality and performance being the primary one.

B. Application Design and Adaptation to the FaaS model

Visual environments have emerged in recent years as a user friendly and abstract mean of development that can speed up application development. Typically these environments are based on flow programming, based on asynchronous event driven languages such as Javascript, and offer ***palettes of readymade nodes or operators*** that incorporate the major functionalities needed. Function code is applied on the input message (triggering function execution and providing the function input data) transforming its contents based on the function logic and passing it to the next node in line. Furthermore, they encompass means of extension for these nodes as well as external repositories in which such nodes or in general flows can be stored and shared by the community. Environments such as open-source Node-RED[18] for event driven applications and KNIME[19] (mixture of open and proprietary models) for data science flows have emerged, indicating that the need for easier development and deployment of application flows is very relevant and user demanded. Typically such frameworks are designed for a specific domain (e.g. Node-RED for the IoT, KNIME for data science) and do not include an end to end rationale, that is from design to development and deployment in one single step, nor do they include ready-made patterns for exploiting the cloud model (yet they can be extended to).

In terms of major open source FaaS platforms, these typically do not come with a UI for workflow definition[9], with the exception of Apache Airflow[20] that also includes the incorporation of operators to include typical cloud services or processes. One drawback of Airflow is that these operators are typically provider specific and thus ***cannot be reused,*** while amplifying the vendor lock-in. Also, they do not include advanced and abstracted cloud design patterns. Fission[21] workflows are mainly programmatically defined while heavily linked with the Kubernetes environment. Proprietary solutions also exist with an extensive list of accompanying services such as the IBM Cloud (formerly Bluemix) environment (and Blueworks[22]) as well as Google Composer[23] (for managing Airflow related workflows) that offer integrated services design, deployment and composition, however tightly coupled with the associated vendor.

In the cloud design patterns domain, due to increasing complexity of Cloud Services, Big Vendors have promoted Pattern-Based development through new programming and deployment paradigms in order to build value added services. This development methodology has the goal of providing complex services and resources by interaction of simpler ones and can be used to define proper orchestration actions across the layers of the cloud architecture following a description of the composite service[24]. Typical cloud design patterns may include template structures and workflows such as AI training and optimization patterns[25], map/reduce types of structures, MPI based patterns, data ingestion and preprocessing flows, data transformation flows, data encryption and privacy preserving flows, load balancer structures, preconfigured messaging structures (e.g. publish/subscribe), data caching mechanisms, endpoint monitoring mechanisms, gatekeeper operations, offloading processes, queue based load levelling and workflow scaling, auto-scaling and throttling functionalities, continuous deployment patterns, node failure or transient app failure monitoring patterns etc. Some of these patterns may exist in implementation (even directly in the FaaS model or enabling transformations to thi through suitable converter frameworks[26]) however they are in need of either full parameterization (so that they can be automatically configured) and/or wrapping around the core design framework in order to achieve maximum abstraction and easy incorporation in an application design process.

Identified Gap

While helpful in the sense of each domain's usability, current design environments lack the ability to aid an application in exploiting cloud benefits through suitable and ready-made supporting structures that enhance functional and non functional aspects. Furthermore, they lack a unified and vertical approach to enable application definition, enhancement with features and creation of cloud deployment specification in an integrated manner. Either deployment specification or workflow specification are supported but not combined and do not include design patterns and functionality automatically incorporated and configured in the application graph. Many of the proposed

operators/environments are provider/platform specific and increase vendor lock-in.

Proposed Enablers for FaaS application design linked to FaaS frameworks

Cloud design and programming patterns offered as FaaS reusable components may significantly aid application adaptation in the new programming model or extension in order to embed this functionality alongside its current implementation. Furthermore, visual environments for workflow creation, given that this type is inherently linked with the functional programming concept, can significantly aid developers in their transition and application adaptation. Thus ***incorporation of such patterns in arbitrary flows*** and instantiation of them with the specific software artefacts/functions needed may be performed. Through this approach, instances supporting these patterns may be automatically included and configured in the application graph. Furthermore, a vertical approach in these tools is needed in the sense of the ability to produce directly the application deployment specification to a FaaS framework from the application design, thus achieving automated and abstracted deployment process in the underlying frameworks.

