applied sciences

MDPI

Article

A Model-Driven Architecture for Automated Deployment of Microservices

Isil Karabey Aksakalli [1,*], Turgay Celik [2], Ahmet Burak Can [3] and Bedir Tekinerdogan [4]

[1] Department of Computer Engineering, Erzurum Technical University, 25050 Erzurum, Turkey
[2] BITES Defense & Aerospace, 06800 Ankara, Turkey; turgaycelik@gmail.com
[3] Department of Computer Engineering, Hacettepe University, 06800 Ankara, Turkey; abc@hacettepe.edu.tr
[4] Information Technology Group, Wageningen University, 6708 PB Wageningen, The Netherlands; bedir.tekinerdogan@wur.nl
* Correspondence: isil.karabey@erzurum.edu.tr

Abstract: Microservice architecture consists of a collection of loosely coupled, self-contained services that can be deployed independently. Given the limited capacity of the resources for a large number of services, the deployment of the services does not scale well and leads to operational complexity and runtime overhead. This paper proposes a model-driven approach for the automated deployment of microservices to minimize the execution cost and communication costs among the microservices. The identification of the feasible deployment is defined at the architecture design level based on the provided capacity of the resources and the collection of microservices. The corresponding tool support using Eclipse Modeling Environment is described, and a case study on book shopping is used to illustrate the approach.

Keywords: microservice architecture; model-driven architecture; eclipse modeling environment; model-driven microservice development; automated deployment of microservices

Citation: Aksakalli, I.K.; Celik, T.; Can, A.B.; Tekinerdogan, B. A Model-Driven Architecture for Automated Deployment of Microservices. *Appl. Sci.* **2021**, *11*, 9617. https://doi.org/10.3390/app11209617

Academic Editor: Ricardo Colomo-Palacios

Received: 6 September 2021
Accepted: 11 October 2021
Published: 15 October 2021

1. Introduction

Microservice architecture is an architectural style that consists of single functionality services adopting high cohesion and low coupling. The microservice architecture style reduces the interdependence of microservices to facilitate independent deployment, scaling, operation, and maintenance. Likewise, many practitioners and researchers adapt their applications to this architectural style to take advantage of the benefits of microservice architectures. Despite the benefits of microservice architectures, the efficient deployment of thousands of services to limited capacity resources is critical regarding resource utilization and deployment costs. Several industrial tools such as Kubernetes and Docker Swarm have been introduced, but, unfortunately, these do not consider the microservices' runtime behaviors, such as resource consumption and intercommunication during deployment. There are some overlay tools based on these management tools to improve deployment architecture. However, these tools generally perform the deployment according to the configuration file that is manually created by the user, which is a relatively slow, cumbersome, and inefficient task. Moreover, it is difficult to determine the minimum and maximum resources needed by microservices, such as CPU and memory requirements at runtime, to configure the deployment cost-effectively. If generated deployment models are not satisfactory, the deployment process is repeated by creating different configuration files.

For generating feasible deployment alternatives, it is necessary to efficiently minimize the total amount of memory required by the services at runtime, the communication costs among microservices, and the execution costs of the services on the existing nodes. Based on the assumption that each node can accommodate a maximum polynomial amount of microservices, this problem can be defined as an NP-complete problem that cannot be solved in polynomial time. As such, near-optimal solutions can be produced in polynomial time [1].

Several studies in the literature have focused on model-driven development (MDD), which is used for microservice architecture (MSA). So far, few studies have proposed the usage of MDD for microservice-based architecture, and these studies focused mainly on the deployment problem of microservices. For example, Rademacher et al. [2] explain how MDD abstraction, model transformation, and modeling perspectives provide solutions to MSA problems. The proposed MDD tools in the literature address various purposes, such as documentation, recovery [3], architectural analysis, verification and resilience [4]. There are also various studies in which CTAP (Capacitated Task Assignment Problem) algorithms are applied to different areas [5,6]; however, there is a gap in the literature regarding the implementation of CTAP at the design phase for microservice deployment.

Our previous study [2] focused on the deployment problem and proposed an MDD-based approach to generate feasible microservice deployment models. The models are presented by extracting many parameters such as the inter-service communication costs, the execution costs of the services on the available nodes, the memory capacity of the services and nodes. In this article, we elaborate on the practical tool challenges and discuss the logical infrastructure of these models, model transformation process, Emfatic [7], and Eugenia [8] plugins used in the architecture.

This paper is organized as follows. Section 2 introduces the Model-Driven Development (MDD), and Section 3 describes the advantages and challenges of microservices architecture. Section 4 presents the metamodels created for microservice-based architecture. Section 5 describes the case study to validate proposed model-driven architecture. Section 6 evaluates the proposed approach for the case study in terms of total communication and execution costs. Section 7 discusses the alternative frameworks to develop a model-driven architecture for deploying microservices, and finally, Section 8 concludes the paper.

