Dynamic Patterns for Cloud Application Life-Cycle Management

Geir Horn[1(✉)], Leire Orue-Echevarria Arrieta[2], Beniamino Di Martino[3], Paweł Skrzypek[4], and Dimosthenis Kyriazis[5]

[1] University of Oslo, P.O. Box 1080 Blindern, 0316 Oslo, Norway
Geir.Horn@mn.uio.no
[2] Fundacion TECNALIA Research and Innovation, Derio, Spain
Leire.Orue-Echevarria@tecnalia.com
[3] University of Campania "Luigi Vanvitelli", Caserta, Italy
beniamino.dimartino@unina.it
[4] 7Bulls.com, Al. Szucha 8, 00-582 Warsaw, Poland
pskrzypek@7bulls.com
[5] University of Piraeus, Piraeus, Greece
dimos@unipi.gr

Abstract. Cloud applications are by nature dynamic and must react to variations in use, and evolve to adopt new Cloud services, and exploit new capabilities offered by Edge and Fog devices, or within data centers offering Graphics Processing Units (GPUs) or dedicated processors for Artificial Intelligence (AI). Our proposal is to alleviate this complexity by using *patterns* at all stages of the Cloud application life-cycle: deployment, automatic service discovery, monitoring, and adaptive application evolution. The main idea of this paper is that it is possible to reduce the complexity of composing, deploying, and evolving Cross-Cloud applications using dynamic patterns.

1 Introduction

The question is not if Cloud computing should be used, but how: There are concerns about private data, and consequently the simultaneous use of private and public Cloud; there are questions about vendor lock-in and portability; there are questions about the best deployment models, like deploy a Virtual Machine (VM) and a database in that machine, or use a database as a service offered in the Cloud; there are questions about scalability and maintenance of the deployed application as application use and Cloud offerings evolve over time. Furthermore, applications in the future will need to relate to major IT-trends[1] like wearable devices and sensors, mobility of users, and strong security requirements.

[1] https://www.iqvis.com/blog/cloud-computing-predictions-2020/.

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 731664 MELODIC: Multi-cloud execution-ware for large-scale optimised data-intensive computing; and grant agreement No. 731533 DECIDE: Multicloud applications towards the digital single market.

© Springer Nature Switzerland AG 2020
L. Barolli et al. (Eds.): 3PGCIC 2019, LNNS 96, pp. 626–637, 2020.
https://doi.org/10.1007/978-3-030-33509-0_59

Today, a decade into the era of Cloud computing, the situation is similar to the one faced a decade into the era of object-oriented programming when software systems grew in complexity making them exponentially harder to develop and maintain. There was a need for *abstractions* [30] and the identification of *best practices*, which gave raise to what became known as *design patterns* [10]. With growing software system complexity this again led to *architectural patterns* [26], and recently to *microservices patterns* [6] to help the design of distributed applications in the Cloud.

Even though such patterns capture the best practices and may help the design and initial deployment of Cloud applications, the current state of the art fails to capture the dynamic nature of Cloud applications. The Cloud deployment must react to variations in use, adopt new Cloud services, and exploit new capabilities offered by Edge and Fog devices, or within data centers offering GPUs or dedicated processors for AI. The Cloud services, virtual and physical resources, and their combinations – in the core Cloud, at the Edge, or in the Fog – will herein be referred to collectively as *Cloud capabilities*.

Autonomic computing [20] applied to the application Cloud deployment can remedy some of the concerns above. This requires a continuous feedback control loop: Monitor, Analyse, Plan, Execute—with Knowledge (MAPE-K) [17]. There are already successful utility based approaches to autonomic application configuration management in context aware mobile computing [22] and ubiquitous computing systems [28]. In the novel Multi-cloud Execution ware for Large scale Optimised Data Intensive Computing[2] (MELODIC) framework these approaches are extended to Cross-Cloud autonomic application deployment and run-time management [14]. These approaches assume that the application to be deployed can be modeled as a set of interconnected components or *objects* [25], and there are many frameworks based on dialects of cloud modeling languages [2]. MELODIC exploits the application model in The Cloud Application Modelling and Execution Language (CAMEL) [1], and can thus be seen as a complete *modelsrun.time* [16] framework that has already been successfully applied to several demanding Computational Intelligence (CI) [19] applications [15].

