OLIVE, a Model-Aware Microservice Framework

Damiano Falcioni, Robert Woitsch

OLIVE, a Model-Aware Microservice Framework

Damiano Falcioni[1] and Robert Woitsch[1]

[1] BOC GmbH, Operngasse 20b, 1040 Vienna, Austria
damiano.falcioni@boc-eu.com, robert.woitsch@boc-eu.com

Abstract. In this paper we want to introduce OLIVE, a model-centric and low-code microservice framework, resulting from the lessons learned in five years of European projects. The requirements on those projects are summarized and used to extract the characteristics that a microservice framework should support in order to be aware of models. An implementation of the OLIVE framework has been proposed, focusing more on the concepts, and required models, instead that on the technical details and has been evaluated in each project with definition of the needed microservices. Microservice development using a low code approach is still an open research field and with OLIVE we want to provide an initial contribution relative to the dependencies between microservices and models, using ADOxx as reference meta-modelling platform, and proposing an initial modelling method for the definition of OLIVE microservices.

Keywords: model-centric microservices, low-code platform, OLIVE framework.

1 Introduction

Models are an abstract way to represent relevant knowledge about a specific domain. The advantages of having such knowledge in a structured model, in addition to help reasoning about a specific problem, concern mainly the following purposes: documentation for the analysed problem/situation, configuration for model dependents functionalities, and data source for processing mechanisms [1]. In this paper we focus on the latter two cases, analysing their requirements and defining a framework for their management. Usually, model dependents functionalities and mechanisms do not have direct access to the model information, except when they are integrated in the modelling platform. In this case they can retrieve the model details directly from the platform, but in the majority of situations, they are implemented in external components and services that must implement the logic to parse and understand the model.

In this paper we identified different modalities in which a model can be used and required by a service, based on the experience in past European projects, and we defined a framework named OLIVE, addressing such requirements for the definition of services. With OLIVE we focus in particular on definition of microservices and their relations with the ADOxx meta-modelling platform, following a low-code approach.

Microservice is an architectural style increasing in popularity. A microservice is a small, autonomous, and isolated service, with a single and clearly defined purpose [2].

The focus of a microservice logic on the specific business task in particular, makes this architecture an ideal candidate for the abstraction in a model type enabling a low-code development [3].

The low-code approach [4] is a visual programming style that minimize the need of coding, abstracting the business logic in graphical form and favouring the configuration of existing components provided by the low-code platform. This approach has been chosen because, according to [5], low-code platforms will be used in 65% of application development work by 2024 and the ADOxx meta-model platform enable such business logic abstraction using a specific modelling method.

An implementation of this framework is provided, focusing more on the motivation and concepts behind the OLIVE framework instead then on its technical details. The proposed implementation has been evaluated in the context of each project and, at the end, the strengths and weaknesses of the approach have been reported.

2 Methodology

The model-aware characteristics of a microservice framework have been extracted in the last five years of involvements in European projects, from the requirements related to the meta-modelling platform ADOxx, used to create the domain specific model types for each project, in terms of both modelling methods and mechanisms [1]. In the following, such requirements have been briefly introduced for each EU project.

The **GO0D-MAN (H2020-723764)** project [6] constitutes a real-world implementation of the Industry 4.0 paradigm, through the integration and convergence of technologies for measurement and quality control, for data analysis and management, at single process and at factory level. The ultimate goal is to develop a production strategy that can guarantee high quality of products without interfering, actually improving, the production efficiency of the entire system. The ADOxx platform was used to develop a model-based tool for Industry 4.0 that supports multi-agent systems, smart on-line inspection tools and data analytics, for implementing a Zero-Defect Manufacturing (ZDM) strategy. A KPI meta-model has been created in this context in order to describe how to combine sensors data in order to calculate a KPI. Such modelling method had to be enriched with a mechanism to perform the effective KPI calculation based on the information in the model. One requirement here was to have such mechanism as a set services, one for each KPI model, where the user specify a KPI, and the service calculate and return the value. This imply that such services must (a) be able to be configured with the specific KPI model and (b) understand the KPI concepts.