2. Model-Driven Development

Modeling is usually defined as a means for communication, analysis or guiding the production process. Depending on the level of precision, a model can be considered a sketch, blueprint, or executable. An executable is defined as a model that model compilers can interpret to generate other artefacts. In model-driven development (MDD) the concept of models can be considered as executable models [9]. The MDD approach serves documentation purposes and includes complex MDD methods such as model transformation for design improvement and code generation [9]. Since the MDD approach is based on transforming the application design to executable code, only the models are updated in case of a requirement change. The application code is regenerated from the model instead of changing both the application code and models. Thus, maintenance cost is significantly reduced [10]. Stakeholder-specific models can be generated, such as for domain experts to create domain-specific conceptual models at a higher abstraction level than developers, and for developers to create models for software design. Thus, this approach is beneficial for complex distributed software systems through dividing the system into concern-specific modeling tasks [9].

MDD focuses on three key goals: (1) portability, (2) interoperability, and (3) reusability, and the key abstraction term to achieve these goals is "architectural separation of concerns" [11]. In the MDD approach, model transformations are used so that the target system can be (semi)automatically derived from the model. The basic model transformation methods can be summarized as follows:

- Model to model (M2M) transformation [12]: In this transformation method, a model has transformed another model according to a certain purpose. It is often used for automated transformation from Platform Independent Model (PIM) to Platform Specific Model (PSM) [13].
- Model to text (M2T) transformation [14]: This approach is designed to meet requirements, such as automatic code generation for different target programming languages (C++, Java, etc.), from the design model or for text generation for documentation purposes.

- Text to model (T2M) transformation: In T2M transformations, target models can be generated from text files suitable for certain metamodels. XText [15] and Gra2Mol (Grammar to Model Language) tools [16,17] are examples of environments that can be used for T2M transformations.

2.1. EMF and GMF Model Transformation Process

In the proposed model-driven architecture for deploying microservices, EMF [18] and GMF [19] modeling frameworks are used on the Eclipse Modeling Tool [20]. Eclipse Modeling Framework (EMF) is a set of Eclipse plugins used to design the data model and generate code and other outputs depending on the model. Graphical Modeling Framework (GMF) is an Eclipse Modeling project that generates the necessary models for the use of graphical tools.

Ecore metamodels represent the artifacts of models that can be created using EMF. An overview of model transformations from Domain Model (Ecore metamodel) to graphical Genmodel is shown in Figure 1. In the EMF part, an abstract syntax of the language is determined using Ecore, and then Java code is generated from the Ecore model in two phases using the EMF built-in code generator. The GMF part involves determining the editor's graphical syntax using a set of graphical syntax-specific GMF models in three stages. Then, the GMF code generator is used to generate the concrete graphical editor.

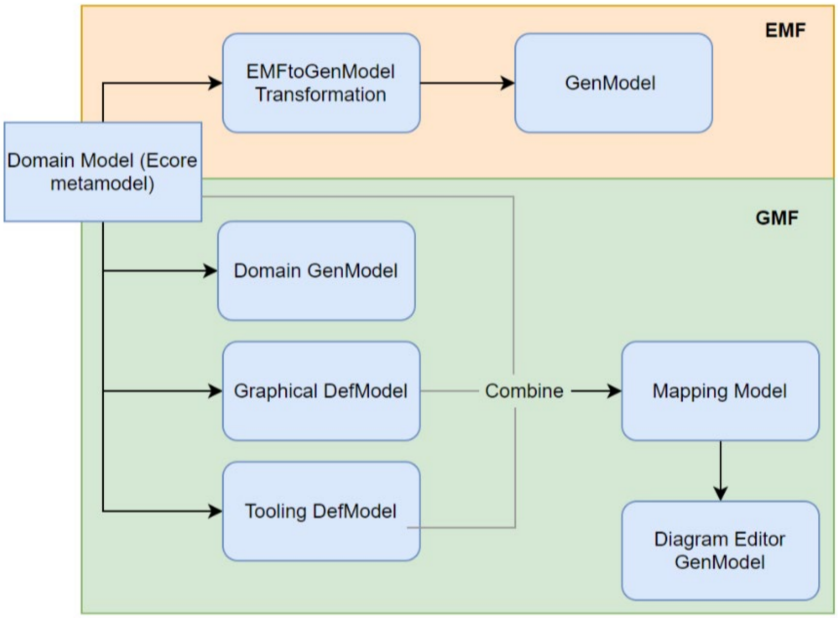

Figure 1. EMF/GMF model transformation process.

