ISSN 2347 - 3983
Volume 8. No. 9, September 2020
International Journal of Emerging Trends in Engineering Research
Available Online at http://www.warse.org/IJETER/static/pdf/file/ijeter217892020.pdf
https://doi.org/10.30534/ijeter/2020/217892020

Devops and Microservices Based Internet of Things Meta-Model

Badr El Khalyly, Abdessamad Belangour, Allae Erraissi, and Mouad Banane
Laboratory of Information Technology and Modeling, Hassan II University, Faculty of sciences Ben M'Sik, Casablanca, Morocco
Emails: badrelkhalyly92@gmail.com, belangour@gmail.com, erraissi.allae@gmail.com, mouadbanane@gmail.com

ABSTRACT

The ecosystem of the Internet of Things is a set of physical devices such as sensors and actuators. It includes a set of servers and gateways that provide connectivity. These devices can be installed at three levels: the edge level, the fog level, and the cloud level. Applications that monitor things and collect data from sensors are deployed at the edge, fog, and cloud levels. These applications can be containerized and deployed in different devices by Docker, which takes advantage of its advantage to create executable containers that are isolated from each other. This paper presents a generic Meta-model of the Internet of Things ecosystem based on microservices and supported by Docker as a containerization tool, Ansible as a monitoring tool and Kubernetes as scaling tool. This Meta-model allows us to generate a system of connected objects that can be deployed on three levels: Fog, Edge, Cloud. This Meta-model can be used in different types of domains such as Smart Home - Smart City - Smart Vehicle - Smart Health - Smart Farm - Smart Factory. The Meta-model proposed in this article is a fusion between 5 Meta-models: Internet of Things - Microservices - Ansible – Docker - Kubernetes.

Keywords: Model Driven Engineering, Meta-modeling, Microservices, Internet of Things, Devops, Docker, Ansible, Kubernetes.

1. INTRODUCTION

Nowadays, developers and designers are adopting several technological trends in the field of the Internet of Things [1]. These technologies include microservices [2] and Devops [3]. Microservices are an architectural style in which the software system is built with standalone components. These components are separated from each other in terms of business functionality and have limited granularity [4].
Devops is a fusion of two words: Development and operation. Devops comprises the work of developers in parallel with the work of integrators. It allows developers to control the entire chain of integration and continuous deployment from development, through pushing code to the built environment, containerization and orchestration of deployed instances. Devops culture tools are at the service of microservices. Among these Devops tools, which participate in the assurance of the deployment and continuous integration chain, we mention the following: Docker [5], Ansible [6] and Kubernetes [7].
Docker makes it possible to containerize the microservices developed in order to ensure their portability and load balancing between several microservice instances [5].
Ansible is an open-source DevOps [8] tool that can help the company in configuration management, deployment, provisioning, etc. It is simple to deploy; it uses SSH technology to communicate between servers [9]. It uses the playbook to describe automation tasks. Ansible participates in the continuous deployment of the Docker Containers realized. Playbook is a configuration in which we cite the name of docker image to be deployed, removed, or updated.
Model-Driven Engineering allows systems to be generated according to the designer's needs [10,11,12,13]. The objective of this article is to propose a PIM model [14] that gathers all the concepts that allow us to build an Internet of Things system based on microservices and supported by the three tools Devops Docker, Ansible and Kubernetes. Namely, a Meta-model is presented for each of the following ecosystems: IoT [15] – Microservices [2] - Docker [5]- Ansible [6]. Each ecosystem is modeled as a package and then these packages are linked together to form a global Meta-model of the Internet of Things system based on microservices and supported by the three tools Devops Docker, Ansible and Kubernetes.
The deployment of containers in a microcontroller must take into account the running performance of that microcontroller. Among the microcontroller performance indices is scalability. Kubernetes offers a solution for self-scalability of Docker containers in a microcontroller. According to kubernetes.io Kubernetes is an extensible and portable open-source platform for managing workloads and containerized services. It supports both declarative configuration writing and automation. It is a large and rapidly growing ecosystem. Kubernetes services, support and tools are widely available. Kubernetes has a number of features. Kubernetes provides a container-centric management environment. It orchestrates

the computing, networking and storage infrastructure on the users' workloads.

This Meta-model can be used in different domains [16,17,18]. The fourth section talks about improving the Internet of Things Meta-model. The fifth section talks about a Meta-model enhancement of microservices. The sixth section talks about a proposed Meta-model for the Docker containerization system. The seventh section discusses a proposed Meta-model for the Ansible monitoring system. The eighth section talks about a proposed Meta-model for Kubernetes. The nineth section merges these Meta-models into a Meta-model for a microservice-based Internet of Things system supported by the three tools Devops Docker, Ansible and Kubernetes.

2. RELATED WORK

It is in the field of software engineering that the problem of the expression of executability in meta-models has been studied in depth, thanks in particular to the development of the IDM approach. Indeed, since IDM is fundamentally based on the extensive use of models in all phases of software development, the question quickly arose of how to execute a model and how to express the executable semantics of that -this. In [19], a first reflection on the link between meta-model and the expression of the executability of models was made on Petri nets. The authors supplement the static meta-model which describes the fixed structures in a model (arcs and transitions in a Petri net), by a dynamic meta-model that describes the data structures necessary during the execution of an instance of this model (markings and token movements). The authors recognize however that this addition is not sufficient to express all the executability, because the formalism used (UML class diagram) does not itself have executable semantics, and they thus call for the creation of an Executable UML. And it is probably as a result of these preliminary reflections on the expression of executability that the Kermeta language was developed [20]. Kermeta complements UML meta-diagrams in the form of a directly operational meta-specification. Combined with the principle of dynamic meta-classes introduced previously [19], this approach makes it possible to construct a complete meta-specification for a model. The Kermeta language has so far been used to construct a complete and executable description of a simple model, that of the finite state machine [21], and tested in the context of embedded software systems to specify meta-models of UML 2.0 in Kermeta [22]. Other work in the software engineering and IDM community has focused on engineering process models (called software process models), given the importance of describing, controlling, and automating the procedures with which software systems are built. An important work in this register is that around UML4SPM, which defines an engineering process modeling language which is based on UML and close to the OMG SPEM model [23], [24]. Several experiments are made to specify the semantics and express the executability of this language: first, using the BPEL business process execution language [25], and then using the Kermeta meta-specification language [26]. For these two approaches, the issue of interaction with the outside world (the user or other systems) is underlined as that which distinguishes the two approaches. Despite several advantages (notably the existence of reliable technical systems for the execution of BPEL models), the use of BPEL is not considered satisfactory because of the absence of concepts to take into account the interaction with the user. Other research, more oriented towards the comparative study of approaches, completes this work around the executability of a process model [27, 28].

In the field of IS engineering and that of method engineering, in particular, few works to our knowledge have addressed the question of the explicit expression of executability in a process meta-model. The definition of a method being the combination of a product meta-model and a process meta-model, the product specification is historically the oldest. [29] Regarding the process, it is the approach by assembling methodological components that are the most used. In [30], the language MEL1, a formal language for the specification of methods, is proposed. Apart from the structural specification of the components, the process aspect is described in MEL as formal operators whose semantics are guaranteed by the underlying mathematical notation.

This approach by assembling methodological components currently remains predominant [31], however, we wonder about the models used to formalize the approach, to specify the content of a methodological component, and to express the assembly process [32]. Finally, a new research perspective in this sense is that of defining the methodological components as being services [33].