III. Resource Semantic Enrichment and usage in runtime deployment (Perspective 2)

Currently, a significantly high number of available cloud services exist in the market. This makes the task of **matchmaking** between user demands and service capabilities difficult, while differences in interfaces, implemented standards or technical features of the service complex even more the selection. Cloud patterns improve to some extent practices in services composition to ease the design and deployment of cloud-oriented applications while approaches have been developed in order to cater for differences in semantics which affect services', operations' and parameters' descriptions and to automatize the whole composition process[27]. However what is needed is a tighter link between semantics and software or platform **self-configuration processes** in order to fully exploit semantic descriptions usage during runtime and remove the human from the loop. In some specialized cases, such transformations are performed for exploiting special purpose cloud services (e.g. cloud-based FPGA's) but only for e.g. pre-trained AI models[28]. In terms of fully integrated runtime usage and management through semantics, the AffectUs framework[29] has used a combination of ontologies, semantic inference, REST services and flow based programming adapters, integrating the use of ontologies in a functional manner in the life-cycle of the application. However the use case is focused on supply chain management and more specifically on operational detection and dissemination of events across a supply chain to affected entities. Semantics are used in that context to infer on the affected entities, to identify endpoints of these entities and automatically propagate the respective events down the chain, as well as link and annotate incoming events with a specific supply chain stage.

Extensive cloud service description frameworks have been proposed in recent years[30], covering a wide variety of common characteristics including general information about a cloud service (type, deployment model, category, evaluation, service reusability, etc), functionality and operations, accessibility and authorization features, QoS capabilities (eg, security, reliability, compliance), legal issues and restrictions, pricing, resource control and visibility.

Identified Gap

While very interesting works exist in the field of cloud service descriptions, thus exploitable in the context of the Linked Data paradigm, what is yet to be accomplished is a fully integrated use of ontologies and semantics, not only for a preselection or interface composition process but also to enable functionally ***more automation*** of deployment and configuration management for general purposes.

Proposed Enablers for usage of semantics in deployment configuration and automation

First of all, relevant ontologies need to be enriched or integrated (following the Linked Data paradigm) in order to capture specific aspects of an application configuration, linked to the way this application is configured, deployed and managed. Examples of such tuples, that would enable inference on the way the application should be configured or whether it can exploit specific resource characteristics appears in Table 1. This can also be extended at the function or cloud template pattern level and can also be more easily supported during runtime due to the inherent incorporation of dockerized artefacts in FaaS environments (e.g. multiple image versions with different characteristics for a given software artefact). Other features such as runtime controlling endpoints for a given parameter can also be defined.

However, this needs to be coupled by a middleware layer that will bridge service/resource descriptions with software or function artefact descriptions. This layer needs to be responsible for inferring whether capabilities of the former can be exploited through the needs/characteristics of the latter, leading to optimized operation in the diverse and versatile computing continuum, including Edge and Fog nodes. Purposes for such inference can include usage ***during deployment and/or application adaptation***, in order to enhance both functionally and non functionally the automated and seamless application adaptation to ***diverse environments***. For example, through the portrayed relations of Table 1, semantic inference can be applied as to whether a given component is indeed multithreaded (thus selecting larger numbers of cores would be expected to improve it) as well as functional information on how to initialize and launch it. Moreover, a semantically enriched controlling logic can improve its agility. In case of a multi-threaded component, increasing the number of application threads through a controller may improve component performance up to a point where thread switching creates more overhead than benefit. When reaching this point, higher number of cores must be assigned to the resource executing it. Furthermore, one could retrieve the specific resource type in which currently the application component is running and infer whether changing a specific parameter would be expected to make a difference. Other aspects such as locality constraints (e.g. imposed by legal[35], ethical or other

requirements) can be expressed and taken under consideration during placement.