The Ecore metamodel contains information about the defined classes. GenModel converted from the Ecore metamodel specifies how the metamodel should be implemented in Java. The model includes necessary information for code generation, such as path and file information. GenModel also provides control parameters to ensure that the code is generated correctly. Graphical DefModel is used to define shapes, nodes, and connections on the diagram to be created. The graphical description created needs a template to be usable. Tooling DefModel is a necessary model for creating graphical elements and specifying operations such as palette, tool creation, actions. The combination of the Domain Model, Graphical DefModel, and Tooling DefModel creates the Mapping Model. The Mapping Model enables the determination of canvas tags required for drawing in the Eclipse Modeling environment. In the last step, Diagram Editor GenModel, which should be generated to present the drawing editor to the user, is derived from the Mapping Model.

2.2. Emfatic

There are many Ecore tools such as Validation, Compare, Search, EEF, MWE (Modeling Work Engine), Teneo, Texo, Emfatic, generators, etc., to create, edit and update Ecore models [21]. Emfatic is a textual syntax that enables EMF Ecore models to be created using software instead of being created in a visual environment [7]. Emfatic provides convenience to the designer in model-driven development. While creating the model-driven architecture for deploying microservices, we preferred Emfatic since it is a rich text editor and it facilitates the transformation from Emfatic files to Ecore models or Ecore models to Emfatic files. We used Emfatic as a plugin to design the metamodels required to deploy microservices; the corresponding Ecore metamodels are automatically generated. Each declaration in the emf file corresponds to Ecore constructs. For instance, each class defined in an Emfatic file corresponds to a metamodel's class and each attribute defined in an emf class represents the properties of the Ecore class.

2.3. Eugenia

EMF and GMF are powerful frameworks for modeling, but many steps need to be applied for model to model transformations (transition from Ecore metamodel to diagram editor). These powerful frameworks bring some complexities, and they require manual work to connect related models. In this study, an open-source tool named Eugenia [8] is used as a plugin on Eclipse Modeling Tool, which enables the model transformations in GMF more easily. Hereby, Domain GenModel, Graphical DefModel, and Tooling DefModel models are created automatically by going through a single transformation performed by Eugenia [8].

3. Microservice Architectures

Microservice architectures (MSA) is a novel paradigm for developing and deploying service-oriented software systems [22]. The architectural building blocks of MSA are software components consisting of services. These software components can be created as (i) loosely coupled with minimizing dependency with other components, (ii) conforming to predefined contracts for their interactions, and (iii) performing coarse-grained tasks [23].

The most important difference of MSA from other software architectures is that it emphasizes the independence principle specific to the services in terms of functionality, technological and organizational aspects [23,24]. Thus, a microservice is created to be responsible for only one business need or technical functionality. Moreover, service interactions are based on (i) the two most widely used communication protocols (synchronous and asynchronous), (ii) choreography as the standard model for architectural internal interactions, and (iii) lightweight API gateways for architectural-third party external system interactions with service consumers [25]. MSA encourages service teams with a high level of independence at the organizational level by keeping a separate team responsible for developing each service and its operation [24]. Adaptation to the MSA approach is expected to increase (i) the adaptability of a service, (ii) the quality and security of services, and (iii) the productivity of development teams [25].

Although adaptation to microservices brings many advantages, deployment challenges arise in transitioning from a monolith to microservice-based architecture. While a monolithic application can be deployed at one-time to the servers behind a load balancer, microservices consisting of multiple runtime instances must be deployed after the configuration process [1]. While this configuration process can be done manually in a microservice-based application consisting of a few services, it becomes challenging to create a cost-effective deployment model manually, as the number of services increases. Therefore, systematic approaches that support automatic deployment are required for microservices. In addition, representing the deployment process on a model and reporting the deployment result in detail ensures the easier management of a microservice-based software. In our previous study, the process steps for deriving feasible deployment alterna-

tives are explained in a case study [1]. In the following section, the logical infrastructure of the developed metamodels is described in detail.

4. Logical Infrastructure of the Generated Metamodels

This study describes the logical infrastructure of the proposed model-driven architecture [1] to derive feasible deployment models for microservices at the early design phase. Some critical information is needed to conduct the deployment process, such as (i) defining microservices, (ii) determining communication patterns between microservices, (iii) identifying the memory capacities and processors for each available node, and (iv) defining runtime execution configuration parameters for microservices. In the following subsections, generated metamodels that allow the determination of these properties are explained as concrete activities.