An issue with this approach is the Planning part of the MAPE-K loop since finding the best application configuration for a given execution context is a combinatorial optimization problem whose time complexity is typically exponential in the number of application components, and possibly also the Cloud capabilities offered by the Cloud providers that may host the application's components. The complexity of the problem remains hard even though it is possible in many cases to find algorithms that work well for smaller configurations, *e.g.* an adapted and improved version of the Nondominated Sorting Genetic Algorithm II (NSGA-II) [21] is used in the DECIDE[3] project to optimize the problem of selecting the most appropriate cloud services for an application that needs to comply with a set of non-functional requirements [24].

[2] https://melodic.cloud/.
[3] https://www.decide-h2020.eu/.

However, a better approach to deal with the complexity could be to identify statically the application topology and architecture as a software pattern. The application topology pattern sub-graphs could then be mapped onto known classes of deployment patterns. This would reduce the problem at run-time to *select* the most suitable deployment patterns for the current application execution context. This paper will explore this vision. Section 2 will show how the application can be statically reduced to a set of patterns. Section 3 discusses Cloud deployment patterns and how they can be used for fast deployment decisions. The application life-cycle considerations are discussed in Sect. 4, and Sect. 5 elaborates on the consequences of using patters for application management and adaptation.

2 Software Patterns and Semantics

There are often practical issues which limit portability and interoperability of legacy or even native Cloud applications [4]: different data formats, parameters semantics, unclear descriptions of the exposed Application Programming Interfaces (APIs), and so on. Furthermore, many vendors try to bind their customers to their own platform, making it difficult or expensive for them to port their applications to another environment when needed An effective approach, enabling automated reasoning, is the adoption of *semantic representations*, and specifically *ontologies*, which is a formal, machine readable knowledge representation by means of a set of domain related concepts and the relationships between those concepts. The Web Ontology Language[4] (OWL) is a semantic mark-up language for publishing and sharing ontologies on the World Wide Web (WWW). A number of ontologies related to Cloud computing emerged in the past few years. Androcec *et al.* [3] provides an overview of Cloud Computing ontologies, their types, applications and scope. Deng *et al.* [9] presents a formal catalog representation of Cloud services that model, with ontologies, a range of Cloud services and their processes. Takahashi *et al.* [29] use ontologies to describe cybersecurity operational information such as data provenance and resource dependency information.

There are no formal, standard, and universally accepted languages to describe design and Cloud patterns in a uniform manner. Semantic based languages have been proposed in literature to formalize and categorize patterns: Dietrich and Elgar defined an ontology based model, called Object Design Ontology Layer (ODOL) [18] that defines a series of OWL classes and properties to describe patterns, focusing on the description of the application context of a pattern, analyzing its possible uses and identifying real implementation of its participants [18]. On the other hand, it neglects other aspects of patterns, like their behavior or the different relationships existing among their participants, thus losing expressiveness.

The mOSAIC project has developed Cloud services and patterns description based on semantic technologies [5]. The defined Semantic Model – based

[4] https://www.w3.org/TR/owl2-overview/.

on standard World Wide Web Consortium[5] (W3C) semantic representation languages OWL and Semantic Markup for Web Services[6] (OWL-S), and on their integration with the ODOL model – is able to represent in a machine readable representation Cloud Services – with their functional and non-functional features – and Cloud Patterns, which represent correlations and composition of Cloud services according to well established design, architectural and process patterns. The mOSAIC Cloud Ontology has been adopted by the Institute of Electrical and Electronics Engineers[7] (IEEE) Standard for Intercloud Interoperability and Federation[8] (SIIF).