DISRUPT (H2020-691436) [7] supports the digital twin of European manufacturer enterprises to Industry 4.0 by utilising the ICT capabilities to facilitate in-depth (self-)monitoring of machines and processes, provide decision support and decentralised (self-)adjustment of production, and foster the effective collaboration of the different IoT-connected machines with tools, services, and actors. The IoT devices, that create virtual counterpart of each element in the production line, provide data that will be analysed using models in the ADOxx platform, to detect complex events that will trigger automated actions. A modelling method for the DMN standard [8] have been

implemented in ADOxx for modelling the rules to apply on each event as well as the mechanism to evaluate the rule, integrating an existing DMN evaluation engine, and to listen for events, integrating a connection with a message bus system. Such requirements, in order to be fulfilled, needed a system that (a) allows to integrate the ADOxx platform with external systems, (b) access the DMN meta-model concepts and (c) be generic enough to work in different scenarios (different event topics to listen on the message bus, each one with a specific DMN model to use for the evaluation) and to develop the services performing the triggered actions.

In **BIMERR (H2020 820621)** [9] have been designed and developed a Renovation 4.0 toolkit to support renovation stakeholders throughout the renovation process of existing buildings, from project conception to delivery. It comprises tools for the automated creation of enhanced building information models, a renovation decision support system to aid the designer in exploring available renovation options through an accurate estimation of renovation impact on building performance, as well as a process management tool based on the ADOxx platform that will optimize the design and on-site construction process toward optimal coordination and minimization of renovation time and cost. This have been done enriching the renovation process meta-model with a simulation mechanism that, taking as input some influencing factors for a specific process model like the expectation of supply chain delay or the weather conditions, will estimate the values of some time and cost related KPIs. The simulation mechanism had to be (a) a service (b) integrated into the meta-modelling platform in order to (c) recognize the concepts in the process to simulate and in the KPIs to estimate. Additionally, the renovation process, in order to be executed as workflow, needed a system that (a) connect all the workflows' automated tasks with appropriate services, (b) implementing the task logic.

With **Change2Twin (H2020 951956)** [10] the manufacturing small-medium enterprises (SMEs) where supported in their digitalization process by providing digital twin solutions through trusted local innovation hubs. Such digital innovation hubs (DIHs) analyse the digitalization potential and propose the best ready-to-use recipe for the use case from a technology marketplace, using dedicated knowledge models created with the ADOxx platform, to prepare the recipe for a complete solution, including the best components for the SME. One of the available receipts include a KPI dashboard, delivered as a service and configured through a specific KPI model that details not only how to calculate each KPI but also how to extract the required metrics from Internet of Things (IoT) sensors. This enhancement of the KPI model and dashboard respect to the ones in the GO0D-MAN project, required (a) a configurable system able to (b) connect with different IoT sensors and (c) integrate with the meta-modelling platform ADOxx. A specific meta-model has been created also to support the definition of marketplace items and the automatic generation of the marketplace portal using (a) a microservice that (b) interpretate the marketplace model and generate the portal pages.

Analysing the requirements pointed out in the above projects, we can identify some common characteristics. First, all the cases require the implementation of the functionalities in form of dedicated services. Service Oriented Architecture (SOA) and in particular Microservice Architecture (MSA) are well known styles that can be used for such purposes due to the focus on the business concepts, isolation, decentralization, and

culture of automation [11]. Another recurring requirement is that such services must be configurable using models. Configurability is a characteristic that allows services to be reused in different contexts and we can see the needs to have multiple instances of the same service running in parallel, each one configured with a different model. The configurability of services using models rise the problem of understanding the semantic and the concepts in the model. Accessing the meta-model information is a requirement in order to avoid that the logic to interpretate the model as a configuration file will be hardcoded in the service. Accessing instead the meta-model information, like hierarchies and dependencies between concepts in the model, allows us to generalize and centralize the model interpretation in a framework instead that inside the service. The ADOxx meta-modelling platform used to create the model type for each project, provide the possibility to access such meta-model information, so an integration is required. This integration is used also to solve a last requirement that we can spot in the above projects: the need to extend the modelling methods of some project dependent model types, with mechanisms that provide a model specific feature directly inside the modelling platform. The ADOxx platform already allows to define custom mechanisms in form of algorithms, for a specific model type, but miss the possibility to use external and existing services as mechanisms and therefore communicate with the external world.