4.1. Microservice Data Exchange Metamodel

Microservice Data Exchange Metamodel includes data type definitions and the size of each element that is used for calculating communication costs among microservices. Although microservices are created independently, services need data exchange to form an application. This metamodel is designed to model data types and object sizes that microservices exchange between them. Different communication infrastructures such as Restful services, gRPC, etc., can be used for microservices. When we assess the level of datatype definition detail of alternative communication protocols, we see that the most comprehensive communication protocol in terms of data types is gRPC [26]. Since the gRPC protocol includes data types rich enough to model data exchange for all alternative communication patterns used for microservices, the data types belonging to gRPC proto3 [27] are used as the data type design foundation within the scope of this study.

gRPC proto3 [27] covers many data types such as message, scalar value, nested, any, oneOf, etc. There are fifteen types of scalar value types such as double, float, byte, string, fixed32, int32, etc., and message type includes these scalar values. In addition to any, oneOf and maps data types can be used for unknown fields that the parser cannot recognize. As well as its comprehensive data types, gRPC is often preferred for microservices communication since it uses HTTP/2 transfer protocol and Protobuf encoding for efficient communication.

4.2. Microservice Definition and Communication Metamodel

Microservice Definition and Communication Metamodel enables defining microservices and creating communication patterns among microservices. This model includes four different classes named REST, graphQL, gRPC, publish/subscribe, and these are the communication protocols used for microservices. Enum classes such as graphQLEnum, PubSubTypeEnum, gRPCEnum, etc., are used for defining selection properties (e.g., for publish/subscribe communication: publish = 1, subscribe = 2) belong to communication protocols. To clarify the data exchanged by microservices, source microservice and target data should be selected through the developed tool. Target data is determined using the Microservice Data Exchange Metamodel described in the previous section. A sample Emfatic language written to establish a publish/subscribe relationship between microservices is shown in Algorithm 1.

Algorithm 1 Sample Emfatic language for defining publish/subscribe relation.

```
@gmf.link(source = "source", target = "target", target.decoration = "arrow",label =
"pubSubType")
1:  class PubSubRelation {
2:      ref Microservice [1] source;
3:      attr PubSubTypeEnum [1] pubSubType;
4:      ref microDataExchangeModel.ObjectModelElement [1] target;
5:  }
```

The PubSubRelation class, defined as a relation on the metamodel, determines the data published or subscribed by a microservice. Hence, it refers to Microservice class as a source link and ObjectModelElement class as a target link. For instance, in a microservice-based e-commerce application, suppose that the Order service subscribes to the UserInfo object, and this object is published by the Customer service. In this case, the source is selected as Order service, pubSubType type is selected as subscribe, and the target object derived from the ObjectModelElement class is selected as UserInfo. In the case that a source class is selected as Customer service, pubSubType is changed as publish.

4.3. Microservice Infrastructure Metamodel

In the Microservice Infrastructure Metamodel, the processing power, memory capacity, and LAN/WAN connection types of the existing resources are determined. This metamodel is entirely independent of software business components created for the automatic deployment of microservices. Therefore, the Microservice Infrastructure Metamodel activity can be developed in parallel by different teams with the other metamodels.

4.4. Microservice Runtime Execution Configuration Metamodel

The necessary infrastructure is created to define microservices, create inter-service communication patterns, and design physical resources using the metamodels described up to this stage. Thus, the structural attributes of the proposed model-driven architecture for the deployment of microservices can be created using these metamodels. In addition to the structural characteristics of the system, information about the system's runtime behavior is also needed to be able to deploy microservices in a cost-effective manner. In the Microservice Runtime Execution Configuration Metamodel, runtime attributes, such as how many services are available in the architecture, the update rates of each data element for each publisher service (updateRate), and the execution costs of the services on physical resources (nodes), are defined.

4.5. Microservice Deployment Metamodel

Microservice Deployment Metamodel enables the deployment of microservices to related nodes. This model includes Member's and Node's classes. A Member contains one or more microservice instances that needs to be deployed on one of the Nodes defined in the Microservice Infrastructure Model. The overall model-driven architecture and relationships of the models are shown in Appendix A. All metamodels are explained in more detail in our previous study [1].

5. Case Study—Online Book Shopping

In this section, a case study related to online book shopping is adopted using the proposed model-driven architecture to validate the efficient microservice deployment approach. Four types of microservices, consisting of Account, Order, Book Inventory, and Shipping, are designed. The number of microservice instances and required memories for each instance are listed in Table 1.

Table 1. Sample scenario for an online book shopping system using number of microservice in-stances and required memory per microservice.

Microservice Name	Number of Instances	Required Memory (MB) per Instance
AccountService	300	20
BookInventoryService	200	10
OrderService	250	40
ShippingService	320	15
Total	**1070**	**22.800**

These services exchange various data such as User, Customer, Seller, CardDetails, Cancellation, CCDetails, PaymentSystem and Transaction. While calculating the communication costs among microservices, an automatic system using algorithmic approaches is critical to deploy microservices at the minimum total communication cost. To evaluate the total communication cost according to algorithmic approaches, the relations of these data designed in Microservice Data Exchange Metamodel and services defined in Microservice Definition and Communication Metamodel are defined as shown in Table 2.