A methodological component is directly executable since it is a service that is written with SOA standards, while the composition of services is expressed informally at a high level of abstraction using intentional Map. Banana et al. [34] proposed a metamodel of Hive [35] and Spark [36], as well as another metamodel [37] of MongoDB query language. The IoT is responsible for the collection and / or creation of a large volume of data. This enormous volume of data, known by the term "Big Data" [38,39,57,58], allows, on the one hand, to have an incredible wealth in terms of information allowing the offer of advanced services. On the other hand, this volume of data creates new challenges to be considered such as securing, processing, and real-time accessibility of this data [40,41].

To end this overview, we must mention the metalisms and meta-modeling languages offered tools and meta-CASE type environment. MetaEdit is a well-known and popular tool [42], it allows you to specify a static meta-model with the GOPRR2 formalism, and immediately generate a graphic editor for the model [43]. It is intended for the creation and tooling of new domain-specific languages [44]. As far as the "processing" and "behavior" perspectives are concerned, they are relegated to the code generation phase where a scripting language (called Merle) makes it possible to browse and manipulate the instances of the model and to generate instructions in n ' any target language (HTML, XML, C ++, Java, etc.). The executable semantics is thus expressed by the transformation

of the static and non-productive structures of the model into a set of software instructions (or components) in another language or another model whose executability is known and supported by a platform. target shape. While the definition of the meta-model is done in a declarative way with a graphical interface, the executability is expressed operationally with a programming type interface. This is the main drawback of MetaEdit.Another meta-modeling formalism supported by a meta-CASE is the ConceptBase environment [45] built around the Telos model and language [46]. Built with the declarative logic language Datalog, ConceptBase is an extremely powerful meta-modeling environment which allows you to specify any number of abstraction levels and to express constraints and requests on several of these levels (and not on one level as in OCL). The specification is entered verbatim, but the content of the repository is displayed graphically at the request of the user. In terms of "processing" and "behavior" perspectives, ConceptBase has introduced in its most recent versions, rules of the Event-Condition-Action (ECA) form to express the dynamics of a meta-model. An illustration is proposed in [47] with the rules for the execution of a Petri net.

The Meta-model of an Internet of Things system based on microservices and supported by the three tools Devops Docker, Ansible and Kubernetes is a fusion of four Meta-models: Internet of Things, Microservice, Docker, Ansible, Kubernetes.

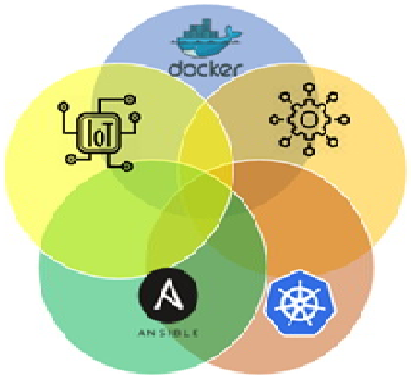

Figure 1: Architecture of the Enhanced Fuzzy Resolution Mechanism using ANFIS.

In the literature, there are proposals for Meta-models for microservice ecosystems and the Internet of Things. The authors in [48] proposed a Meta-model for the ecosystem of connected objects. Authors in [49] proposed a PIM Level for the microservice ecosystem.

The microservices in this model are part of the overall microservice architecture. They are divided into two types: functional microservices and infrastructure microservices. Microservices use load balancers and are deployed on containers. Microservices use service interfaces that are published at endpoints. Microservices depend on service operations. Data caching, storage, and asynchronous buses are part of service operations.

In the literature, there is a lack of Meta-models for containerization Docker, Ansible and Kubernetes.

The contributions of this article reside in:

- Improving the Meta-model of connected objects.
- Improving the Meta-model of microservices and proposing the appropriate type of microservice for IoT applications.
- Meta-model proposal for Docker.
- Meta-model proposal for Ansible.
- Meta-model proposal for Kubernetes.

3. CONSTRUCTING META-MODEL

First, we produce a reference framework of a given subject; it contains important elements that should be included in the Meta-model. Then a second step is to gather existing models of the subject, note that the more models we gather the best is the quality of our Meta-model. Then we analyze the concepts using matching technics. Them, finally we optimize the concepts in one Meta-model containing an aggregation to all models.

The following step must be executed to build a final Meta-model:

- Step 0: Reference Framework Definition

The reference framework definition in our case has to take into consideration 4 fields of search.

For each field of search, we define a set of rules & concepts that are interconnected to get as results in the relationship (Ri) and concepts (Cj).

- Step 1: Concepts Gathering

Each ecosystem has its PIM that has its specificities that are useful during implementation. PIMi is modeling of an ecosystem i.

$i \in$ {Docker, IoT, Ansible, Kubernetes, Microservices}

Number equations consecutively. Equation numbers, within parentheses, are to position flush right, as in (1), using a right tab stop.

- Step 2: Meta-model Construction

The Meta-model of an Internet of Things that is based on microservice and supported by Docker and Ansible is an intersection between the following PIM: PIM(IoT), PIM(Microservices), PIM(Docker), PIM(Ansible), PIM(Kubernetes).

4. META-MODEL OF THE INTERNET OF THINGS ECOSYSTEM

The Meta-model of an Internet of Things that is based on microservice and supported by Docker and Ansible is an intersection between the following PIM: PIM(IoT), PIM(Microservices), PIM(Docker), PIM(Ansible), PIM(Kubernetes).

This Meta-model is an improvement of the cited Meta-model in the state of the art [51].

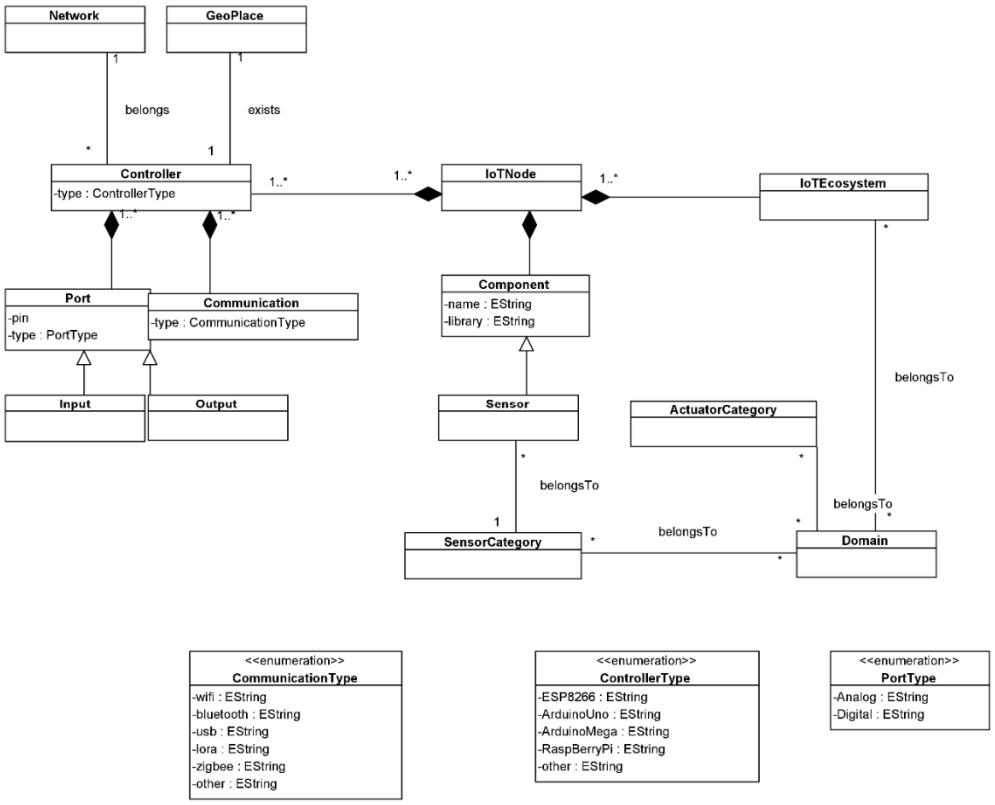

Figure 2: Meta-Model of Internet of Things Ecosystem

We have added the following classes:

- Network: The network to which the controller belongs manages the connected objects.
- GeoPlace: These GPS geographic coordinates determine the location of a controller in an ecosystem in order to locate them. This information is important in some ecosystems such as those belonging to the category: Smart Logistic - Smart Farm.
- SensorCategory: determines the category a sensor belongs to such as temperature, humidity, distance, etc.
- ActuatorCategory: determines the category an actuator belongs to such as motors, light bulbs, etc.
- Domain: Determines the domain to which the sensor and actuator categories belong. As an example, we have sensors that belong to the smart farm domain such as temperature, humidity, and acidity sensors. Also, some actuators belong to the smart vehicle domain such as engines.