Figure 1 shows the Semantic model as a graph, structured into five conceptual layers. The graph represents concepts (graph nodes) and relationship (graph edges) at different layers. In each layer relationships among concepts of the same layer are represented, in addition to inter-layer relationships.

The Application Software Architectural Patterns layer, labeled (4) in Fig. 1, represents the description of patterns describing the application to be deployed on a platform where an application pattern represents a composition of application components embodying application domain functionality. The Cloud Patterns layer, layer (3) in Fig. 1, represents the semantic description of agnostic and vendor dependent patterns. It represents patterns at the Infrastructure as a service (IaaS) and at the Platform as a Service (PaaS) level. The Services layer, layer (2) in Fig. 1, represents the semantic annotation of the provider dependent platform services and the supporting ontologies needed to identify the platform provider supported operation, input and output parameters. This layer presents details of the provider platform architecture, the functionality exposed and the underlining details. This layer contains also the semantic description of the agnostic platform services exposed through an ontology that exposes in vendor neutral terms platform resources, operations and exchanged parameters. This layer is grounded onto the two underneath layers - representing the grounding of the semantic representation, and following the Web Service Description Language (WSDL) standard: the Operations layer, layer (1) in Fig. 1, that represents the syntactic description of the operation and functionality exposed by the platform services, and provides a machine-readable description of how the service can be called, what parameters it expects, and what data structures it returns; and the Parameters layer (layer 0 in Figure) which represents the description of the data type exchanged among services as input and output of the operations.

3 Deployment Patterns

Microservices are a key pattern both in terms of software engineering and in terms of software management in terms of deployment and configuration of

[5] https://www.w3.org/.
[6] https://www.w3.org/Submission/OWL-S/.
[7] https://www.ieee.org/.
[8] https://standards.ieee.org/project/2302.html.

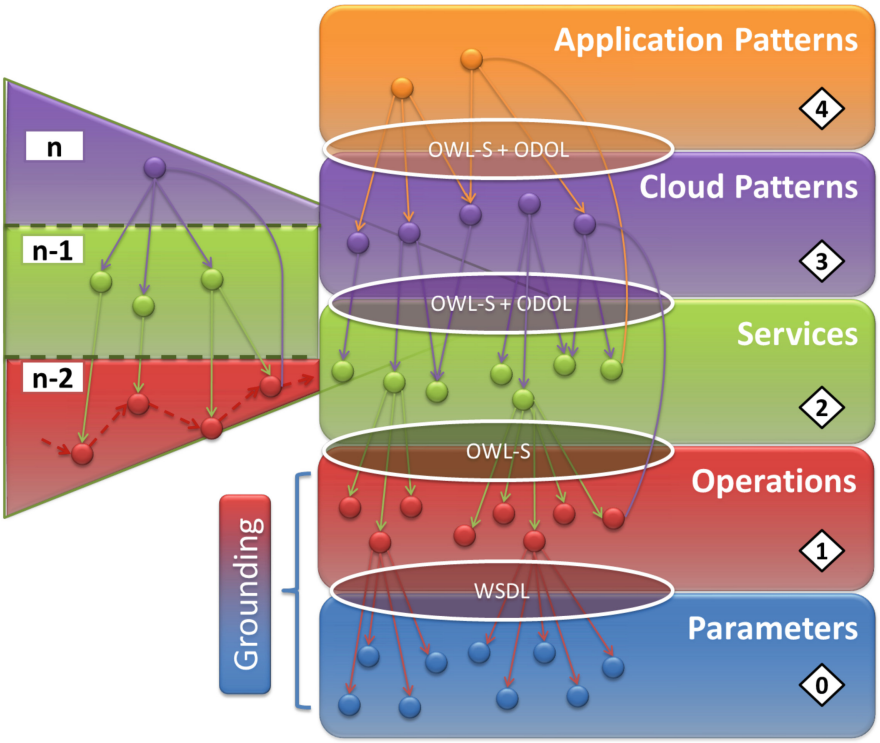

Fig. 1. The Semantic Model as a graph with five layers highlighting the generic and provider agnostic side, and the platform and vendor specific side. The semantic representations of both sides are fundamental for defining the equivalence among the different levels and to enable semantic inference and matchmaking within a coherent pattern based methodology.