Table 2. Publish/subscribe relation schema for the case study.

Microservice Name	Publishes	Publish Rate (Hz)	Subscribes
AccountService	User	5	-
	Seller	3	-
	Customer	3	-
BookInventoryService	Cancellation	4	Transaction
OrderService	CardDetails	2	PaymentSystem
ShippingService	Transaction	2	Cancellation

6. Evaluation

In this section, we demonstrate the validation of the case study since the process steps of the proposed approach for the efficient deployment of microservices are covered in our previous work [1]. The metamodels described in the previous sections appear as diagrams to the designer at runtime. While the definition of microservices and communication patterns among them are carried out using the Microservice Definition and Communication Metamodel, the nodes required for allocating services to physical resources are designed using the Microservice Infrastructure Metamodel. After these structural features of the proposed model-driven architecture are completed, the runtime execution configuration parameters of the system are also defined using the Microservice Execution Configuration Metamodel. The aim is to obtain the total minimum cost by assigning the services that are in frequent communication with each other to the same node, taking into account the execution costs of the services on the nodes. Therefore, a genetic algorithm proposed in [28] is implemented to consider communication and execution costs for microservices. Figure 2 shows a sample deployment alternative generated from the proposed model-driven architecture.

To validate the approach using the online book-shopping case study, two different CTAP [29] methods were evaluated concerning the total execution and communication cost metrics. Many algorithms suitable for the CTAP can be easily adapted as plugins to this architecture. This architecture generates efficient deployment alternatives using an algorithmic approach and is also suitable for manual deployment. Thus, the designer can manually deploy microservices, and the system enables comparison of manual and algorithmic approaches.

As seen in Table 3, total execution and communication costs vary among different algorithmic approaches. The genetic algorithm performs better than the Minimum Nodes algorithm in terms of execution cost by 14.3%. On the other hand, the Minimum Nodes algorithm has the lowest communication cost by 23% compared to the genetic algorithm. The improvement rate in terms of the communication cost is reasonable because the communication cost among services allocated on the same node is calculated as zero according to the CTAP [29]. The algorithm aims to allocate each service to the minimum number of nodes. Therefore, this algorithm utilizes all memory of a node, then starts to assign microservices on the other node until the node's memory is full.

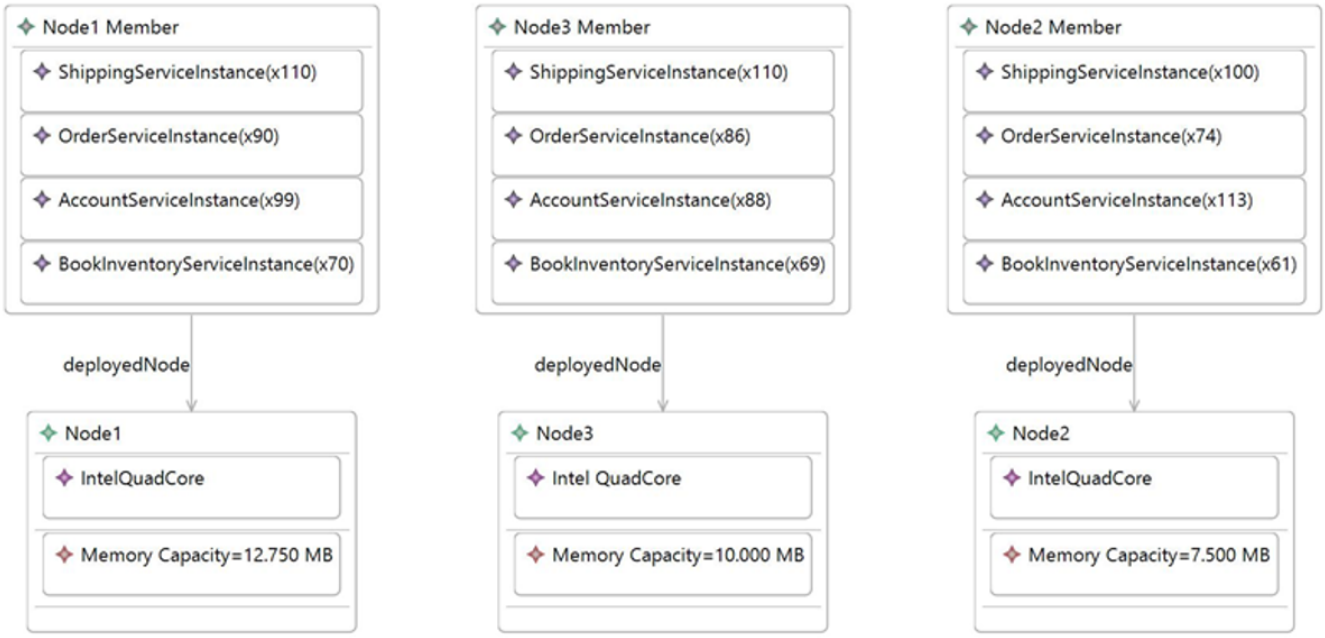

Figure 2. Generated sample deployment alternative for 1070 microservices on three nodes using a genetic algorithm.