From this analysis we defined a framework, named OLIVE, for the definition of microservices with strong (meta-)model dependencies. Such framework highlights the following characteristics related to the model-awareness:

- Model-based Configurability: The framework gives the possibility to use models for the configuration of services. Additionally, the framework abstracts the services enough to automatically generate running service instances using only models. The microservice is created reflecting the concepts in the meta-model and is instantiated by the single model.
- Model Understandability: The framework allows to use models as input data for a service, giving the possibility to understand the concepts in a model, accessing the meta-model information.
- Meta-Model Enrichment: The framework expands the functionalities of the connected meta-modelling platform, providing a way to use services to enrich the mechanisms associated to a model type.

The OLIVE framework has been implemented as a microservice management system connected to the ADOxx meta-modelling platform. The proposed implementation has been successfully validated over the previously introduced European projects as described in the section 5.

3 Related works

Microservice Architecture is a recent style in software engineering and there are not so many papers addressing the application of a model-based approach for their definition. Proposals have been made for model-based microservice frameworks but mainly to solve specific microservice issues, like self-adaptability, and scalability in multi-cloud

environments, with the help of specific models containing dynamic requirements and adaptation rules [12], or to define the microservice logical architecture and its implementation skeleton from models describing functional requirements [13]. The OLIVE framework instead, want to address the complete generation of running microservice instances, starting from a model, used as configuration.

The approach followed share similarities with concepts in low-code/no-code frameworks [4]. Indeed, in the OLIVE framework, the user will not code a microservice, but will configure existing generic components provided by the platform, using specific models, in order to generate running microservices instances.

The authors in [14] use models to create self-configuring microservices based on TOSCA [15] standard in order to avoid a centralized orchestration. Microservices require indeed decentralized management and prefer choreography over orchestration [16]. The authors of [17] state that more research on choreography rather than orchestration is required. In the past, model-based orchestration has been used in the BPEL standard [18] that define a model-driven SOA architecture to address a specific business need, combining existing SOAP services. Within OLIVE we tried to be general enough to support both choreography and orchestration patterns, with the possibility to define dependencies between running microservices instances as well as generate REST microservices that act as orchestrators of other services.

The enhancement of a modelling method with mechanisms is a concept seen in [1] and supported in the ADOxx meta-modelling platform by a scripting language named ADOScript that only provide a limited set of features and do not allows complex interactions with external services. Other meta-modelling platforms like Eclipse EMF [19] do not permit to integrate mechanisms in the defined domain specific language (DSL) but allows only to use separate plugins to extend the DSL modelling environment with the needed features. The only attempt to extend a meta-modelling environment with microservices is in [3], where the authors provided an approach to integrate microservices as components of the DIME modelling environment. With the OLIVE framework we want to extend the ADOxx meta-modelling platform to use microservices as mechanisms for specific modelling methods.

4 Olive

OLIVE is a low-code framework to create model aware microservices through configuration of existing components, named Connectors resulting in ready to use REST microservices. The configuration of such Connectors can be performed using specific models, reflecting the framework concepts, created using the ADOxx graphical modelling environment. The connectors are part of the OLIVE platform but can be extended in case of needs, using plug-ins and provide the atomic functionality of a microservice. The created microservices can be combined together using a choreography or orchestration approach for the definition of your specific business logic.

The strength point of OLIVE is its model-awareness in the sense that the configurations of connectors are abstract enough that can be represented as models, and the out-of-the-box integration with the ADOxx meta-modelling environment allows to create

the whole behaviour of a service, using models. This integration allows also ADOxx to communicate with the external world through a common interface, in a bi-directional way, using the microservices as mechanisms to enrich the modelling methods and using models as data for microservices. The OLIVE platform provides such features as a cloud environment where the user, instantiating existing components with specific configuration, can define the microservices, that can use and be used by ADOxx, and control their lifecycle. The Fig. 1 provide an overview of all the components involved in the framework.