Finally, through modelled information like endpoints and their relation to QoS features (monitoring or controlling parameter endpoints), automated incorporation of controller logic can be inserted in the designed workflows and parameterized on the fly by deployment information (such as obtained IP addresses at runtime).

Table 1: Example Ontology Relations Usable for Deployment Optimization

Subject (example instance)	Predicate	Object Class (example instance)
applicationComponent (mycomponent)	startsWith	cliCommand (mycomponent.sh)
applicationComponent (mycomponent)	isMultithreadedThrough	cliOption (-threads)
cliOption (-threads)	setsParameter	Parameter (numberOfThreads)
Resource (aws.medium)	isA	ResourceType (VM)
ResourceType (VM)	hasParameter	Parameter (numberOfCores)
Parameter (numberOfCores)	isUsedBy	Parameter (numberOfThreads)
Endpoint (/setThreads)	managesParameter	Parameter (e.g. numberOfThreads)
Endpoint (/setThreads)	isA	HTTPEndpoint
Parameter (numberOfThreads)	Affects	QoSMetric (e.g. ResponseTime)
applicationComponent	hasSpecificLocalityConstraints	ProviderDC (if specific location and provider is needed)
applicationComponent	hasGenericLocalityConstraints	Country
applicationVersion (myImage)	isA	ContainerImage
ContainerImage(myImage)	isOptimizedFor	ResourceType (GPUEnabledVM)
CloudPattern (MessagingMQTTPattern)	isOptimizedFor	ResourceType (EdgeResource)
ResourceType (EdgeResourceA)	hasBenchmarkedPerformance	QoSMetric (FunctionBenchTimeToCompletion)

One aspect that needs to be stressed is that this mechanism needs also to be integrated with respective annotation capabilities of typical orchestrator systems. Thus annotating the abilities of nodes (feasible in current orchestrators through e.g. node naming following specific conventions such as _GPUEnabled) as well as creating service composition descriptions that embed such needs will bridge the gap between selection and enforcement of a given deployment scheme. Dictating the deployment scheme (as well as deployment preferences) is feasible via typical deployment specification standards available such as Docker Compose, Kubernetes etc and their metadata annotation mechanisms.

IV. Space-time Continuum evaluation and Deployment Optimization (Perspective 3)

Even though the FaaS model promises reductions of cost compared to IaaS and PaaS offerings, its billing mechanisms typically include function invocation numbers as well as execution time. However, in this context ***costs are less predictable***, especially because they are tied to function performance (as well as the provider's environment). Reports[31] have observed significant differences (up to 8.5× in performance and 67 × in cost between providers, 16.8× in performance and 67.2× in cost between programming languages) of performance and cost depending on the choice of memory allocation, FaaS provider, and programming language. The problem of the assessment of black box Cloud services performance is especially intense in FaaS environments in which server and infrastructure management is completely hidden from users by design[32]. Cloud Service Providers usually restrict the maximum size of code, memory and runtime of Cloud Functions. The aforementioned work introduces a baseline FaaS benchmarking tool, which allows users to evaluate the performance of Cloud Functions. Challenges as well as requirements for a future FaaS measurement framework include taking into account more than just runtime overhead, and include notions of cost, realistic workloads, more (open-source) platforms, and cloud integration.

Baseline approaches such as FunctionBench[33], including workloads such as ML, network and microlevel benchmarks are a step forward. Comparisons have been performed to better understand the cost vs performance trade-off between FaaS and IaaS such that cloud users can decide which approach is suitable for them. Reports[34] on comparisons between Google cloud's FaaS (Cloud Functions) and its IaaS (Compute Engine) in terms of cost and performance have indicated that FaaS can be 14% to 40% less expensive than IaaS for the same level of performance. However, performance of FaaS exhibits higher variation since CPUs allocated depend on the cloud provider.