Table 3. Total execution and communication costs according to Genetic and Minimum Nodes algorithm.

Algorithm	Communication Cost (Mbytes/s)	Execution Cost (Unit)
Genetic	17.711853	21,240
Minimum Nodes	13.634338	24,290

When we examined the processing time of the proposed algorithm for generating feasible deployment of microservices [30], we saw that the performance of the deployment process depended on the applied CTAP algorithm and the number of microservices. Therefore, the communication and execution cost for each microservice instance was calculated according to the CTAP solution. When we analyzed the formulation of CTAP problem [30], the time complexity of the total cost could be represented by $O(n^3)$ in a worst-case scenario.

7. Discussion

Model-driven engineering (MDE) enables designing software models close to the executable model at the early design phase by using the simplified abstractions of the structures in the application. The visual artifacts in MDE show the business need and ways to solve technical problems.

In this study, the logical infrastructure of the model-driven architecture developed to deploy microservices to limited capacity resources, the frameworks and the plugins used are explained. To develop the model-driven architecture for microservice deployment, we used EMF and GMF software tools in Eclipse Modeling Tools. In addition to EMF and GMF, many software tools, such as Acceleo [31], JetBrains MPS [32], Simulink [33], Sirius [34], etc., can be used for MDE. Among these models, Sirius [34] allows the user to create a complex graphic modeling workbench by using EMF [18] and GMF [19] modeling frameworks. Although this model provides a rich modeling experience, it requires the designer to create all the necessary components to use the tool. Thus, the designer needs deep knowledge about Sirius components. On the other hand, our model-driven architecture developed by using EMF, GMF and Emfatic offers a simple framework to the designer, but it requires more code generation and a more complex structure on the developer side. In this context, EMF, GMF and Emfatic stack still can be considered more advantageous than the other

alternative software tools for the designer. The Sirius tool is still promising, but it needs Emfatic-like automation overlays to define complex metamodels.

8. Conclusions

Deriving feasible deployment alternatives for microservice-based applications in the early design phase enables the efficient use of cloud resources. Current industrial management tools rely on manually created configuration files. When the generated deployment model using these tools is not satisfactory, the user needs to re-create the configuration file. This process is time-consuming, and there is no guarantee that the created deployment model will yield an efficient cost. The developed model-driven architecture allows a microservice-based application to be designed with all the details, and it permits organization of the case study in a feasible way at the beginning of the project life cycle. In addition, it enables selecting the efficient deployment model in terms of resource usage by comparing the models generated by various algorithmic approaches. Using EMF and GMF software tools, the proposed model-driven architecture allows the end-user to design microservice-based applications easily and derive efficient deployment alternatives.

Author Contributions: Conceptualization, B.T. and T.C.; methodology, A.B.C. and T.C.; software, I.K.A. and T.C.; validation, I.K.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

