F(X)-MAN: An Algebraic and Hierarchical Composition Model for Function-as-a-Service

Chen Qian* and Wenjing Zhu†
*School of Computer Science and Technology, Donghua University
†Institute of Scientific and Technical Information of Shanghai, Shanghai Library
Email: *chen.qian@dhu.edu.cn, †wjzhu@libnet.sh.cn

***Abstract*—Function-as-a-Service promises a new era in which functionalities are implemented, executed and managed on a cloud platform with the aim of developing and launching applications. This paper puts forward an algebraic and hierarchical model that has ability to construct composite services for cloud applications. Our model brings many advantages, which are also presented in this paper.**

Keywords—Function-as-a-Service; algebraic composition; service model.

I. Introduction

Over the past decade, cloud computing has been developed at an unprecedented speed, whilst cloud applications have increasingly penetrated all areas of industry [1]. Cloud computing service model allows user to tailor the off-the-shelf services and adopt them nearly immediately, in spite of sometimes such a service does not completely customize to a user's application.

In the context of cloud computing, conventional services are developed and deployed in the form of monolithic computation units, in which all the code is interwoven into one large piece [2]. As a result, the monolith hinders the scalability and efficiency of client applications, especially when the number of participant services is increased. To address the issue, Function-as-a-Service (FaaS) is proposed, by means of structuring a *serverless architecture* that decomposes software applications into fine-grained functions [3]. Such functions are further invoked remotely at runtime.

At present, FaaS providers do not pay more attention to the service modeling. Unfortunately, the lack of modeling possibly (i) impairs the understanding of the system, (ii) blurs the details of software design, and (iii) obstructs quick and frequent changes in high levels of abstraction.

In this paper, we present a novel model specified for FaaS, which composes services in an algebraic and hierarchical manner.

II. Related Work

With the growing of cloud computing, the concept of "everything is a service" has been formulated, such as *Platform-as-a-Service* (PaaS), *Infrastructure-as-a-Service* (IaaS) and *Software-as-a-Service* (SaaS) [4].

DOI reference number: 10.18293/SEKE2020-024

IaaS provides a base infrastructure for sale, e.g., virtual machines and repositories, on which user must configure and manage a platform before deploy applications on it. PaaS provides a platform which is already installed in the infrastructure. Hence, end user can develop and deploy the applications based on the platform. In comparison with the first two cloud services, SaaS is simpler. It enables cloud applications for direct use.

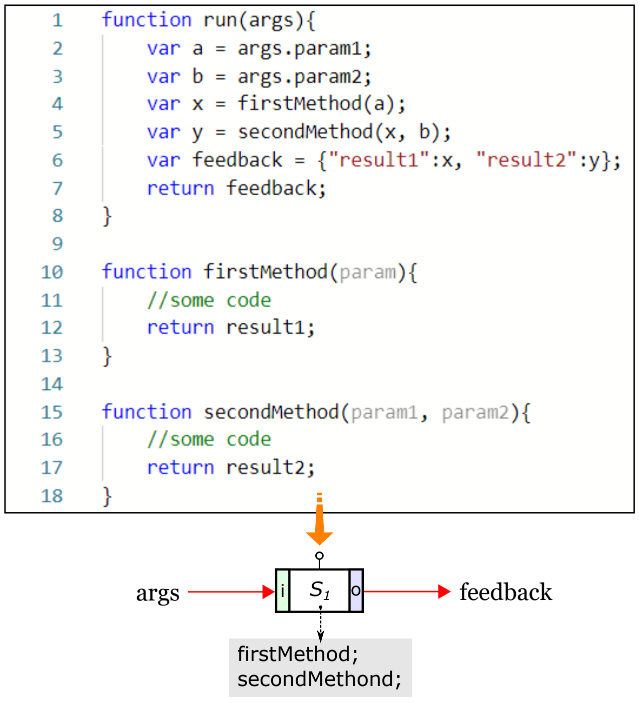
FaaS is even more flexible than PaaS. It allows developer to 'assemble' an application from functions on the cloud. However, current FaaS providers such as AWS Lambda, Google Cloud Functions, Microsoft Azure Functions and so forth focus on the language support, single function execution time and other properties, instead of the composition mechanism and underlying modeling methodology.

III. F(X)-MAN Service Model

In this section, we put forward a service composition model, called F(X)-MAN, which is inspired by X-MAN component model and its extensions [5]–[9]. In F(X)-MAN, we regard both services and *exogenous* connectors as first-class entities. Figure 1 illustrates the F(X)-MAN constructs which we further describe below.