different services that contribute to an overall application. Microservices refer to atomic and individual services with a single but well-defined aim, that are deployed autonomously [8] and in composition formulate a service (i.e. application). Given that each microservice is deployed individually, the overall deployment of an application raises the challenge of how different microservices are deployed, *i.e.* how the overall deployment of the application can be optimized. The latter becomes even more challenging in cloud infrastructure deployments given the distributed underlying infrastructures and the different types of resources that can be provided, *e.g.* VMs, containers, or bare metal; and the way these resources are connected, *e.g.* bandwidth and latency.

Deployment patterns regard the process of identifying the optimal actions and practices that will automate the deployment and execution of services in a computing infrastructure, such as a computing Cloud [7], meaning how the different microservices are deployed in the same VM for example or in different VMs with specific connection requirements and constraints between these

VMs. Amato *et al.* [12] propose a pattern-based orchestration methodology that describes the composition of a Cloud service and provides information regarding the way the service should be deployed. The same authors suggest that a way to orchestrate more efficiently the computing infrastructure and deploy the services on top of them is related to the proper description of the services [13]. When the services are properly defined, with respect to their needs, then the deployment is adjusted in order to achieve higher performance. Yamato proposed a platform that analyses the computing infrastructure usage during deployment in order to achieve high performance [31]. Still, the challenge of the proper placement of the services in heterogeneous environments such as a computing Cloud is valid.

Furthermore, there are several frameworks that aim at addressing the aforementioned deployment patterns and configurations challenge. Octopus Deploy[9] is a tool that is used for the deployment of services in cloud computing infrastructures and automates the process of the application deployment. Through Octopus Deploy run-time adaptation can be achieved since it understands possible infrastructure changes and dynamically adjust. IBM's WAS[10] deployment manager is a yet another production tool whose purpose regards the administration and orchestration of application services in WAS deployment shells. In addition to the above-mentioned tools, Google Cloud Deployment manager[11] regards a deployment service that speeds up the configuration and orchestration of services hosted in Google Cloud Platform. The Runtime Configurator of this Deployment manager allows the service provider to dynamically configure services.

One of the key enablers for optimized deployment patterns are the software patterns and semantics, which provide the ground for describing and characterizing the microservices to be deployed. Approaches aim at analyzing the capabilities of the microservices through their semantic descriptions, and the associated requirements in order to identify the deployment patterns in Cloud environments by utilizing information about the current state of the Cloud infrastructures where the services are to be deployed, the foreseen availability of resources, the failure estimations, etc. At the same time such approaches must take additional aspects into consideration, such as the interdependent services and the data and network dimensions. An additional aspect to deployment patterns generation and configuration refers to their adaptation during run-time given the services evolution, the utilization of specific microservices of the application, and the infrastructure evolution. The latter enables the provision of adaptive deployments through the identification and dynamic adaptation of the deployment patterns.

[9] https://octopus.com/.

[10] https://www.ibm.com/support/knowledgecenter/en/linuxonibm/liaag/wecm/l0wecm00_was_deployment_manager.htm.

[11] https://cloud.google.com/deployment-manager/.

4 Cloud Application Life-Cycle

The application life-cycle of multi- and cross-Cloud native applications has varied to what the literature in software engineering reported. The development of Cloud native applications involve shorter and faster development cycles, and the operation activities have now a more prominent role than before. Users expect Cloud applications to be always available and perform well, which implies that application providers need to respond quicker to malfunctions or bottlenecks. This has created the need to merge both the development and operation teams into one, namely the DevOps [27], which aim at being able to continuously 'architect', develop, integrate, test, deploy and operate the application. Furthermore, in order to understand what needs to be improved in the application and to provide always a responsive and available solution, means have to be provided to monitor the different components of the application and be able to deploy automatically, these components into a new configuration that responds to the user's needs and expectations.