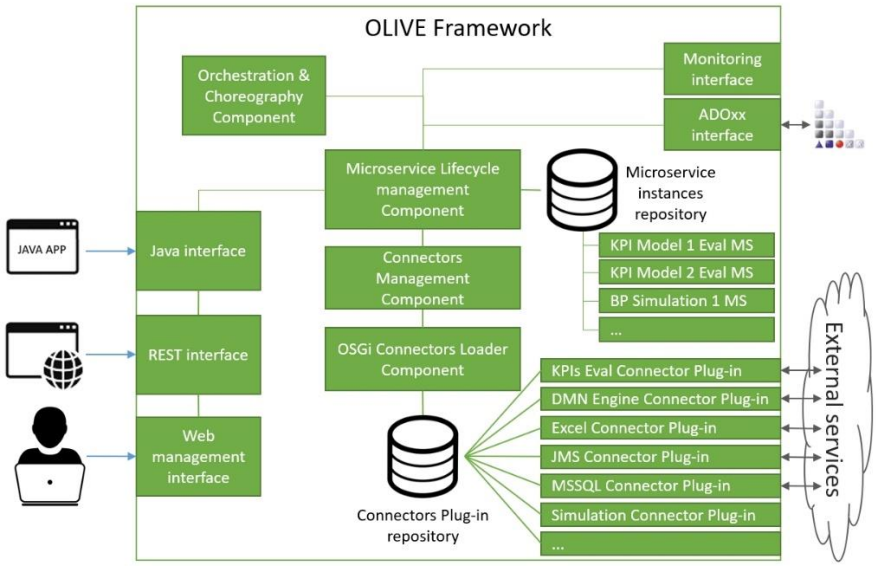

Fig. 1. OLIVE Framework Architecture

A Connector is a component developed in form of OSGi plug-in [20] that provide a specific functionality, like perform a query on a specific database or evaluate a DMN model. The name Connector derive from the fact that, usually (but not always), the main functionality is provided by an external system (like a database system or DMN engine) and the Connector is responsible to connect to such system to exploit its features. Each connector has its specific set of parameters that must be configured to generate a microservice instance (i.e., a database connector is configured with the endpoint and the query to perform). Olive integrates out-of-the-box 24 connectors. Custom connectors can be added to the platform and their OSGi standard format allows to reuse as connectors existing OSGi bundles, like all the one provided by the Apache Camel [20] project.

OLIVE allows to manage the configuration of such Connectors, giving the possibility to create microservices and control their whole lifecycle. Is responsibility of the Lifecycle Management component to (1) generate an instance of the REST microservice from the configuration of a Connector, (2) start the microservice, (3) keep the microservice running in an isolated environment, (4) stop the microservice and (5) dismiss it.

Each instantiated connector is exposed through a common REST interface with a standardized inputs and output formats, managed by the OLIVE platform.

The OSGi Connectors Loader component is responsible to load all the Connectors and make them available to the platform. It is built on the OSGi framework Apache Felix [21] and will dynamically check the presence of the OSGi bundles (plug-ins) defining Connectors, loading, and unloading them on request.

The ADOxx interface is creates as a special type of Connector, managing the bi-directional communication with the ADOxx platform. This component allows to retrieve models or concepts and use the available microservice instances from inside the ADOxx modelling environment.

As soon as the microservices have been defined, it can be combined to achieve the business logic task, thanks to the Orchestrator & Choreography component. This component allows to define and evaluate dependencies between microservice instances. Depending on the topology of such dependencies the group of microservices can work as a choreography or as an orchestration.

The OLIVE framework exposes all this functionality both with Java and REST APIs. The firsts are used to integrate OLIVE in local and desktop application. The seconds are used to integrate it with remote applications. Over the REST APIs has been made available a management web user interface that allows to exploit all the features of the framework through a web browser.

A specific model type, named Microservice Definition Model, has been created in the ADOxx platform in order to collect the most relevant concepts of the OLIVE framework and be able to define OLIVE microservices using models in a low-code style. The model in Fig. 2 is an example of a Microservice Definition Model created in the context of the BIMERR project. A Microservice Definition Model is composed of objects representing microservices of different types, based on the OLIVE Connector used, and relations representing dependencies between microservices.