The F(X)-MAN model defines two types of services: *atomic* or *composite* service, which are demonstrated in Figure 1(a) and 1(b), respectively. An atomic service encapsulates a set of methods in the form of an input-output function with a purpose that different services can access, whereas a composite service consists of sub-services (atomic or composite) composed by exogenous connectors, which coordinate control flows between sub-services from the outside. Therefore, services are unaware they are part of a larger piece of behavior, which become perfectly suitable for FaaS. Ideally, the services do not have to be implemented by the same programming language.

Notably, F(X)-MAN model defines *algebraic service composition* [10]. This idea is enlightened by algebra where functions are composed hierarchically into a new function of the same type. Similarly, in F(X)-MAN, we utilize an exogenous connector as an operator to hierarchically compose multiple sub-services into a bigger service, while the resulting service can be further composed with other services, yielding a more complex one. The algebraic nature brings an advantage that F(X)-MAN services can be designed, implemented, deployed and remotely invoked with high flexibility. For instance, as

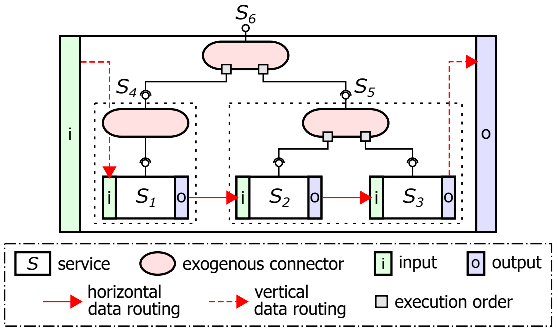


```
function run(args){
    var a = args.param1;
    var b = args.param2;
    var x = firstMethod(a);
    var y = secondMethod(x, b);
    var feedback = {"result1":x, "result2":y};
    return feedback;
}

function firstMethod(param){
    //some code
    return result1;
}

function secondMethod(param1, param2){
    //some code
    return result2;
}
```

(a) Atomic service.



(b) Composite service.

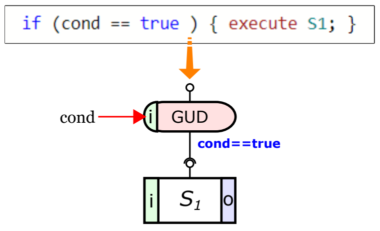
Fig. 1. F(X)-MAN: Service model.

shown in Figure 1(b), when we deploy composite service S_6 in a network, in fact all its sub-services, no matter atomic (S_1, S_2, S_3) or composite (S_4, S_5), are exposed and ready for use.

In order to simulate the statements in computer programming, we define three categories of exogenous connectors in F(X)-MAN. *Composition connectors* compose multiple services by coordinating control flows among them, whereas *adaptation connectors* are applied to individual services with the aim of adapting the received controls. In addition, a *parallel connector* is specified to handle the concurrent invocation of services.

We hereby list the most commonly used statements along with their related connectors.

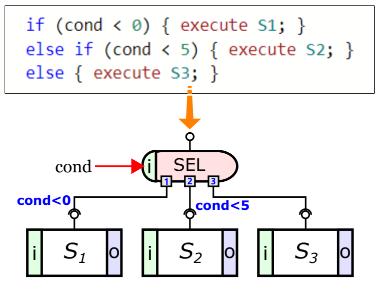
- **If statement.** The *if statement* (sometimes called *if-then statement*) is common across major programming languages [11]. If the expression is evaluated to true, statements inside the body of *if* are executed. Accordingly, we define an adaptation connector called *guard* that allows control to reach a service only if the condition is satisfied, as Figure 2 shows.



```
if (cond == true ) { execute S1; }
```

Fig. 2. F(X)-MAN: Guard.

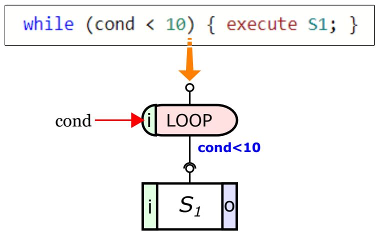
- **If-else statement.** In *if-else statement* (sometimes called *if-elif-else statement*), only the statements following the first expression that is evaluated to true will be executed [12]. Thus, we define a composition connector, namely *selector*, with the aim of branching. Figure 3 presents an example. If the input of selector has a value of 6, S_3 will be invoked. It is worth noting that *switch statement* usually can be converted to an equivalent if-else statement [13]. Therefore, we do not design another F(X)-MAN connector for it.



```
if (cond < 0) { execute S1; }
else if (cond < 5) { execute S2; }
else { execute S3; }
```

Fig. 3. F(X)-MAN: Selector.