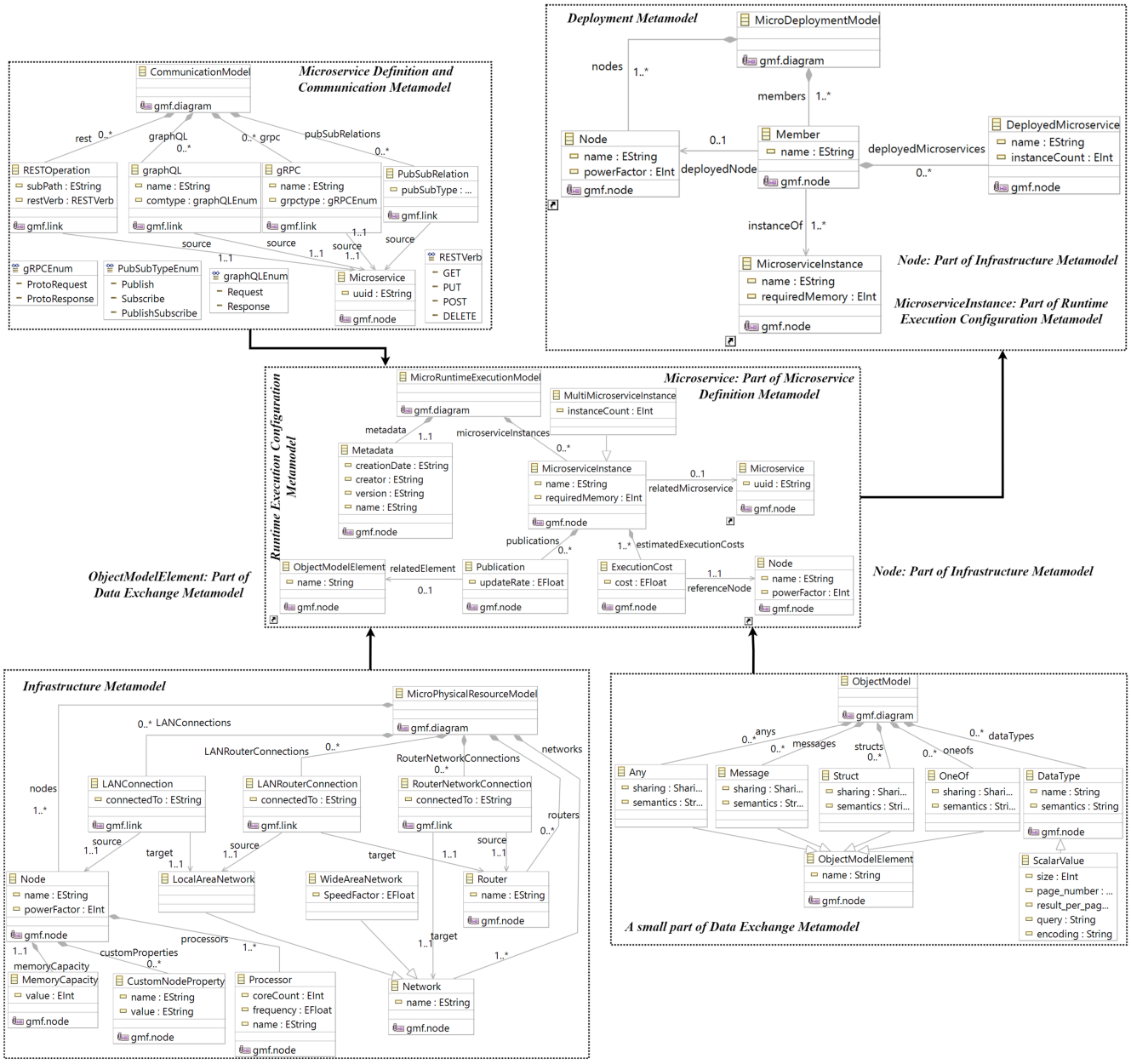

Figure A1. The overall model-driven architecture for deploying microservices.

References

1. Bravetti, M.; Giallorenzo, S.; Mauro, J.; Talevi, I.; Zavattaro, G. Optimal and automated deployment for microservices. In *International Conference on Fundamental Approaches to Software Engineering*; Springer: Cham, Switzerland, 2019; pp. 351–368.
2. Rademacher, F.; Sorgalla, J.; Wizenty, P.N.; Sachweh, S.; Zündorf, A. Microservice architecture and model-driven development: Yet singles, soon married (?). In Proceedings of the 19th International Conference on Agile Software Development: Companion, Porto, Portugal, 21–25 May 2018; pp. 1–5.
3. Granchelli, G.; Cardarelli, M.; di Francesco, P.; Malavolta, I.; Iovino, L.; di Salle, A. Microart: A software architecture recovery tool for maintaining microservice-based systems. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*; IEEE: Gothenburg, Sweden, 2017; pp. 298–302.
4. Düllmann, T.F.; van Hoorn, A. Model-driven generation of microservice architectures for benchmarking performance and resilience engineering approaches. In Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion, L'Aquila, Italy, 22–26 April 2017; pp. 171–172.