5. META-MODEL OF MICROSERVICES ECOSYSTEM

These microservices are programs that are deployed on machines belonging to the ecosystem [50].

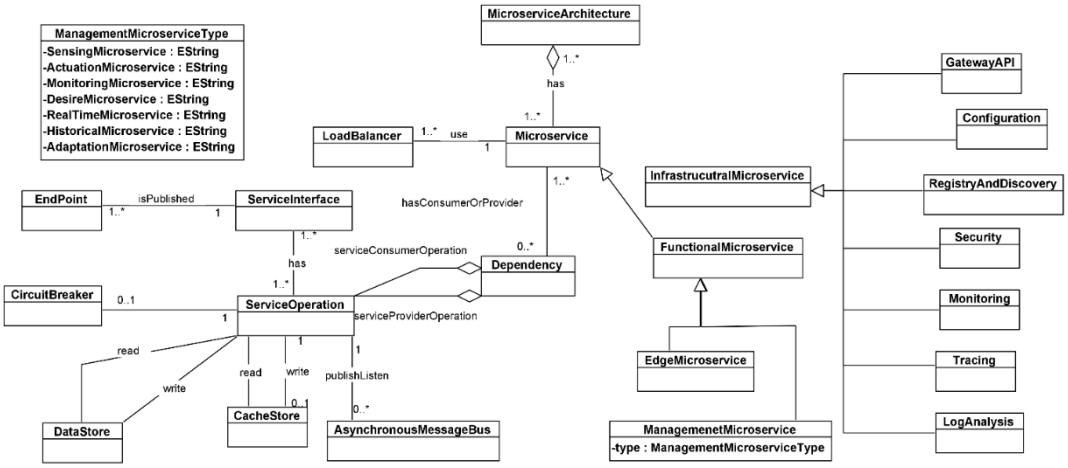

Figure 3: Meta-Model of Microservice Ecosystem

We were inspired by the meta-model cited in the state of the art. The components we used are as follows: Microservice is part of MicroserviceArchitecture by being the main component.
Microservices are divided into two types: InfrastructuralMicroservice and FunctionalMicroservice. Among the InfrastructuralMicroservice there are API Gateway, Configuration, Registry and Discovery, Security, Monitoring, Tracing, Log Analysis. Microservices expose Service Interfaces that are published in EndPoint.
We have proposed types of microservices appropriate to the ecosystems of connected objects called FunctionalMicroservice they are divided into 3 categories:
ActuationMicroservice which is responsible for sending commands to the actuator. SensingMicroservice which are responsible for sensing physical quantities from sensors. CompositeMicroservice are microservices that retrieve information from the SensingMicroservice and perform processing in order to make decisions to send commands to the ActuationMicroservice.

6. META-MODEL OF DOCKER ECOSYSTEM

Docker provides containerization services and it is based on Linux containers. Docker provides a standard runtime across Docker Engine. It allows us to build Image format. When the image is run in such an environment like Cloud, Fog, Edge it is called container. Microservices are organized in the form of containers are dedicated to being deployed on machines belonging to the Internet of Things ecosystem.
The Meta-model of the Docker ecosystem is organized as follows:
DockerDaemon is the docker engine that manages a set of ControlGroup. Docker's ControlGroup manages DockerObject. DockerObjects are divided into two types, Image and Container. A Container is an instantiation of an Image. An Image is stored in a DockerRegistry which can be private or public. An Image has a set of ImageVersion versions. The DockerFile is a file that is responsible for creating the DockerImage in the DockerRegistry. DockerCompose is a file that is composed of several services, each service representing a DockerImage. DockerCompose is responsible for deploying an application composed of several DockerImages. The NameSpace are used to provide isolation to the Container. Docker creates a set of NameSpace for each Container.

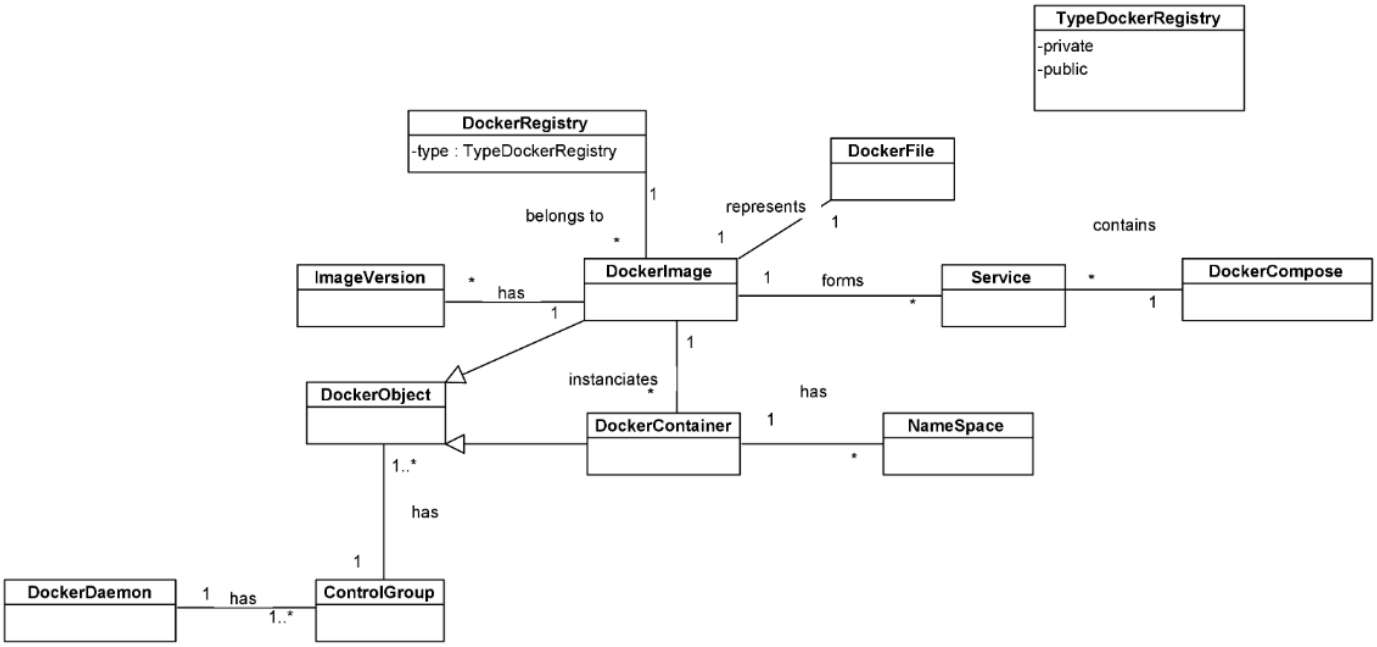

Figure 4: Meta-Model of Docker Ecosystem

6. META-MODEL OF ANSIBLE ECOSYSTEM

Ansible is an open-source DevOps tool that can help the enterprise in configuration management, deployment, provisioning, etc. It is simple to deploy; it uses SSH technology to communicate between servers. It uses the playbook to describe automation tasks, and the playbook uses a very simple language, YAML [52].Ansible is responsible for deploying the Docker Containers on the machines. Its Meta-model is organized as follows:Ansible's architecture is a Master-Slave architecture. The AnsibleMaster is the machine that performs the administration of other machines that are no other than microcontrollers. The machines we want to monitor are stored in a file called Inventory. The AnsibleMaster generates an SSHKey that it shares with the other "microcontroller" machines to allow resource sharing with these machines. Resource sharing and container deployment is done through YAML files called PlayBook [53].