The development of multi- and cross-Cloud native applications deployed on a hybrid scenario pose several challenges, both in development and operations time such as:

- *Applications need to be always responsive* in terms of availability and performance, among other aspects, under this hybrid Cloud configuration, especially when the micro services of the application are deployed on different Cloud resources from different Cloud Service Providers (CSP). To achieve that, the health and conditions of the application micro services should be continuously monitored in order to be able to respond to malfunctions or inefficiencies at operation time. To this end, and to ensure a complete availability, the application shall be able to self-adapt itself and redeployed in a new appropriate configuration.
- *Communication in terms of information exchange among the micro services and different components of the multi-Cloud application*: As systems evolve and grow, the management of endpoints becomes cumbersome. To this end, there are tools, *e.g.* Kubernetes[12] that support the orchestration of containers and can manage such endpoints, can be of assistance, but the learning curve is high.
- *Vendor lock-in*: Each CSP offers its own technology, libraries and frameworks that should be used when deploying an application on their services. Furthermore, porting data and applications from one Cloud service to another can be challenging from the technical and regulatory point of view, although the container approach eases this task to a great extent. Moreover, article 6 of the new regulation of Free Flow of Data [11] aims to facilitate this, through the adherence of the CSPs to the code of conducts currently being developed.
- *Discovery, selection, management of the most appropriate Cloud service or combination of cloud services for a specific application*: The selection of cloud services is currently more and more complex taking into consideration the

[12] https://kubernetes.io/.

plethora of offerings and configurations there exist, this activity becomes even more challenging. To alleviate this, cloud service brokers such as Cloudmore[13], Computenext[14], Nephos hybrid Cloud management[15], Intercloud[16], IBM cloud brokerage solutions (previously Gravitant[17]) and Jamcracker[18], come into play. However, these solutions do not present an easy way to select Cloud services or do not consider the combination of multiple Cloud resources to respond to the needs of multi-Cloud native applications. This situation can also be extended to the monitoring of the non-functional characteristics of the different Cloud resources. An initial solution to respond to this challenge has been implemented in the DECIDE project[19], as part of the Advanced Cloud Service meta-Intermediator (ACSmI).

5 Application Adaptation

There are many types of Cloud applications. Our vision assumes that the application will run over quite some time, and the deployment must therefore consider the various factors indicated in the previous Sect. 4 life-cycle discussion. Additionally, the application may have intrinsic variability caused by the data it processes or caused by the application users. Consider for instance an airline industry application responsible for scheduling planes, crews certified for some type of planes, and the passengers. It must be available around the clock, all days around the year. It may experience variation in demand and use between day and night, and between week days and weekends. It may see changes in users' location as the earth rotates. Finally, if there is a major incident closing a major airport hub, there may temporary be a large demand for resources to re-schedule planes, crews and passengers to clear this exceptional situation as quickly as possible. All of these situations call for the application to be *adaptive* to its current *execution context*.

It will be costly and error prone to have a human DevOps team constantly monitoring the application and exploiting the elasticity of the Cloud computing paradigm to provide or to remove resources as the demand fluctuates. Current, autonomic approaches based on the MAPE-K loop are *reactive*. In other words, when some Complex Event Processing (CEP) shows that the monitored sensors indicate a change in the application's execution context in the analysis phase, there must be an efficient *planning* phase finding a better deployment solution and *executing* this solution before the application context has changed again. The planning phase typically involve solving a combinatorial optimization problem whose time complexity grows exponentially with the number of factors to

[13] https://web.cloudmore.com/.
[14] https://www.computenext.com/platform/enterprise-cloud-brokerage.
[15] http://www.nephostechnologies.com/technology/hybrid-cloud-management/.
[16] https://www.intercloud.com/platform/overview.
[17] https://www.ibm.com/us-en/marketplace/cloud-brokerage-solutions.
[18] https://www.jamcracker.com/.
[19] https://www.decide-h2020.eu.

consider for the next deployment. Using *anytime* algorithms will produce the new deployment configuration solution within the time window of stability for the current execution context. However, this may severely impact the solution quality, and consequently the usefulness of the adapted deployment.