In terms of deployment optimization and placement, numerous approaches are available[38], even from the initial Cloud era, focusing on multi-cloud service placement[36] or more recently for cloud-to-edge specific issues[37]. In many cases the size of the problem due to its NP hard nature is impossible to calculate optimally without breaking another constraint (e.g. time to reach the deployment decision that is acceptable to the user), given the range of possibilities to take under consideration, therefore more heuristic based approaches are considered.

Identified Gap

Surveys on task scheduling and offloading between cloud, edge and fog systems have been performed[39], identifying relevant methods for scheduling improvement at the global level. However there needs to be a closer link and tight collaboration between multiple functionalities and primarily resource capacity measurement/evaluation (through e.g. benchmarks that take under consideration other parameters such as function based execution, time to completion, memory used etc.) and deployment optimization approaches.

Proposed Enablers for enhanced deployment optimization across the continuum

Through the inclusion of periodic benchmarking/ evaluation details for each resource type (integrated also with the semantic descriptions of Section III), as well as relevant factors included in the optimization model, the space-time

continuum approach of [5] may be implemented, taking under consideration experienced performance and its relation to the total optimized goal (whether it is cost, performance again etc.). Furthermore, the integration of the semantic querying as an initial filter of resources (and based on the discussion in Section III), can significantly reduce the search space of an optimization algorithm and therefore potentially enable the application of globally optimal algorithms such as branch-and-bound.

The overall process of the previous sections can be summarized in the sequence presented in Figure 1.

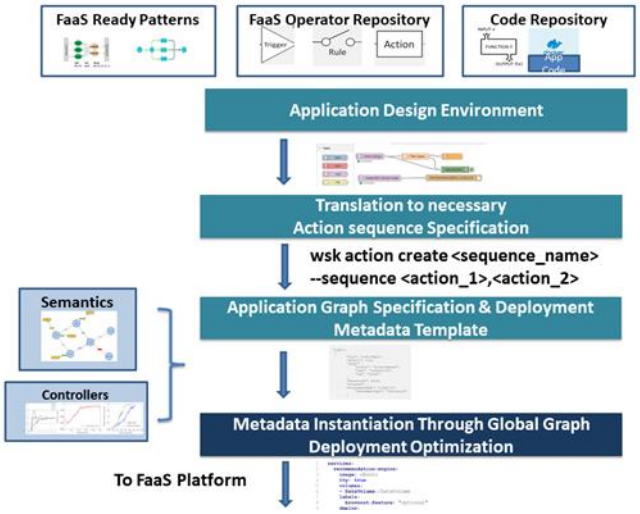

Figure 1: Process sequence starting from application design and resulting to a deployable FaaS compose file

V. GENERALIZED ARCHITECTURE LAYER AND FUNCTIONALITIES

The functionalities presented in the previous sections are included in the form of generalized architectural building blocks in Figure 2. This architecture does not assume that there is any special connection or federation between the entities (e.g. Cloud or Edge providers), only a middle managing entity like a broker/platform manager that creates and operates resources on the cloud/edge through standard APIs.

The entry point is the Application Design Layer, that includes aspects such as management of the application code basis, visual flow programming environments in which developers can design applications exploiting ready-made software patterns, FaaS operators and other reusable flows through a drag and drop manner, followed by suitable parameterization. A necessary step in this case is also the semantic description of the application parts, whether these are existing legacy components and executables or newly created functions. This step aids in the creation of semantic triples in an according knowledge base, following the respective Ontological definitions. The same step of semantic description population needs to be undertaken also by the other part of the platform, the different and diverse cloud or edge services and resources, so that the inference engine can exploit and combine the two sources of semantics (applications and resources descriptions).