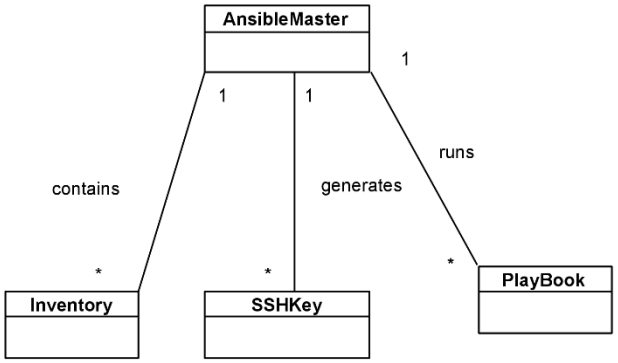

Figure 5: Meta-Model of Ansible Ecosystem

7. META-MODEL OF KUBERNETES ECOSYSTEM

Depending on Kubernetes.io, Kubernetes cluster is an orchestrated set of nodes. Kubernetes offers two types of node is Master Node, Slave Node. The Master Node represents the server that performs the deployment of the Pods. A Pod is the basic execution unit of a Kubernetes application - the smallest and simplest unit of the Kubernetes object model that you create or deploy. A Pod represents the processes running on your cluster. A Pod encapsulates an application Container (or, in some cases, multiple Containers), storage resources, a single network IP, and the options that govern how the Container(s) should operate. A Pod represents a single deployment unit is a single instance of an application in Kubernetes, which may consist of either a single container or a small number of closely coupled and resource-sharing containers.

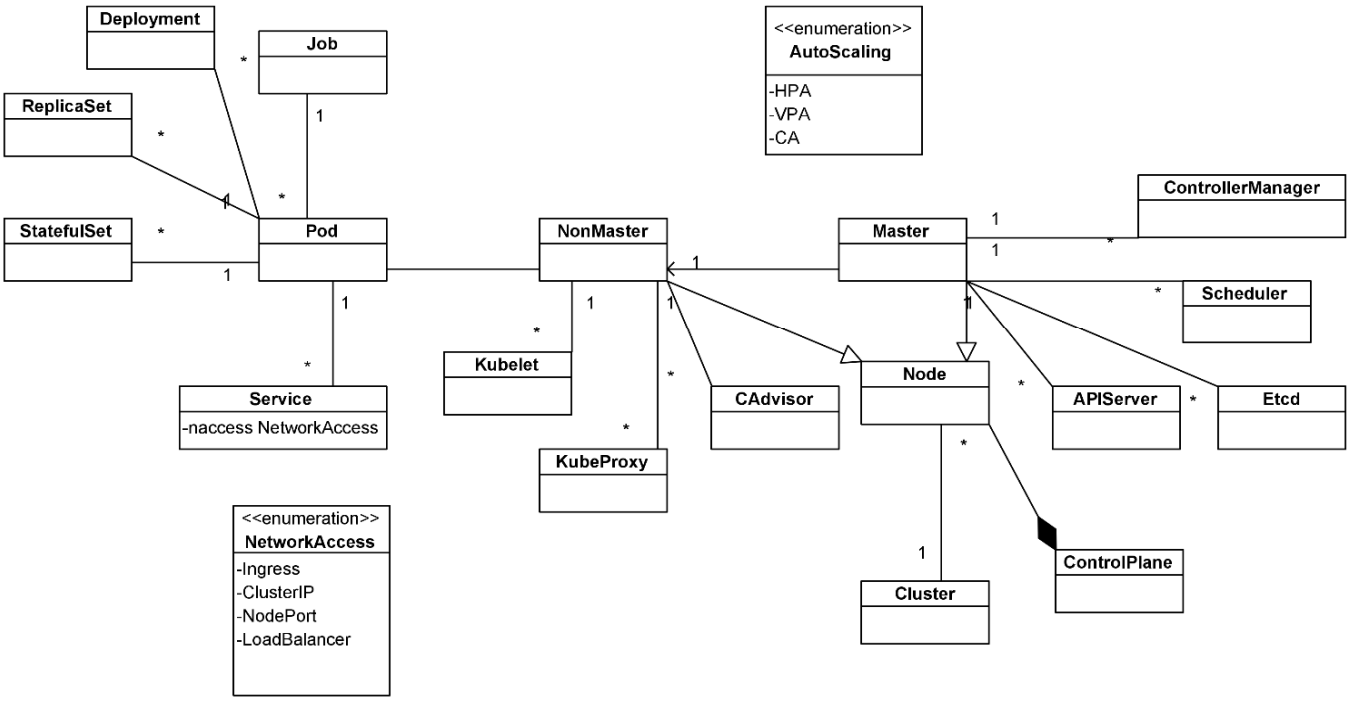

Figure 6: Meta-Model of Kubernetes Ecosystem

In particular, the Master Node is the server that performs the deployments on the microcontrollers. The microcontrollers are the Slave Nodes on which the Docker containers are deployed in the form of Pods.

Kubernetes is responsible for the orchestration of the Docker containers.

Cluster is a set of machines. Node is the machine on which the Container Pods run. NonMaster is the node on which the Pods are deployed. Master is a node that controls the NonMaster Nodes. Pod is a group of containers deployed together.

Kubectl is a command line application to interact with Kubernetes. Etcd is a lightweight persistent distributed storage unit. APIServer is a REST API for communication with internal and external components. Scheduler is a Scheduler that allows to select which Node should run a Pod. ControllerManager is Process in which the main Kubernetes controllers run. Kubelet is Responsible for the execution status of the Node. Kube-proxy is Responsible for routing traffic to the appropriate Container based on the IP address and port number of the incoming request. CAdvisor is Agent that monitors and recovers resource consumption and performance data such as CPU, memory, disk and network usage of the Node's Containers. ReplicaSet aims to maintain a

stable set of Pods at a given time. This object is often used to guarantee the availability of an identical number of Pods. Deployment provides declarative updates for Pods and ReplicaSets. Service provides network access to a set of Pods in Kubernetes. StatefulSet is the workload API object used to manage stateful applications and allows to manage the deployment and scaling of a set of Pods, and provides guarantees on the order and uniqueness of these pods. The scaling of a set of Pods is typed according to 3 categories Horizontal Pod Autoscaling - Vertical Pod autoscaling - Cluster Autoscaling.