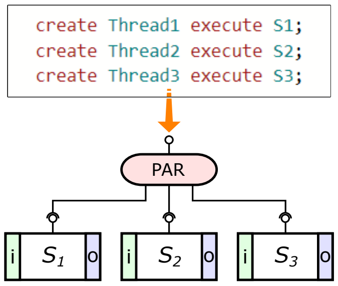
- **While statement.** The *while statement* presents the basic iteration [14]. In F(X)-MAN, we define an adaptation connector called *loop*, as illustrated in Figure 4. The loop connector repeatedly evaluates the expression, then invokes the service if the evaluation result is true, or stop the iteration if the evaluation result is false. Except for the while statement, *do-while statement* and *for statement* are also widely used, and both can be easily converted to equivalent while statements. Hence, we suggest to use the loop connector for all iterations.
- **Parallel statement.** The *parallel statement*, also known



```
while (cond < 10) { execute S1; }
```

Fig. 4. F(X)-MAN: Loop.

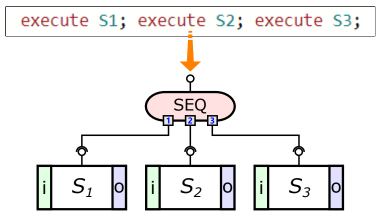
as *concurrent statement*, indicates a certain synchronization of concurrent activities [15]. Figure 5 shows a composition connector called *parallel* that denotes contemporaneous execution of three services. Notably, when parallel connectors are used in an F(X)-MAN service, we must pay more attention to its underlying issues, e.g., deadlock [16] and race condition [17].



```
create Thread1 execute S1;
create Thread2 execute S2;
create Thread3 execute S3;
```

Fig. 5. F(X)-MAN: Parallel connector.

- **Block statement.** The *block statement*, also known as *compound statement*, is generally adopted to group a sequence of multiple statements [18]. Therefore, F(X)-MAN model provides a composition connector, namely *sequencer*, which allows user to determine the execution order.



```
execute S1; execute S2; execute S3;
```

Fig. 6. F(X)-MAN: Sequencer.

While the exogenous connectors coordinate the control flows, the *data channels* coordinate the data flows. After a composite service is structured, we need to add data channels between services in order to define the direction of each data flow. Such a channel links the *input* and *output* of services. Figure 1(b) demonstrates two types of data channels in the composite service: *horizontal data routing* and *vertical data routing* [19]. The former is between two individual services, which indicates a service passes the outcome data to another, while the latter is data propagation between the services and its sub-services, which illustrates the data received by the composite service is passed to the first invoked sub-service, whereas the outcome data of last invoked sub-service becomes the output of the composite service. It is worth noting that we use JSON as the data interchange format, because (i) JSON has simple API that are available for many languages [20], (ii) the name-value pairs provides consistent patterns that are understandable by any user [21], and (iii) JSON provides faster object serialization and deserialization with less resources in comparison with other formats [22].

Next section presents how to use the F(X)-MAN composition semantic to construct FaaS.

IV. Example

In this section, we present an example of using F(X)-MAN service model for application development based on FaaS.

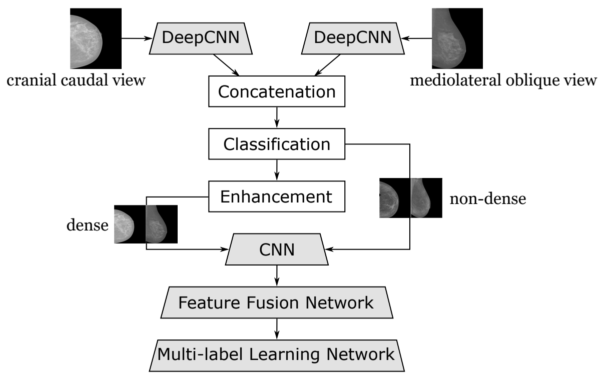
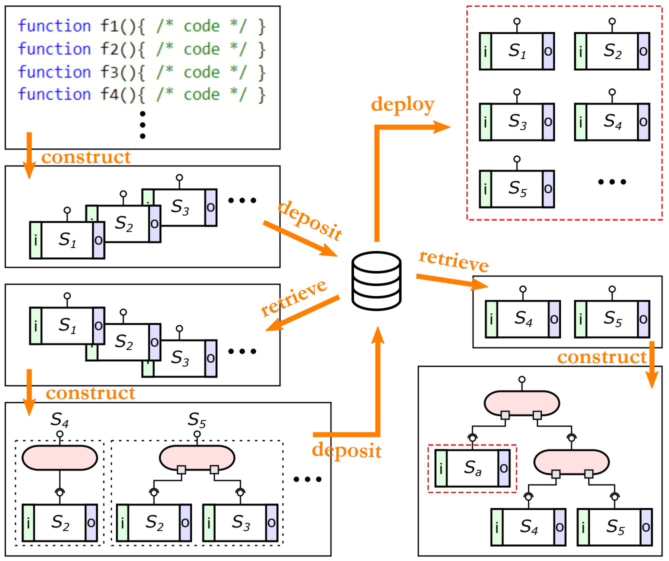


Fig. 7. A brief overview of mammography reading system.