Next, the main management layer (Continuum Deployment Layer) is responsible for receiving the application graphs extracted from the design environments. The optimization process is only performed after a relevant query towards the Inference Engine, that can apply functional or non functional constraints and return the subset of resources that address the requirements. These can then be mapped on the application graph and evaluated in terms of performance, cost or other goal for which information is available (e.g. energy efficiency of the selected provider if relevant certifications or other information is included in their semantic descriptions). Benchmarking or historical monitoring information for resources can be acquired from the Performance Measurement block, that aims to capture this either through monitoring of previous executions or from periodic performance evaluation tests executed on the resources (through e.g. benchmarks).

From this optimization process (applied in the Global Continuum Placement Optimizer), the final resulting deployment graph may be acquired and forwarded for deployment to a mainstream FaaS platform (e.g. Openwhisk), properly annotated via the mechanisms described in Section III in order to dictate the deployment scheme. However this mainstream FaaS platform needs to be managed by the Global FaaS layer, in the sense that the latter will have the responsibility to create resource instances of the FaaS platform (e.g. Openwhisk nodes) in each of the Cloud/Edge federation resources used and link them together in one unified virtual cluster setup. It is necessary to stress that this resource creation **does not assume any special agreement or link between the different entities** (e.g. broker entity that manages the Continuum deployment and the according Cloud/Edge providers). The Continuum Deployment layer can act like any typical customer of the latter and request resources via the typical API process. A final step is the incorporation of supporting structures to functionally link these distributed resources, like the creation of virtual networking layers and/or in-memory data services that can enforce functional aspects (e.g. visibility between the distributed FaaS resources) or more optimization aspects such as data locality and state preservation necessary for parts of the application.

At the provider-local level, once the application is deployed, the elasticity controllers (embedded in the application graph) can check the detailed function execution logs or application endpoints for performance information and upon detection of an under or over performing application part they can toggle local resources given by the local CSP through the typical structures available currently (e.g. through API calls). If despite these efforts this application part does not suitably adapt to the desired QoS levels, then the problem can be redirected to the global optimization plane for a new deployment optimization. Through this joint handling of global and local level control, applications can exploit the FaaS model and better adapt to the distributed continuum, abstracting from the practical aspects of their execution and management.