8. GLOBAL META-MODEL

The principle of this Meta-model stipulates that the microservice is containerized using Docker (fig.7). The Docker container is made in the form of a Pod in order to scale it up using Kubernetes (fig.8). The Kubernetes Pods are started in the Kubernetes Node which forms a cluster. The cluster is created and managed by Ansible's playbooks (fig.9). The AnsibleMaster machine monitors the existing microcontrollers in the IoT ecosystem by sharing SSH keys and launching playbooks to deploy the Kubernetes Pods (fig10).

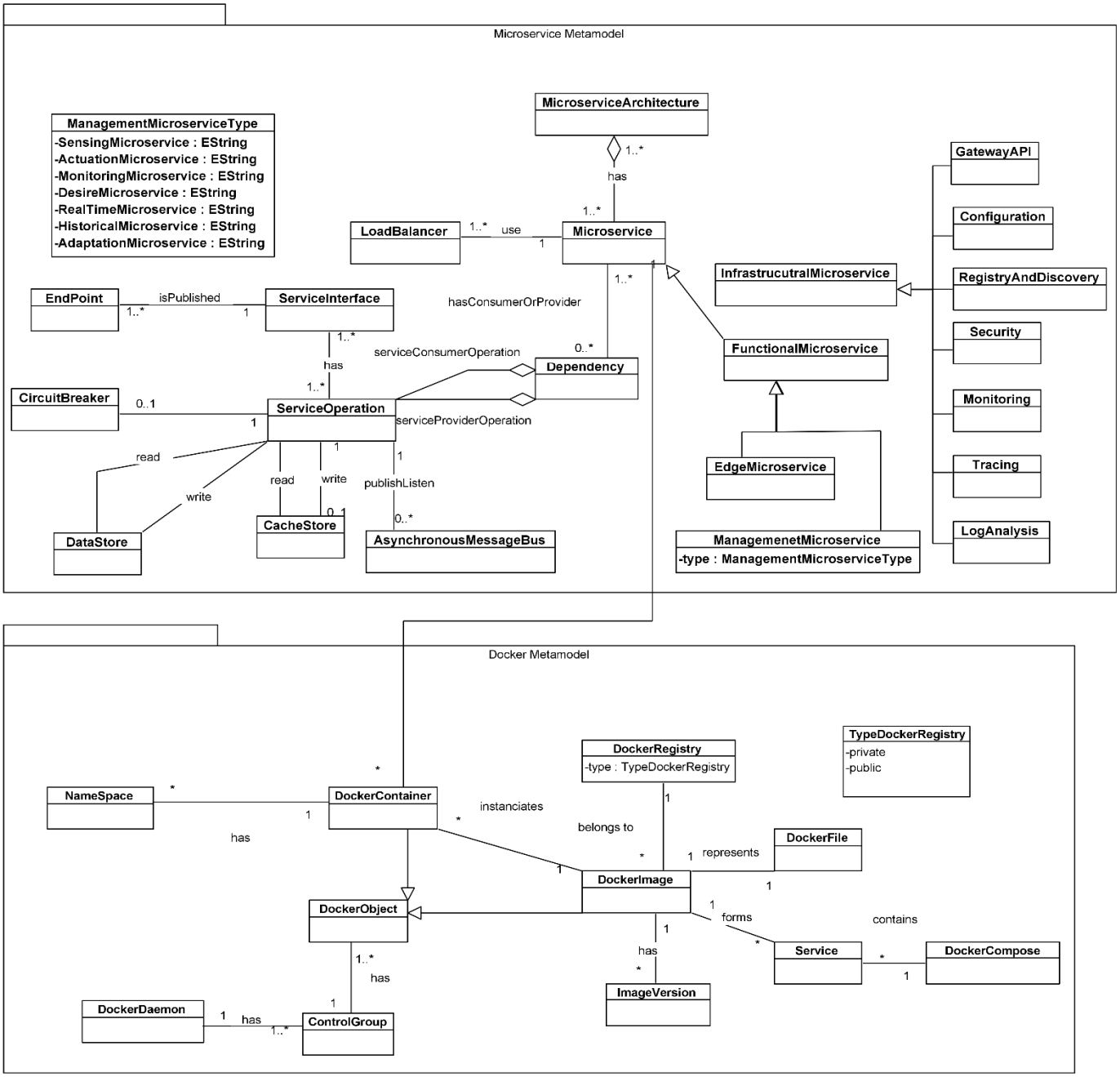

Figure 7: Communication between Microservice and Docker Ecosystem

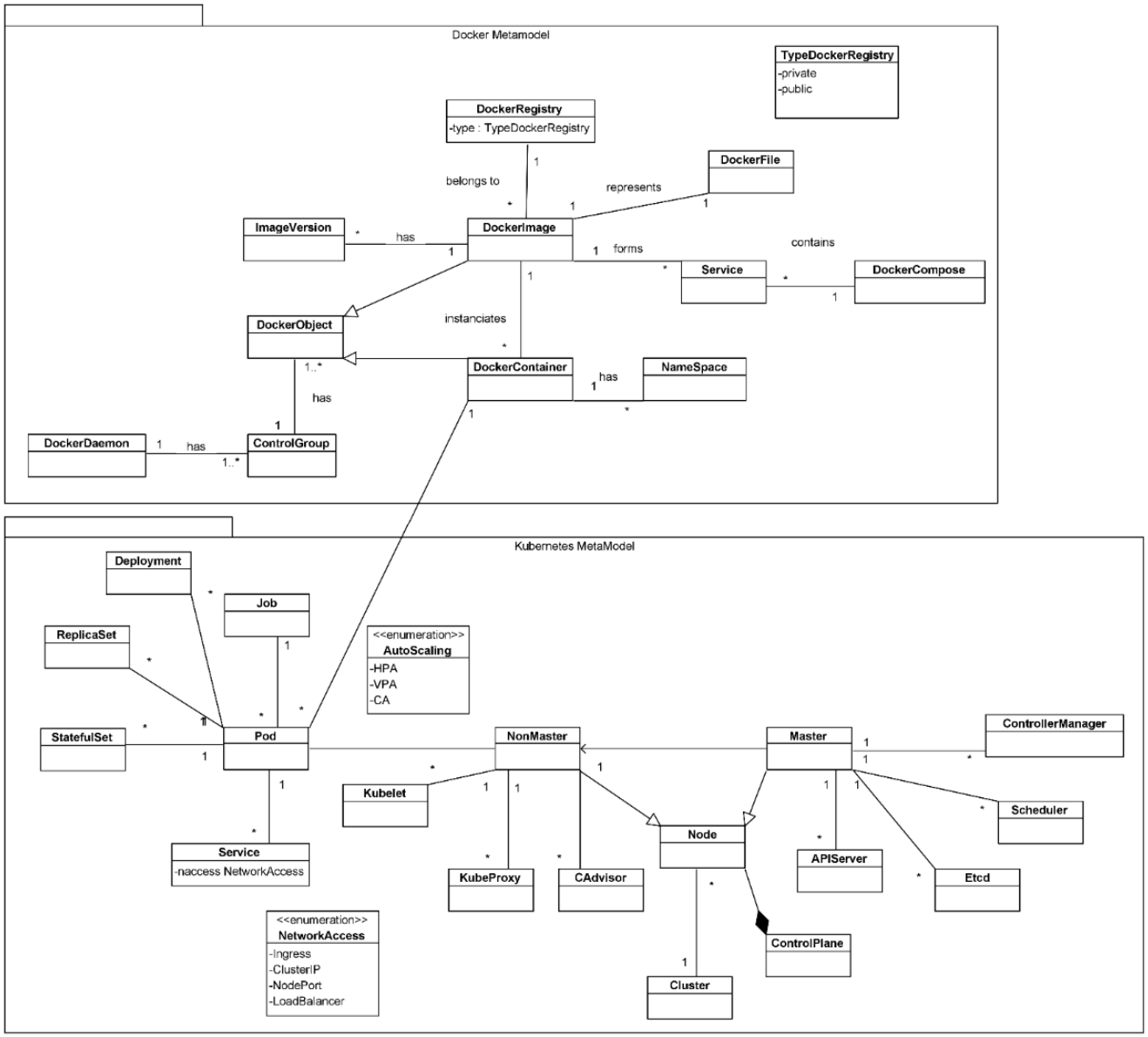

Figure 8: Communication between Kubernetes and Docker Ecosystem

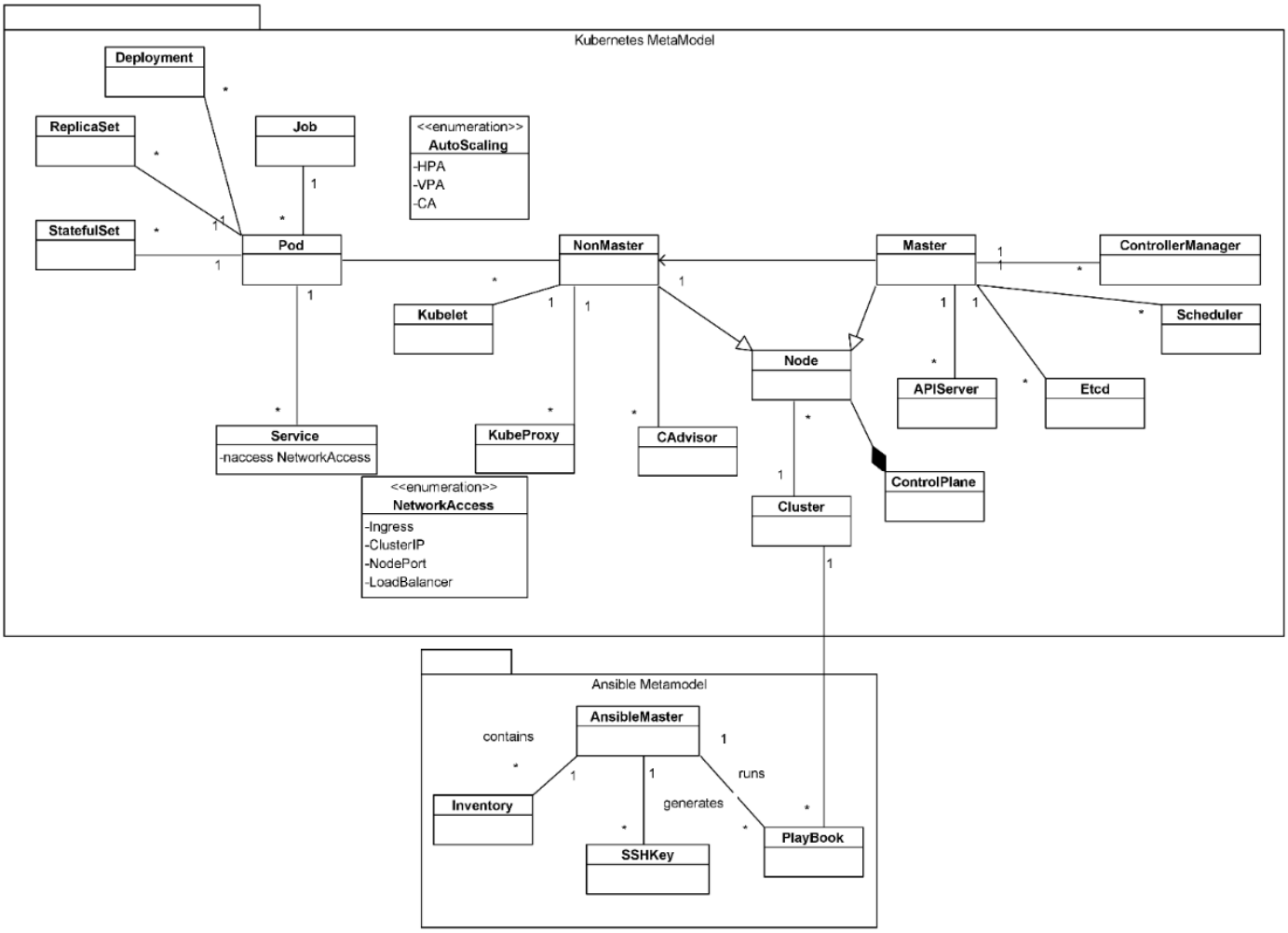

Figure 9: Communication between Kubernetes and Ansible Ecosystem

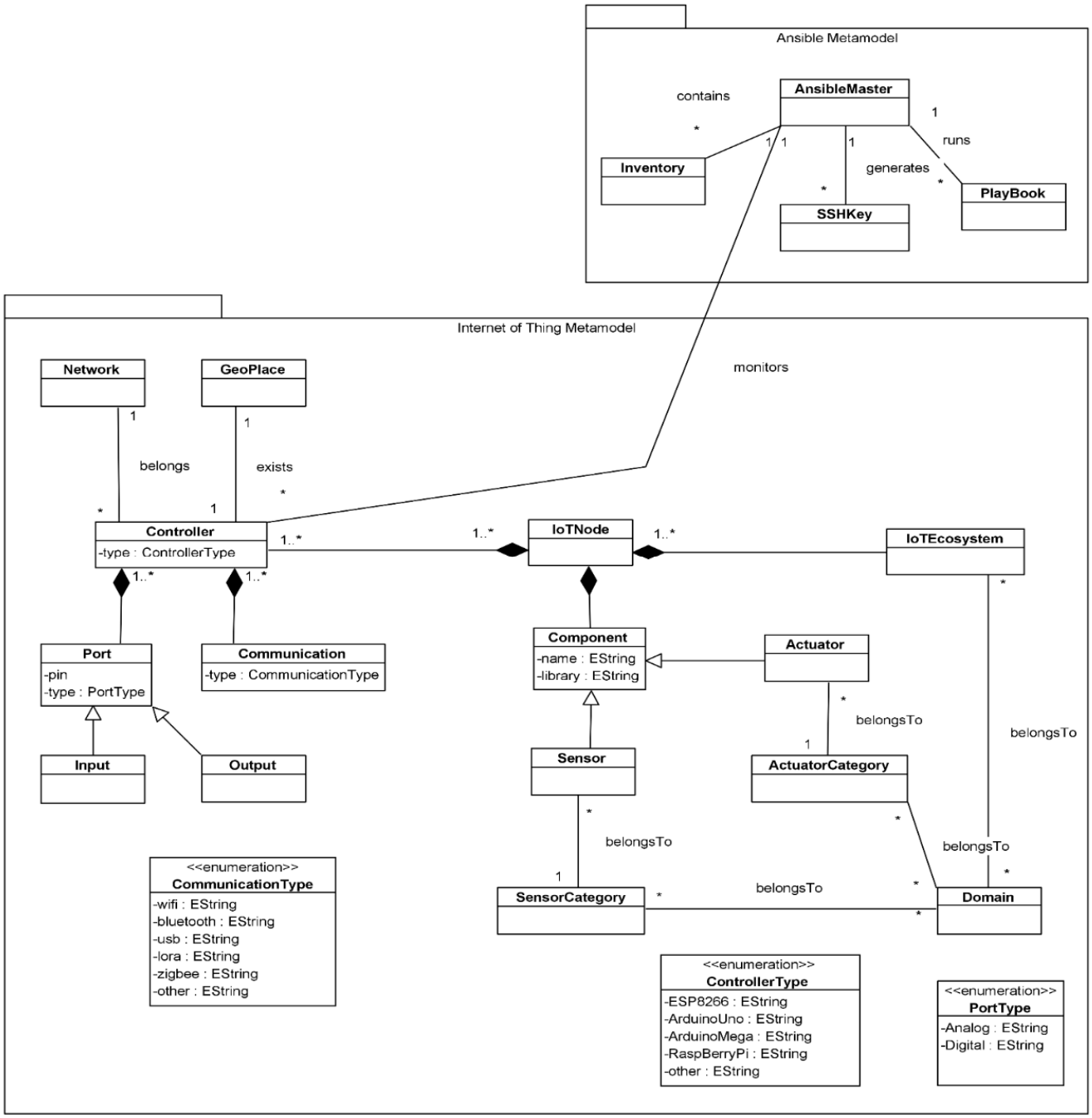

Figure 10: Communication between IoT and Ansible Ecosystem

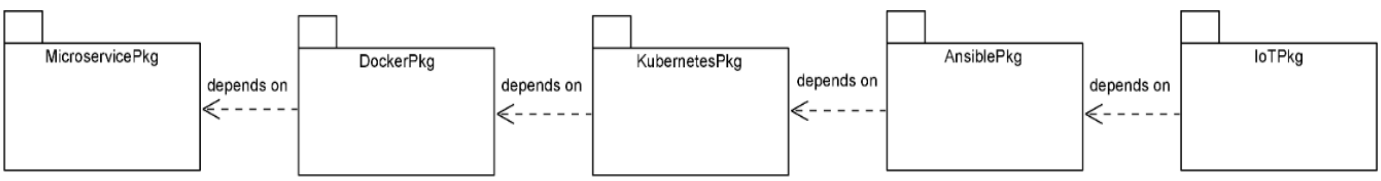

Figure 11: Global Meta-packages

9. DISCUSSION

In this paper, we have elaborated a Meta-model which is a fusion of 5 Meta-models: Meta-model of connected objects - Meta-model of microservices - Meta-model of Docker - Meta-model of Ansible – Meta-model of Kubernetes. First of all, we improved the Meta-model of connected objects quoted in the state of the art, this improvement will allow identifying the domain to which the objects belong as well as the

categories of sensors and actuators, so it allows the geographical and network identification in an Internet of Things ecosystem. Secondly, we have realized an improvement for the Meta-model of microservices by providing appropriate microservices for the management of connected objects and data processing and decision support for the Internet of Things agents. Besides, we have developed a Meta-model for containerization Docker, in which we have grouped the concepts constituting an ecosystem that aims to dock the microservices that will be deployed in objects and machines, whether at the cloud or fog level. Also, we have developed an Ansible Meta-model, which brings together the components of this system and whose objective is to monitor the containers in the machines and controllers. Finally, we have grouped