Consequently, it is paramount to reduce the number of factors to consider in the planning phase. Knowing the software patterns of the application, and their surjective mapping onto architecture patterns, and the surjective mapping of these again onto deployment patterns will reduce the decision problem to *select* the deployment *pattern* that is best suited for the application's current context. This is essentially a *sorting* problem that can be solved in polynomial time. Consider as an example a very simple application with a front-end web-server and a back-end database. Some deployment patterns can be pre-computed for these components: They can be co-located on the same VM, they can be on separate VMs, and the web-server and the database can both be hosted as services offered by the Cloud platform provider. If cost is the main concern, the current cost of using any of these three deployment patterns can quickly be retrieved from the potential Cloud providers, and the less expensive pattern is selected and enacted.

6 Discussion

The significant rise of Cloud computing models, services and solutions [23], in conjunction with automatic adaptation platforms create a very complex environment for the development and maintenance of modern applications. Also, modern applications are becoming more complex and integrated with various sources of data in different ways. Applications may process huge amount of data in real-time, based on data streams or complex data lakes. On the other hand, even when using Cloud computing, the cost of computing resources is a significant element of the whole Total Cost of Ownership (TCO) of an Information Technology (IT) system. The rising complexity of modern applications requires dedicated methods to handle this issue. As presented in this paper, by using well defined, described, implemented, and tested patters, it is possible to reduce the complexity of modern Cloud applications. It is very important to distinguish the Cloud deployment patterns and software development patters. The first ones are dedicated patters representing application architecture, which handle typical situation for the deployment into the Cloud. Use of these predefined patterns limits the impact of the complexity of deployment. Also, there is limited need for testing configurations based on these patterns, as they have already been deployed and tested. On the other hand, it is still the issue of selecting the right pattern in the certain moment, especially for some applications where different patterns must be deployed at a various levels of applications use, *e.g.* high workload or low workload. Increasing workload would sometimes even require to completely redeploy the application with a different deployment architecture.

As automatic application management platforms become standard for the deployment of modern, complex applications, the importance of deployment

patterns is increasing. Automatic adaptation platforms need exploit predefined, well-described, and tested patterns; and select the best one for the particular situation. Without patterns, it would be very difficult to use these platforms, since one must define and hard code, for each situation, a particular type of deployment. Using patterns, one can measure the effectiveness of each deployment and learn which pattern is the most suitable for the given or similar situations. According to our knowledge, the only competitive approach using patterns for Cloud deployments is using one deployment pattern for all situations, and hard code it into the application configuration. This is a less flexible approach and requires more effort for implementation and testing that the approach outlined here with automatic application adaptation. Also, the static use of patterns limits the potential benefit of automatic adaptation, as adaptation would be limited to the size of resources, and would not be allowed to change deployment architecture.

When automatically selecting the best deployment patterns for the given situation, the automatic adaptation platform is able not only to adjust the amount of resources given to the application, but also to find more optimal deployment architecture. We are expecting a rising number of Cloud deployment patterns, as the complexity of modern applications will be rising. Also, the increasing number of Cloud services and models will support this trend, in conjunction with wider usage of automatic adaptation platforms.

7 Conclusion

In this paper we have devised and illustrated a preliminary methodology to support the complex Cloud application life-cycle consisting of designing, developing, composing and deploying Cloud applications in a multiservice and multiplatform scenario, which include possible edge and fog integration. Our proposal is to use extensively *patterns* at all stages. We are investigating techniques for automating the enactment of the different stages of the life-cycle, with use of machine readable semantic representation of patterns and service, inference, and machine learning techniques.