5. Ali, T.A.A.; Xiao, Z.; Sun, J.; Mirjalili, S.; Havyarimana, V.; Jiang, H. Optimal design of IIR wideband digital differentiators and integrators using salp swarm algorithm. *Knowl.-Based Syst.* **2019**, *182*, 104834. [CrossRef]
6. Zeng, F.; Li, Q.; Xiao, Z.; Havyarimana, V.; Bai, J. A price-based optimization strategy of power control and resource allocation in full-duplex heterogeneous macrocell-femtocell networks. *IEEE Access* **2018**, *6*, 42004–42013. [CrossRef]
7. Eclipse. Emfatic. Available online: https://www.eclipse.org/emfatic/ (accessed on 20 March 2021).
8. Kolovos, D.S.; García-Domínguez, A.; Rose, L.M.; Paige, R.F. Eugenia: Towards disciplined and automated development of GMF-based graphical model editors. *Softw. Syst. Modeling* **2017**, *16*, 229–255. [CrossRef]
9. Rademacher, F.; Sorgalla, J.; Sachweh, S.; Zündorf, A. A model-driven workflow for distributed microservice development. In Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, Limassol, Cyprus, 8–12 April 2019; pp. 1260–1262.
10. Çelik, T. Federe Tabanlı Koşut ve Dağıtılmış Benzetim Sistemlerinin Mimari Modellemesi. Ph.D. Thesis, Hacettepe University, Ankara, Turkey, 2013.
11. Liddle, S.W. Model-driven software development. In *Handbook of Conceptual Modeling*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 17–54.
12. Allilaire, F.; Bézivin, J.; Jouault, F.; Kurtev, I. ATL-eclipse support for model transformation. In Proceedings of the Eclipse Technology eXchange workshop (eTX) at the ECOOP 2006 Conference, Nantes, France, 3–7 July 2006.
13. Bézivin, J. On the unification power of models. *Softw. Syst. Model.* **2005**, *4*, 171–188. [CrossRef]
14. Eclipse. Eclipse Model to Text Transformation. Available online: http://www.eclipse.org/modeling/m2t/ (accessed on 21 August 2021).
15. Eclipse. Eclipse XText Project Web Site. Available online: http://www.eclipse.org/Xtext (accessed on 21 August 2021).
16. Eclipse. Grammar to Model Language. Available online: http://modelum.es/trac/gra2mol/ (accessed on 21 August 2021).
17. Izquierdo, J.L.C.; Molina, J.G. A domain specific language for extracting models in software modernization. In Proceedings of the European Conference on Model Driven Architecture-Foundations and Applications, Enschede, The Netherlands, 23–26 June 2009; pp. 82–97.
18. Eclipse. Eclipse Modeling Framework (EMF). Available online: https://www.eclipse.org/modeling/emf/ (accessed on 20 May 2021).
19. Eclipse. GMF Tooling. Available online: https://www.eclipse.org/gmf-tooling/ (accessed on 14 May 2021).
20. Eclipse. Eclipse Modeling Tools. Available online: http://www.test.org/doe/ (accessed on 2 August 2021).
21. Eclipse. Eclipse Modeling Framework Technology (EMFT). Available online: https://www.eclipse.org/modeling/emft/ (accessed on 20 March 2021).
22. Zimmermann, O. Microservices tenets. *Comput. Sci. -Res. Dev.* **2017**, *32*, 301–310. [CrossRef]
23. Neuman, S. *Building microservices: Designing fine-grained systems*; O'Reilly Media Publishing: Sebastopol, CA, USA, 2015.
24. Nadareishvili, I.; Mitra, R.; McLarty, M.; Amundsen, M. *Microservice Architecture: Aligning Principles, Practices, and Culture*; O'Reilly Media Publishing: Sebastopol, CA, USA, 2016.
25. Rademacher, F.; Sachweh, S.; Zündorf, A. Differences between model-driven development of service-oriented and microservice architecture. In Proceedings of the 2017 IEEE International Conference on Software Architecture Workshops (ICSAW), Gothenburg, Sweden, 5–7 April 2017; pp. 38–45.
26. gRPC. A high performance, open source universal RPC framework. Available online: https://grpc.io/ (accessed on 20 March 2021).
27. Language Guide (proto3). Available online: https://developers.google.com/protocol-buffers/docs/proto3 (accessed on 20 March 2021).
28. Mehrabi, A.; Mehrabi, S.; Mehrabi, A.D. An adaptive genetic algorithm for multiprocessor task assignment problem with limited memory. In Proceedings of the World Congress on Engineering and Computer Science, San Francisco, CA, USA, 22–24 October 2009; p. 115.
29. Pirim, T. *A Hybrid Metaheuristic Algorithm for Solving Capacitated Task Allocation Problems as Modified XQX Problems*; University of Mississippi: Oxford, MS, USA, 2006.
30. Aksakalli, I.K.; Celik, T.; Can, A.B.; Tekinerdogan, B. Systematic Approach for Generation of Feasible Deployment Alternatives for Microservices. *IEEE Access* **2021**, *9*, 29505–29529. [CrossRef]
31. Acceleo User Guide. Available online: https://wiki.eclipse.org/Acceleo/User_Guide (accessed on 12 August 2021).
32. Pech, V.; Shatalin, A.; Voelter, M. JetBrains MPS as a tool for extending Java. In Proceedings of the 2013 International Conference on Principles and Practices of Programming on the Java Platform: Virtual Machines, Languages, and Tools, Stuttgart, Germany, 11–13 September 2013; pp. 165–168.
33. Karris, S.T. *Introduction to Simulink with Engineering Applications*; Orchard Publications: London, UK, 2006.
34. Eclipse. Sirius. Available online: https://www.eclipse.org/sirius/ (accessed on 12 May 2021).