these Meta-models into a single one and realized a dependency between the layers that are represented by the packages. Each package represents a Meta-model. The objective of this Meta-model is to generate an Internet system of objects based on microservices and supported by Devops technologies: Docker, Ansible & Kubernetes.In future works, we will use this Meta-model to realize a model transformation using ATL Language [54,55,56,57], and Model Driven Engineering approach [58,59]

10. CONCLUSION

The paper talks about a fusion of 5 Meta-models: Internet of things Meta-model, Microservice Meta-model, Docker Meta-model, Ansible Meta-model, Kubernetes Meta-model. We have elaborated an amelioration of the existing Meta-model regarding IoT, we have elaborated an amelioration of Microservice Meta-model by adding enumerations that correspond to the Internet of Things need. Then we have made a new proposition regarding Docker, Ansible Meta-model and Kubernetes Meta-model

REFERENCES

1. Ashton, Kevin. "That ‘internet of things’ thing." RFID journal 22.7 (2009): 97-114.
2. Thönes, Johannes. "Microservices." IEEE software 32.1 (2015): 116-116.
3. Bass, Len, Ingo Weber, and Liming Zhu. DevOps: A software architect's perspective. Addison-Wesley Professional, 2015.
4. Dragoni, Nicola, et al. "Microservices: yesterday, today, and tomorrow." Present and ulterior software engineering. Springer, Cham, 2017. 195-216.
5. Merkel, Dirk. "Docker: lightweight linux containers for consistent development and deployment." Linux journal 2014.239 (2014): 2.
6. Mohaan, Madhurranjan, and Ramesh Raithatha. Learning Ansible. Packt Publishing Ltd, 2014.
7. Kubernetes. https://kubernetes.io/. Accessed 19 Aug 2020
8. Hüttermann, Michael. DevOps for developers. Apress, 2012.
9. LIU, Bin, and Zui WANG. "Application of office automation based on ssh framework [J]." Computer Technology and Development 1 (2010): 39.
10. Schmidt, Douglas C. "Model-driven engineering." COMPUTER-IEEE COMPUTER SOCIETY- 39.2 (2006): 25.
11. Erraissi, A., And Belangour, A. (2018). Data sources and ingestion big data layers: meta-modeling of key concepts and features. International Journal of Engineering and Technology, 7(4), 3607- 3612.