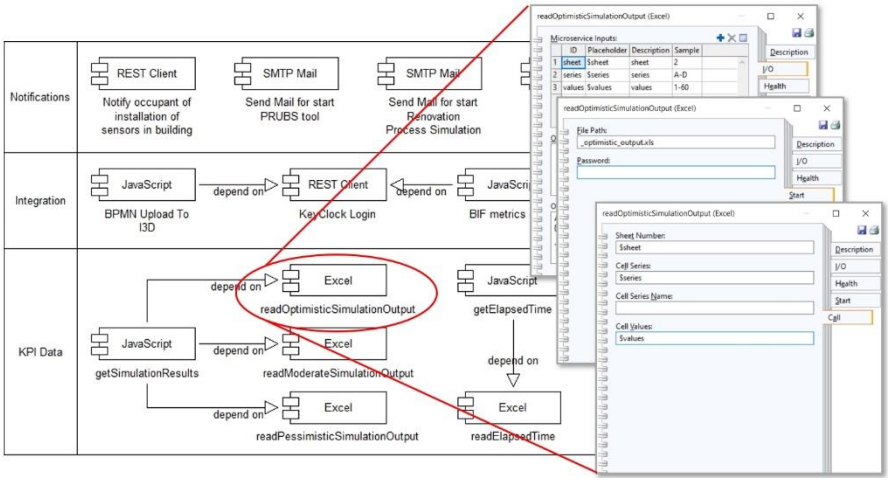

Fig. 2. OLIVE Microservice Definition Model sample

Each of the specific microservice type objects has a set of common and specific attributes. Common attributes represent general microservice concepts like all the attributes related to the input parameters and the output of the microservice or about its health status. Specific attributes dependent instead on the Connector selected to provide the specific functionality for the microservice and are the parameters that this connector needs in order to work correctly. As a sample, in the case of the Excel connector used in the "ReadOptimisticSimulationOutput" microservice in Fig. 2, that read a simulation output from an Excel file, such attributes involve the path of the Excel file to open and the cells value to read, specified by sheet, rows and columns IDs. The microservice input parameters are then mapped and forwarded to the specific connector attributes values.

The Microservice Definition Model type also contains a mechanism to automatically push the modelled microservice to the OLIVE platform for their execution and lifecycle management.

More detailed documentation on the OLIVE platform involving technical concepts and APIs, as wells as its open-source code and binaries for download and test, can be found in the OLIVE webpage [22].

5 Evaluation

In the context of the GO0D-MAN project a specific connector for the OLIVE framework, named KPIs dashboard, has been created in order to calculate the value of a KPI in a model. This connector is configurable with the model id, used to retrieve the model directly form the ADOxx platform, and with the KPI name to find the KPI object. Seven KPI models have been created for the different use cases in the project and the new connector has been configured with each model, resulting in seven running microservices instances.

In DISRUPT the OLIVE framework has been extended with a connector to an external DMN engine and with two Java Messaging Systems (JMS) [23] compliant message bus connectors, one for subscribing and another for publishing on specific channels. The JMS connectors have been used in the configuration of four microservices, three for receiving events, actions, and data, subscribing to the respective topics, and one for publishing the actions handling the received events. Another microservice is created configuring the DMN engine connector, configured with the specific DMN model obtained from the ADOxx platform. This microservice is called by the event subscription microservice each time an event is received, then it evaluates the DMN to find the appropriate action for the event, and finally send it to the publishing microservice. With the OLIVE framework a microservice for each action has been configured and instantiate. Such microservices involve sending SMS and e-mail notifications, reset a device in critical status, etc.

For the BIMERR project, a microservice for simulating specific renovation process KPIs, has been defined and integrated in the ADOxx platform as a feature specific for the renovation process and KPI models. Thanks to the OLIVE framework the models required for the simulation, are extracted from the ADOxx platforms and the process

concepts are mapped to a petri-net semantic that enable a token-based simulation. Also in this project, like in DISRUPT, the OLIVE framework has been used to generate notification services that, in this case, are referenced into renovation process workflows and called during their execution. In BIMERR, a modelling environment based on ADOxx for the definition of OLIVE microservices has been first released and used to model all the project specific OLIVE microservices. In this context, specific mechanisms for this microservice modelling environment have been created, in order to automatically generate the microservices documentation and to import the microservice definition in the OLIVE framework for their instantiation.

With Change2Twin the KPI model type defined in GO0D-MAN has been improved with information about how to obtain metrics values, referring the appropriate OLIVE microservice, responsible for the retrieval of specific IoT sensor data. The involved IoT sensors pushed data to different relational and time-series databases and connectors for each specific database have been created in OLIVE and configured to instantiate one microservice for each metric. Such microservices, referenced in the KPI model, are used, and called by the microservices responsible for the evaluation of each KPI model like in the GO0D-MAN use case. Finally, the modelling method created in ADOxx to describe the marketplace items, has been enriched with a mechanism, expressed as microservice in OLIVE, to generate the marketplace web pages for each item.