References

1. Rossini, A., Kritikos, K., Nikolov, N., Domaschka, J., Griesinger, F., Seybold, D., Romero, D., Orzechowski, M., Kapitsaki, G., Achilleos, A.: The cloud application modelling and execution language (CAMEL). OPen Access Repositorium der Universität Ulm, p. 39 (2017). https://doi.org/10.18725/OPARU-4339
2. Bergmayr, A., Rossini, A., Ferry, N., Horn, G., Orue-Echevarria, L., Solberg, A., Wimmer, M.: The evolution of CloudML and its applications. In: Paige, R., Cabot, J., Brambilla, M., Hill, J.H. (eds.) Proceedings of the 3rd International Workshop on Model-Driven Engineering on and for the Cloud 18th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2015), vol. 1563, pp. 13–18. CEUR Workshop Proceedings, Ottawa (2015). http://ceur-ws.org/Vol-1563/

3. Androcec, D., Vrcek, N., Seva, J.: Cloud computing ontologies: a systematic review. In: The Third International Conference on Models and Ontology-based Design of Protocols, Architectures and Services, MOPAS 2012, pp. 9–14 (2012)
4. Di Martino, B., Cretella, G., Esposito, A.: Cloud Portability and Interoperability - Issues and Current Trends. Springer, Berlin (2015). https://doi.org/10.1007/978-3-319-13701-8
5. Beniamino, D.M., Antonio, E., Giuseppina, C.: Semantic representation of cloud patterns and services with automated reasoning to support cloud application portability. IEEE Trans. Cloud Comput. **5**(4), 765–779 (2017). https://doi.org/10.1109/TCC.2015.2433259
6. Richardson, C.: Microservices Patterns: With Examples in Java, 1st edn. Manning Publications, Shelter Island, New York (2018)
7. Namiot, D., Sneps-Sneppe, M.: On micro-services architecture. **2**(9), 24–27 (2014). http://injoit.org/index.php/j1/article/viewFile/139/104
8. Taibi, D., Lenarduzzi, V., Pahl, C.: Architectural patterns for microservices: a systematic mapping study. In: CLOSER 2018, pp. 221–232 (2018). https://pdfs.semanticscholar.org/f6e8/8823482de1729584acbfb450d4502f4d393d.pdf
9. Deng, Y., Head, M., Kochut, A., Munson, J., Sailer, A., Shaikh, H.: Introducing semantics to cloud services catalogs. In: 2011 IEEE International Conference on Services Computing (SCC), pp. 24–31 (2011)
10. Gamma, E., Helm, R., Johnson, R., Vlissides, J., Booch, G.: Design Patterns: Elements of Reusable Object-Oriented Software, 1st edn. Addison-Wesley Professional, Reading (1994)
11. European Union: Regulation (EU) 2018/1807 of the European Parliament and of the Council of 14 November 2018 on a framework for the free flow of non-personal data in the European Union (text with EEA relevance) (2018). http://data.europa.eu/eli/reg/2018/1807/oj
12. Amato, F., Moscato, F.: Pattern-based orchestration and automatic verification of composite cloud services. **56** (2016). https://www.sciencedirect.com/science/article/pii/S0045790616302026
13. Amato, F., Moscato, F.: Exploiting cloud and workflow patterns for the analysis of composite cloud services. Future Gener. Comput. Syst. **67**, 255–265 (2017)
14. Horn, G., Skrzypek, P.: MELODIC: utility based cross cloud deployment optimisation. In: 32nd International Conference on Advanced Information Networking and Applications (AINA) Workshops, pp. 360–367. IEEE Computer Society, Krakow (2018). https://doi.org/10.1109/WAINA.2018.00112
15. Horn, G., Skrzypek, P., Materka, K., Przeździek, T.: Cost benefits of multi-cloud deployment of dynamic computational intelligence applications. In: Barolli, L., Takizawa, M., Xhafa, F., Enokido, T. (eds.) Proceedings of the Workshops of the 33rd International Conference on Advanced Information Networking and Applications (WAINA-2019). Advances in Intelligent Systems and Computing, vol. 927, pp. 1041–1054. Springer, Matsue (2019). https://doi.org/10.1007/978-3-030-15035-8_102
16. Blair, G., Bencomo, N., France, R.B.: Models@run.time. Computer **42**(10), 22–27 (2009). https://doi.org/10.1109/MC.2009.326
17. IBM: An architectural blueprint for autonomic computing. White Paper Third Edition, IBM, 17 Skyline Drive, Hawthorne, NY 10532, USA (2005). http://www-03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf
18. Dietrich, J., Elgar, C.: A formal description of design patterns using OWL. In: 2005 Australian Software Engineering Conference, pp. 243–250. IEEE Computer Society, Brisbane (2005). https://doi.org/10.1109/ASWEC.2005.6