12. Erraissi A., Belangour A. (2019) Capturing Hadoop Storage Big Data Layer Meta-Concepts. In: Ezziyyani M. (eds) Advanced Intelligent Systems for Sustainable Development (AI2SD'2018). AI2SD 2018. Advances in Intelligent Systems and Computing, Flight 915. Springer, Ham
13. Erraissi Allae, and Abdessamad Belangour. "Hadoop Storage Big Data Layer: Meta-Modeling of Key Concepts and Features." International Journal of Advanced Trends in Computer Science and Engineering 8, No. 3 (2019): 646-53.
14. Kleppe, Anneke G., et al. MDA explained: the model driven architecture: practice and promise. Addison-Wesley Professional, 2003.
15. Lee, In, and Kyoochun Lee. "The Internet of Things (IoT): Applications, investments, and challenges for enterprises." Business Horizons 58.4 (2015): 431-440.
16. Erraissi Allae, and Abdessamad Belangour. "Meta-Modeling of Big Data visualization layer using On-Line Analytical Processing (OLAP)." International Journal of Advanced Trends in Computer Science and Engineering 8, No. 4 (2019).
17. Erraissi, Allae, and Abdessamad Belangour. "An Approach Based on Model Driven Engineering For Big Data Visualization In Different Visual Modes." International Journal of Scientific & Technology Research (2020).
18. Erraissi Allae, and Abdessamad Belangour. "A Big Data Security Layer Meta-Model Proposal." Advances in Science, Technology and Engineering Systems Journal 4, No. 5 (2019). https://doi.org/10.25046/aj040553.
19. Breton, Erwan, and Jean Bézivin. "Towards an understanding of model executability." Proceedings of the international conference on Formal Ontology in Information Systems-Volume 2001. 2001.
20. Muller, Pierre-Alain, Franck Fleurey, and Jean-Marc Jézéquel. "Weaving executability into object-oriented meta-languages." International Conference on Model Driven Engineering Languages and Systems. Springer, Berlin, Heidelberg, 2005.
21. Online http://www.kermeta.org.
22. Koudri, Ali, Joël Champeau, and Denis Aulagnier. "Une sémantique opérationnelle pour une meilleure méta-modélisation." 3e journées sur l’Ingénierie Dirigée par les Modèles, France (2007): 223-228.
23. Bendraou, Reda, Marie-Pierre Gervais, and Xavier Blanc. "UML4SPM: A UML2. 0-based metamodel for software process modelling." International Conference on Model Driven Engineering Languages and Systems. Springer, Berlin, Heidelberg, 2005.