6 Conclusions

In this paper we presented our experience in different EU projects resulting on the definition of the relevant characteristics of a model centered microservice framework. An implementation has been proposed and evaluated over the projects. In particular in GO0D-MAN we evaluated the model-based configurability and model understandability characteristics of the framework with microservices for the evaluation of each KPI model. In DISRUPT, the meta-model enrichment and model understandability has been evaluated for the integration of ADOxx with microservices, handling the communication with a message bus and with a DMN evaluation engine. With BIMERR the model understandability has been exploited in a renovation process simulation microservice, while the model-based configurability enabled the definition of specific microservices for workflow automatic activities. In Change2Twin the meta-model enrichment characteristic of the framework has been exploited to integrate a marketplace generation microservice for a defined marketplace model while the model-based configurability, to generate microservices for retrieving the metric value of each IoT sensor.

Based on the experience in the above projects we have seen that the OLIVE framework enabled the integration of services in the ADOxx platform and improved the reuse of previously created logic, in particular about the extraction of information from models, speeding-up the release of microservices by reusing and adapting existing OLIVE connectors. Despite the big potential, the OLIVE framework still lacks some features in order to be extensively used as a low-code platform, in particular related to the logic abstraction. This is why we are planning to extend it with support for the Enterprise Integration Pattern (EIP) [24] notation and continue its evaluation in future projects.

References

1. Karagiannis, D. and Kühn, H. Metamodelling platforms. In EC-Web, vol. 2455, p. 182 (2002).
2. Lewis, J. and Fowler, M. Microservices 2014. http://martinfowler.com/articles/microservices.html, last accessed 2021/07/20.
3. Chauhary, H, Margaria, T. Integration of microservices as components in modelling environments for low code development, Syrcose (2021).
4. WOO, M. The rise of no/low code software development—No experience needed?. Engineering. Beijing, China, 6.9: 960 (2020).
5. Wong, J., Driver, M., & Vincent, P. Lowcode development technologies evaluation guide. https://www.gartner.com/en/documents/3902331. Gartner, Inc (2019).
6. GO0D MAN. Agent Oriented Zero Defect Multi-Stage Manufacturing. http://go0dman-project.eu/. Last accessed on 2021/07/20.
7. DISRUPT. Transform manufacturing for Industrie 4.0. http://www.disrupt-project.eu/. Last accessed on 2021/07/20.
8. OMG DMN. https://www.omg.org/dmn/. Last accessed on 2021/07/20.
9. BIMERR. BIM-based holistic tools for Energy-driven Renovation of existing Residences. https://bimerr.eu/. Last accessed on 2021/07/20.
10. Change2Twin. Bringing Digital Twins to Manufacturing SMEs. https://www.change2twin.eu/. Last accessed on 2021/07/20.
11. Newman, S. Building microservices: designing fine-grained systems. O'Reilly Media, Inc. (2015).
12. Pahl, C.; Jamshidi, P. Software architecture for the cloud–a roadmap towards control-theoretic, model-based cloud architecture. In: European Conference on Software Architecture. Springer, Cham, p. 212-220 (2015).
13. Santos, N., et al. Inputs from a Model-Based Approach Towards the Specification of Microservices Logical Architectures: An Experience Report. In: International Conference on Product-Focused Software Process Improvement. Springer, Cham, p. 473-488 (2019).
14. Kehrer, S.; Blochinger, W. AUTOGENIC: Automated Generation of Self-configuring Microservices. In: CLOSER, p. 35-46 (2018).
15. OASIS TOSCA. http://docs.oasisopen.org/tosca/TOSCA/v1.0/cs01/TOSCA-v1.0-cs01.html. Last accessed on 2021/07/20.
16. Zimmermann, O. Microservices tenets. Computer Science - Research and Development, 32(3-4):301– 310 (2017).
17. Schermann, G., Cito, J., and Leitner, P. All the Services Large and Micro: Revisiting Industrial Practice in Services Computing, pages 36–47. Springer, Berlin, Heidelberg (2016).
18. Pasley, J. How BPEL and SOA are changing web services development. IEEE Internet computing, 9.3: 60-67 (2005).
19. Steinberg, D. et al. EMF: eclipse modelling framework. Pearson Education (2008).
20. Apache CAMEL. https://camel.apache.org. Last accessed on 2021/07/20.
21. Apache FELIX. https://felix.apache.org/. Last accessed on 2021/07/20.
22. OLIVE Homepage. https://www.adoxx.org/live/olive. Last accessed on 2021/07/20.
23. JMS. https://en.wikipedia.org/wiki/Jakarta_Messaging. Last accessed on 2021/07/20.
24. EIP. https://www.enterpriseintegrationpatterns.com/. Last accessed on 2021/07/20