Over the past decade, artificial intelligence (AI) technologies have been developed at an unprecedented speed, whilst AI applications have increasingly penetrated all areas of industry [23], e.g., intelligent healthcare [24] and E-commerce [25]. As a nature consequence, developing such a practical AI application commonly requires experts from various domains. Moreover, realizing the built-in AI algorithms such as training neural networks requires significant investment of time, effort, data and computing resource. Therefore, to the companies who want to focus on the core business whilst gain the benefits from data, FaaS looks like a promising solution, by means of a distribution model allowing AI services outsourcing offered by third-party vendors [26].

We hereby use F(X)-MAN service model to construct a *mammography reading system* (adapted from [27]) with the



```
function f1(){ /* code */ }
function f2(){ /* code */ }
function f3(){ /* code */ }
function f4(){ /* code */ }
⋮
```

Fig. 8. F(X)-MAN: Workflow.

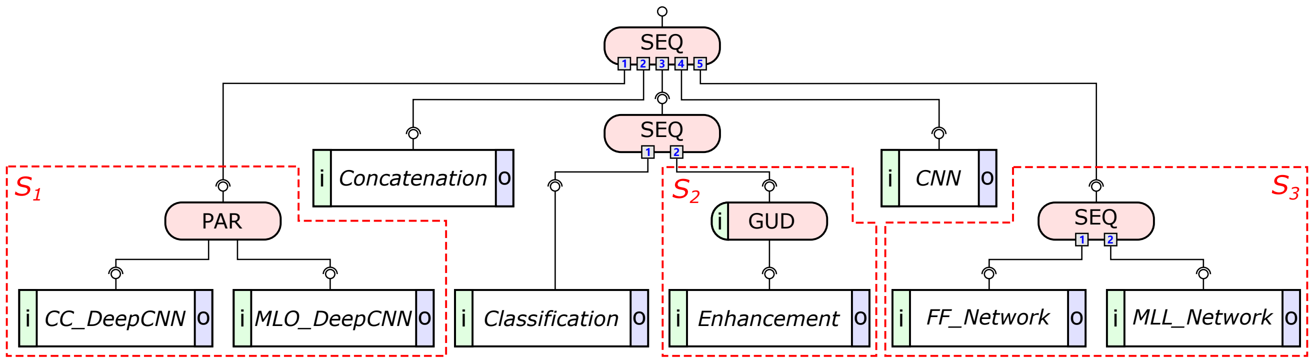


Fig. 9. F(X)-MAN: Mammography reading system.

ability to detect and label the lesions for breast cancer diagnosis. This system analyzes two-view mammography images through several well-trained deep learning models, each of which is a multi-layer neural network that processes medical data in a hierarchical fashion. Figure 9 demonstrates the primitive functional requirements.

At design-time, we firstly implement the relevant functions, and use them to create atomic services. Then we deposit the atomic services to a repository, and later retrieve them to construct composite services. The resulting services can be deposited again for further construction. After that, we can deploy the services (atomic or composite) from the repository to other developers and becomes a part of their programs at run-time, or use these services as building blocks to construct our own applications. The whole workflow of F(X)-MAN is illustrated in Figure 8.

Accordingly, we construct the mammography reading system, as shown in Figure 9. For the clarity, we omit the data channels. It is worth noting that S_1, S_2 and S_3 are services developed and deployed by third-parties, which are seamless connected to our system.

V. Evaluation

This section provides an evaluation of our algebraic and hierarchical composition model via several quality attributes, i.e., low coupling, testability, scalability, reusability, maintainability and evolvability.

A. Low Coupling

In software evaluation, *coupling* is a term used to measure the degree of connection and the amount of interaction between modules [28]. The higher the coupling, the more likely it is that changes to the inside of a module will effect the original behavior of another one [29]. Thereby, low coupling is one of the ultimate pursuits for software engineering.