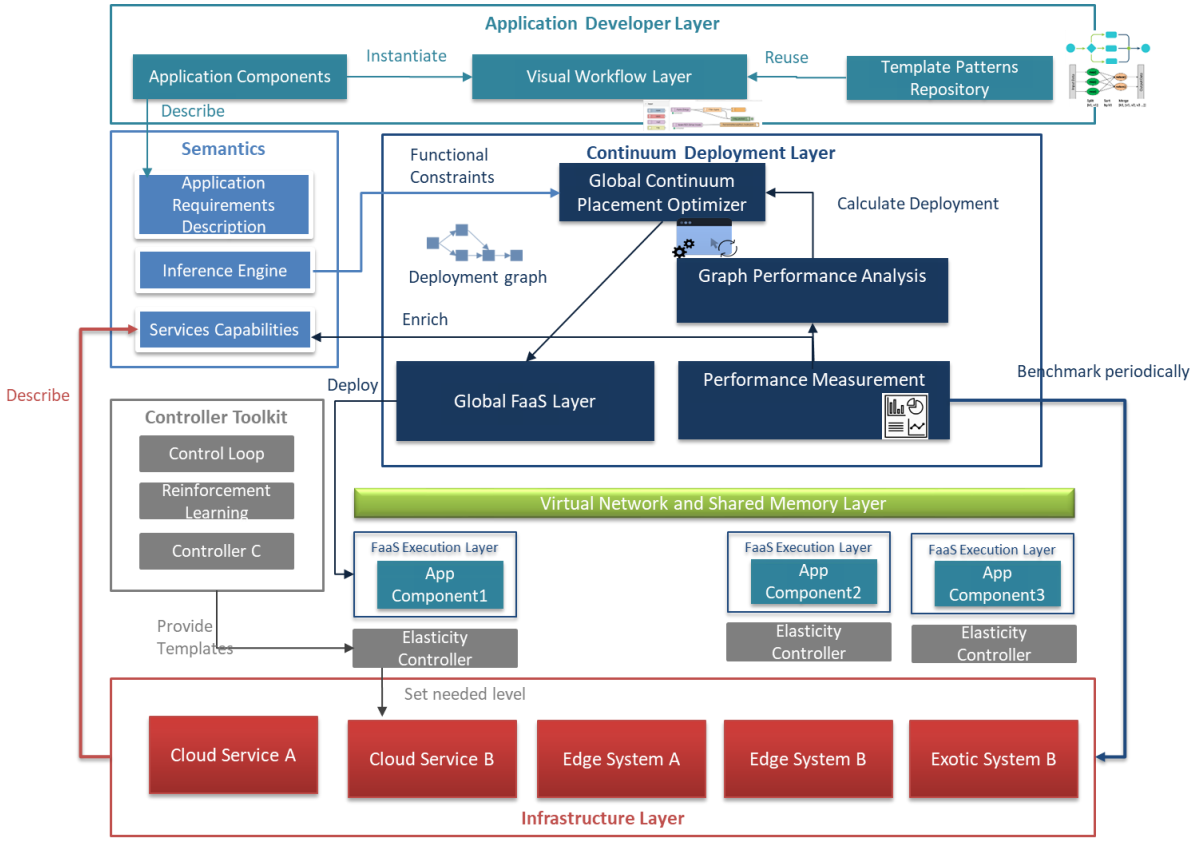

Figure 2: Generalized Architectural Building Blocks

VI. CONCLUSIONS

As a conclusion, FaaS presents a set of characteristics (e.g. granular execution, flexible deployment structures, supporting platform tools) that enable its usage across diverse devices as well as distributed environments. However in order for this upcoming computing model to reach its full potential, a number of additions need to be performed to enable easier application adaptation, harvesting of its benefits as well as seamless management across diverse and distributed locations.

The work in this paper proposed such additions from application design to deployment and operation. Initially the application design and implementation process, supported by implementation templates that cover typical cloud oriented functionalities directly in the FaaS model as well as visual flow programming environments, will aid in abstracting FaaS migration processes or application functionality extensions. To this end, the polyglot ability of FaaS frameworks is a major strength of the domain, as well as their inherent event driven nature.

Following, extensions in terms of semantic descriptions and their incorporation during the runtime selection and configuration is another key enabler, that aids in automatically adapting to the diverse environments and optimizing application setup and management. Optimization of deployment selection can significantly benefit from applying such semantic descriptions as well as runtime evaluation of Cloud/Edge resources.

The overall analysis has led to a generalized architectural approach, that can aid in addressing functional and non functional requirements of complex applications deployed over dynamic and volatile environments, with enhanced abstraction in terms of resource management and application adaptation. The proposed architecture does not imply any specific relation or exposure between providers and is regulated by a brokering entity that has the goal of managing the middleware layer.

ACKNOWLEDGMENT

The research leading to the results presented in this paper has received funding from the European Union's funded Project PHYSICS under grant agreement no 101017047.

REFERENCES

[1]. Flexera State of the Cloud Report, 2020, available at: https://info.flexera.com/SLO-CM-REPORT-State-of-the-Cloud-2020

[2]. Ana Juan Ferrer, Robert Woitsch, Kyriakos Kritikos, George Kousiouris, Fotis Aisopos, David Garcia, Pierluigi Plebani, and Xavi Masip. 2017. Future Cloud Research Roadmap, FP9 Cluster Inputs. Technical Report. Retrieved from https://drive.google.com/file/d/0B4hHTKjZDMXGSGxoYnh4eXhURzA/view.

[3]. Bittencourt, L., Immich, R., Sakellariou, R., Fonseca, N., Madeira, E., Curado, M., Villas, L., DaSilva, L., Lee, C. and Rana, O., 2018. The internet of things, fog and cloud continuum: Integration and challenges. Internet of Things, 3, pp.134-155.