19. Kacprzyk, J., Pedrycz, W. (eds.): Springer Handbook of Computational Intelligence. Springer Handbooks. Springer, Heidelberg (2015)
20. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer **36**(1), 41–50 (2003). https://doi.org/10.1109/MC.2003.1160055
21. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002). https://doi.org/10.1109/4235.996017
22. Geihs, K., Barone, P., Eliassen, F., Floch, J., Fricke, R., Gjørven, E., Hallsteinsen, S., Horn, G., Khan, M.U., Mamelli, A., Papadopoulos, G.A., Paspallis, N., Reichle, R., Stav, E.: A comprehensive solution for application-level adaptation. Softw.: Pract. Exp. **39**(4), 385–422 (2009). https://doi.org/10.1002/spe.900
23. Maciaszek, L.A., Skalniak, T.: Confluent factors, complexity and resultant architectures in modern software engineering: a case of service cloud applications. In: 5th International Symposium on Business Modeling and Software Design, BMSD 2015, pp. 37–45. SciTePress (2015)
24. Arostegi, M., Torre-Bastida, A., Bilbao, M.N., Del Ser, J.: A heuristic approach to the multicriteria design of IaaS cloud infrastructures for big data applications. Expert Syst. **35**(5), e12259 (2018). https://doi.org/10.1111/exsy.12259
25. Ferry, N., Chauvel, F., Song, H., Rossini, A., Lushpenko, M., Solberg, A.: CloudMF: model-driven management of multi-cloud applications. ACM Trans. Internet Technol. (TOIT) **18**(2), 16:1–16:24 (2018). https://doi.org/10.1145/3125621
26. Avgeriou, P., Zdun, U.: Architectural patterns revisited - a pattern language. In: Longshaw, A., Zdun, U. (eds.) Proceedings of the 10th European Conference on Pattern Languages of Programs (EuroPLoP 2005), vol. D3, pp. 1–39. UVK - Universitaetsverlag Konstanz, Irsee (2005)
27. Jabbari, R., bin Ali, N., Petersen, K., Tanveer, B.: What is DevOps? A systematic mapping study on definitions and practices. In: Proceedings of the Scientific Workshop Proceedings of XP2016, XP 2016 Workshops, pp. 12:1–12:11. ACM, Edinburgh (2016). https://doi.org/10.1145/2962695.2962707
28. Hallsteinsen, S., Geihs, K., Paspallis, N., Eliassen, F., Horn, G., Lorenzo, J., Mamelli, A., Papadopoulos, G.A.: A development framework and methodology for self-adapting applications in ubiquitous computing environments. J. Syst. Softw. **85**(12), 2840–2859 (2012). https://doi.org/10.1016/j.jss.2012.07.052
29. Takahashi, T., Kadobayashi, Y., Fujiwara, H.: Ontological approach toward cybersecurity in cloud computing. In: Proceedings of the 3rd International Conference on Security of Information and Networks, pp. 100–109. ACM (2010)
30. Cunningham, W., Beck, K.: Constructing abstractions for object-oriented applications. Technical report CR-87-25, Tektronix, Inc, Computer Research Laboratory (1987)
31. Yamato, Y.: Optimum application deployment technology for heterogeneous IaaS cloud. Inform. Process. Soc. Jpn. **25**, 56–58 (2017)