24. Bendraou, Reda, et al. "Software process modeling and execution: the UML4SPM to WS-BPEL approach." 33rd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO 2007). IEEE, 2007.
25. Bendraou, Reda, et al. "Definition of an Executable SPEM 2.0." 14th Asia-Pacific Software Engineering Conference (APSEC'07). IEEE, 2007.
26. Bendraou, Reda, et al. "Vers l'Exécutabilité des Modèles de Procédés Logiciels." 2008.
27. Combemale, Benoît, et al. "Expériences pour décrire la sémantique en ingénierie des modèles." Hermes SCIENCES/LAVOISIER, éditeur: 2ième journées sur l'Ingénierie Dirigée par les Modèles (IDM) (2006): 17-34.
28. Combemale, Benoit, et al. "Towards Rigorous Metamodeling." MDEIS 6 (2006): 23-27.
29. Harmsen, Frank, and Motoshi Saeki. "Comparison of four method engineering languages." Working Conference on Method Engineering. Springer, Boston, MA, 1996.
30. Rolland, Colette. "Capturing system intentionality with maps." Conceptual modelling in Information Systems engineering. Springer, Berlin, Heidelberg, 2007. 141-158.
31. Henderson-Sellers, Brian, and Jolita Ralyté. "Situational method engineering: state-of-the-art review." Journal of Universal Computer Science (2010).
32. Seidita V., Ralyté J., Henderson-Sellers B., Cossentino M., Arni-Bloch N., « A comparison of deontic matrices, maps and activity diagrams for the construction of situational methods », CAiSE Forum, 19th Int. Conf. on Advanced Information Systems Engineering, Trondheim, Norway, 11-15 June, 2007
33. Rolland C., « Method engineering: towards methods as services », Software Process: Improvement and Practice, vol. 14, n°3, pp. 143-164, 2009
34. BANANE, Mouad, ERRAISSI, Allae, et BELANGOUR, Abdessamad. SPARQL2Hive: An approach to processing SPARQL queries on Hive based on meta-models. In: 2019 8th International Conference on Modeling Simulation and Applied Optimization (ICMSAO). IEEE, 2019. p. 1-5.
35. Banane, Mouad, and Abdessamad Belangour. "A new system for massive RDF data management using Big Data query languages Pig, Hive, and Spark." International Journal of Computing and Digital Systems 9.2 (2020): 259-270.
36. BANANE, Mouad et BELANGOUR: A Big Data Solution To Process Semantic Web Data Using The Model Driven Engineering Approach, International Journal of Scientific & Technology Research. vol. 9, no 02, 2020.
37. Banane, Mouad, and Abdessamad Belangour. "Towards a New Scalable Big Data System Semantic Web Applied on Mobile Learning." International Journal of Interactive Mobile Technologies (iJIM) 14.01 (2020): 126-140.
38. A. Erraissi, B. Mouad and A. Belangour, "A Big Data visualization layer meta-model proposition," 2019 8th International Conference on Modeling Simulation and Applied Optimization (ICMSAO), Manama, Bahrain, 2019, pp. 1-5. doi: 10.1109/ICMSAO.2019.8880276
39. Erraissi Allae, et Abdessamad Belangour. « A Big Data Security Layer Meta-Model Proposition ». Advances in Science, Technology and Engineering Systems Journal 4, nº 5 (2019). https://doi.org/10.25046/aj040553.
40. Erraissi, Allae, and Abdessamad Belangour. "An Approach Based On Model Driven Engineering For Big Data Visualization In Different Visual Modes." International Journal of Scientific & Technology Research.
41. Fatima Kalna, Allae Erraissi, Mouad Banane, Belangour "A Scalable Business Intelligence Decision-Making System in The Era of Big Data" International Journal of Innovative Technology and Exploring Engineering 2019. https://doi.org/10.35940/ijitee.L3251.1081219
42. Kelly, S., Lyytinen, K., Rossi, M., « MetaEdit+: A Fully Configurable Multi-User and MultiTool CASE Environment », Proceedings of 8th Int. Conf. on Advanced Information Systems Engineering (CAiSE'96), LNCS 1080, Springer-Verlag, pp. 1–21.
43. Online http://www.metacase.com/.
44. Kelly S., Tolvanen J., Domain-Specific Modeling: Enabling Full Code Generation, Wiley & Sons, New Jersey, 2008
45. Jarke M., Jeusfled M.A., Nissen H.W., Quix C., Staudt M., « Metamodelling with Datalog and Classes: ConceptBase at the Age of 21 » 2nd Int. Conf. Object Oriented Data Bases (ICOODB'09), LNCS 5936, pp.95-112, Springer, 2010
46. Mylopoulos J., Borgida A., Matthias J., Koubarakis M., « Telos: representing knowledge about information systems », ACM Transactions on Information Systems, vol. 8, Issue 4
47. Jeusfled M.A., Jarke M., Mylopoulos J., (Eds), Metamodeling for Method Engineering, The MIT Press, 2009.
48. D. Alulema, J. Criado, and L. Iribarne, "A Model-Driven Approach for the Integration of Hardware Nodes in the IoT," in Advances in Intelligent Systems and Computing, 2019, vol. 930, pp. 801–811.
49. N. Alshuqayran, N. Ali, and R. Evans, "Towards Micro Service Architecture Recovery: An Empirical Study," in Proceedings - 2018 IEEE 15th International Conference on Software Architecture, ICSA 2018, 2018, pp. 47–56.
50. Badr El Khalyly, et al. " A comparative study of Microservices-Based IoT platforms" International Journal of Advanced Computer Science and Applications (IJACSA) 12, no. 8 (2020).
51. M. Zadka and M. Zadka, "Ansible," in DevOps in Python, Apress, 2019, pp. 139–145.
52. Hall, Daniel. Ansible Configuration Management. Packt Publishing Ltd, 2015.
53. Erraissi, Allae, and Abdessamad Belangour. Meta-Modeling of Big Data Management Layer. International Journal of Emerging Trends in Engineering Research 7, 7, 36-43, 2019. https://doi.org/10.30534/ijeter/2019/01772019.
54. Banane, Mouad, Allae Erraissi, and Abdessamad Belangour. "SPARQL2Hive: An approach to processing

SPARQL queries on Hive based on meta-models." 2019 8th International Conference on Modeling Simulation and Applied Optimization (ICMSAO). IEEE, 2019.
55. Erraissi, Allae. "Using model Driven Engineering to transform Big Data query languages to MapReduce jobs." International Journal of Computing and Digital Systems 9 (2020): 1-9.
56. Jouault, Frédéric, et al. "ATL: A model transformation tool." Science of computer programming 72.1-2 (2008): 31-39.
57. BANANE, Mouad et BELANGOUR, Abdessamad. Towards a New Scalable Big Data System Semantic Web Applied on Mobile Learning. International Journal of Interactive Mobile Technologies (iJIM), 2020, vol. 14, no 01, p. 126-140.
58. BANANE, Mouad et BELANGOUR: A Big Data Solution To Process Semantic Web Data Using The Model Driven Engineering Approach, International Journal of Scientific; Technology Research. vol. 9, no 02, 2020.
59. Banane, Mouad, and Abdessamad Belangour.; A new system for massive RDF data management using Big Data query languages Pig, Hive, and Spark; International Journal of Computing and Digital Systems 9.2 (2020): 259-270