[4]. Lynn, T., Mooney, J.G., Domaschka, J. and Ellis, K.A., 2020. Managing distributed cloud applications and infrastructure: A self-optimising approach.

[5]. I. Foster, "Coding the Continuum," 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS), Rio de Janeiro, Brazil, 2019, pp. 1-1, doi: 10.1109/IPDPS.2019.00011.

[6]. Kousiouris, G., Cucinotta, T. and Varvarigou, T., 2011. The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks. Journal of Systems and Software, 84(8), pp.1270-1291

[7]. Chen, Y., He, J., Zhang, X., Hao, C. and Chen, D., 2019, February. Cloud-DNN: An open framework for mapping DNN models to cloud

FPGAs. In Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (pp. 73-82).
[8]. Kuhlenkamp, J., Werner, S. and Tai, S., 2020, April. The Ifs and Buts of Less is More: A Serverless Computing Reality Check. In 2020 IEEE International Conference on Cloud Engineering (IC2E) (pp. 154-161). IEEE.
[9]. Van Eyk, E., Grohmann, J., Eismann, S., Bauer, A., Versluis, L., Toader, L., Schmitt, N., Herbst, N., Abad, C. and Iosup, A., 2019. The SPEC-RG Reference Architecture for FaaS: From Microservices and Containers to Serverless Platforms. IEEE Internet Computing.
[10]. Chard, R., Babuji, Y., Li, Z., Skluzacek, T., Woodard, A., Blaiszik, B., Foster, I. and Chard, K., 2020. funcX: A Federated Function Serving Fabric for Science. arXiv preprint arXiv:2005.04215.
[11]. S Vieira, L., Vasconcelos, A., Batista, Í., Silva, R.A. and Brasileiro, F., 2019, September. DisOpenFaaS: A Distributed Function-as-a-Service Platform. In Anais Estendidos do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (pp. 33-40). SBC
[12]. Baresi, L., Mendonça, D.F., Garriga, M., Guinea, S. and Quattrocchi, G., 2019. A unified model for the mobile-edge-cloud continuum. ACM Transactions on Internet Technology (TOIT), 19(2), pp.1-21
[13]. Zheng, G. and Peng, Y., 2019, July. GlobalFlow: A Cross-Region Orchestration Service for Serverless Computing Services. In 2019 IEEE 12th International Conference on Cloud Computing (CLOUD) (pp. 508-510). IEEE
[14]. Hall, A. and Ramachandran, U., 2019, April. An execution model for serverless functions at the edge. In Proceedings of the International Conference on Internet of Things Design and Implementation (pp. 225-236).
[15]. Philipp Leitner, Erik Wittern, Josef Spillner, and Waldemar Hummer. A mixed-method empirical study of Function-as-a-Service software development in industrial practice. Journal of Systems and Software, 146:340–359, March 2019.
[16]. Yussupov, V., Breitenbücher, U., Leymann, F. and Wurster, M., 2019, December. A Systematic Mapping Study on Engineering Function-as-a-Service Platforms and Tools. In Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing (pp. 229-240).
[17]. Sreekanti, V., Lin, C.W.X.C., Faleiro, J.M., Gonzalez, J.E., Hellerstein, J.M. and Tumanov, A., 2020. Cloudburst: Stateful Functions-as-a-Service. arXiv preprint arXiv:2001.04592.
[18]. Node-RED, Low-code programming for event-driven applications, available at: https://nodered.org/
[19]. KNIME https://www.knime.com/
[20]. Apache Airflow workflow definition and monitoring tool, available at: https://airflow.apache.org/
[21]. Fission Open source Kubernetes-native Serverless framework, available at: https://fission.io/
[22]. IBM Blueworks, Available at: https://www.ibm.com/products/blueworkslive
[23]. Google Composer offering, available at: https://cloud.google.com/composer
[24]. Amato, F. and Moscato, F., 2017. Exploiting cloud and workflow patterns for the analysis of composite cloud services. Future Generation Computer Systems, 67, pp.255-265
[25]. Giampa, P. and Dibitonto, M., 2020. MIP An AI Distributed Architectural Model to Introduce Cognitive computing capabilities in Cyber Physical Systems (CPS). arXiv preprint arXiv:2003.13174
[26]. Carvalho, L. and de Araújo, A.P.F., 2019. Framework Node2FaaS: Automatic NodeJS Application Converter for Function as a Service. In Proceedings of the 9th International Conference on Cloud Computing and Services Science (Vol. 1, pp. 271-278).
[27]. Di Martino, B., Cretella, G. and Esposito, A., 2017. Cloud services composition through cloud patterns: a semantic-based approach. Soft Computing, 21(16), pp.4557-4570
[28]. Chen, Y., He, J., Zhang, X., Hao, C. and Chen, D., 2019, February. Cloud-DNN: An open framework for mapping DNN models to cloud FPGAs. In Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (pp. 73-82).
[29]. Kousiouris, G., Tsarsitalidis, S., Psomakelis, E., Koloniaris, S., Bardaki, C., Tserpes, K., Nikolaidou, M. and Anagnostopoulos, D., 2019. A microservice-based framework for integrating IoT management platforms, semantic and AI services for supply chain management. ICT Express, 5(2), pp.141-145.
[30]. S. Ghazouani, H. Mezni, and Y. Slimani, “Bringing semantics to multicloud service compositions,” Softw: Pract Exper, vol. 50, no. 4, pp. 447–469, Apr. 2020, doi: 10.1002/spe.2789.
[31]. Bortolini, D. and Obelheiro, R.R., 2019, November. Investigating Performance and Cost in Function-as-a-Service Platforms. In International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (pp. 174-185). Springer, Cham.
[32]. Pellegrini, R., Ivkic, I. and Tauber, M., 2019. Function-as-a-Service Benchmarking Framework. arXiv preprint arXiv:1905.11707.
[33]. Kim, J. and Lee, K., 2019, July. Functionbench: A suite of workloads for serverless cloud function service. In 2019 IEEE 12th International Conference on Cloud Computing (CLOUD) (pp. 502-504). IEEE.
[34]. Malla, S. and Christensen, K., 2020. HPC in the cloud: Performance comparison of function as a service (FaaS) vs infrastructure as a service (IaaS). Internet Technology Letters, 3(1), p.e137
[35]. Barnitzke, B., Ziegler, W., Vafiadis, G., Nair, S., Kousiouris, G., Corrales, M., Wäldrich, O., Forgó, N. and Varvarigou, T., 2011. Legal restraints and security requirements on personal data and their technical implementation in clouds. In Workshop for E-contracting for Clouds, eChallenges (pp. 51-55).
[36]. Ferrer, A.J., HernáNdez, F., Tordsson, J., Elmroth, E., Ali-Eldin, A., Zsigri, C., Sirvent, R., Guitart, J., Badia, R.M., Djemame, K. and Ziegler, W., 2012. OPTIMIS: A holistic approach to cloud service provisioning. Future Generation Computer Systems, 28(1), pp.66-77.
[37]. S. Meixner, D. Schall, F. Li, V. Karagiannis, S. Schulte and K. Plakidas, "Automatic Application Placement and Adaptation in Cloud-Edge Environments," 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Zaragoza, Spain, 2019, pp. 1001-1008, doi: 10.1109/ETFA.2019.8869256.
[38]. Cao, B., Zhang, L., Li, Y., Feng, D. and Cao, W., 2019. Intelligent offloading in multi-access edge computing: A state-of-the-art review and framework. IEEE Communications Magazine, 57(3), pp.56-62.
[39]. Sindhu, V. and Prakash, M., 2019, November. A Survey on Task Scheduling and Resource Allocation Methods in Fog Based IoT Applications. In International Conference on Communication and Intelligent Systems (pp. 89-97). Springer, Singapore