There are six levels of coupling, as enumerated in increasing order of malignity [30]: *data coupling*, *stamp coupling*, *control coupling*, *common coupling* and *content coupling*. Our F(X)-MAN service model only generates the loosest two couplings in a system:

1) In data coupling, the communication between services is limited, i.e., via scalar parameters, in which only simple arguments are allowed to pass directly, e.g., variable and array. The passed data is always used for an invocation or a return of control [31].
2) Likewise, the communication in stamp coupling is also limited. But it passes composite data item, which usually is a entire data structure. Thus, sometimes a data structure may contain pieces of data that are unnecessary to the recipient module [32].

The coupling has a huge impact on testability, scalability, reusability, maintainability and evolvability.

B. Testability

Testability refers to the effort and time required to validate the software against its requirement. Thus, a system with better testability can be validated faster and easier [33]. However, perform testing in a serverless environment is never a simple task. An application using FaaS indicates that local code and foreign code are tangling together. It is difficult to run such

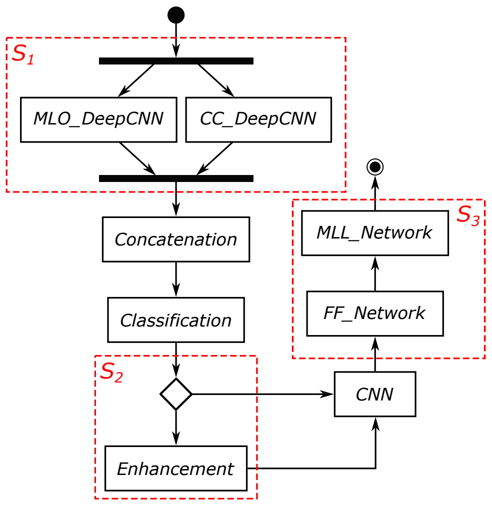


Fig. 10. F(X)-MAN: Deriving a statechart.

an application locally, unless the local environment can fully simulate the cloud environment [34].

Although the problem cannot be tackled once for all, we do facilitate the testability of systems build by F(X)-MAN service model. Firstly, such a system is completely modularized with low coupling, which means every behavior implemented in the local environment can be examined in isolation by means of unit testing. Secondly, the control flows are coordinated by the exogenous connectors, i.e., outside of the services, which implies we can verify the system behavior through a statechart directly derived by following the control flows, without take the services (local and remote) into account. Figure 10 is the statechart derived from the mammography reading system in Figure 9.

C. Scalability

Scalability is a term that frequently appears in a variety of computer science domains. Hence, we must explicitly understand the scalability needed to be evaluated in the scope of FaaS. In order to avoid the ambiguity, we hereby define the scalability from two different aspects.

From the perspective of software engineering, scalability is a fundamental quality referring to the impact of code expansion [35]. In other words, the scalability of F(X)-MAN denotes the effectiveness of F(X)-MAN when used on differently sized problems. As presented in Section III and IV, F(X)-MAN provides outstanding mechanisms for partitioning, composition and visibility control, which result in great scalability [36]. For example, the mammography reading system constructed in Figure 9 can be regarded as another F(X)-MAN service, which can be composed with other services, such as a *mammography report generator*, to create a *breast cancer auxiliary diagnosis application*, which can be further composed again for a very large software.

On the other hand, scalability in the context of cloud computing describes the capability of a system to increase its throughput under an increased load, e.g., creating more service instances [37]. As a matter of fact, comparing to the traditional monolithic models, FaaS achieves much better scalability. Figure 11 makes a comparison. As Figure 11(a) shows, a monolithic model encapsulates all its functionalities into a single process, and scales by replicating the entire monolith. Contrariwise, current FaaS models put implemented functionalities into separate services, and replicate the desired services for scaling, as expressed in Figure 11(b). Apparently, in FaaS, only services with higher demand will be scaled, while the monolithic models anyhow waste resources.

Our F(X)-MAN model make a further improvement on scalability. Except the advantages brought by general FaaS models, F(X)-MAN also has a superb tailorability, which is another way of assessing scalability [36]. For example, we can directly instantiate the sub-services of a F(X)-MAN composite service, and use them for new compositions, as illustrated in Figure 11(c).

D. Reusability

Reusability is the capability of a previously implemented service to be used again or used repeatedly in part or in its entirely, with or without modification [38]. As aforementioned, due to the loose couplings among F(X)-MAN services, for a composite service, the reuse can happen at every level of granularity, from the atomic services to itself. Hence, the reusability of a concrete application also depends on the software design, e.g., how many methods should be put in a service. The details has been discussed earlier and demonstrated in Figure 8.

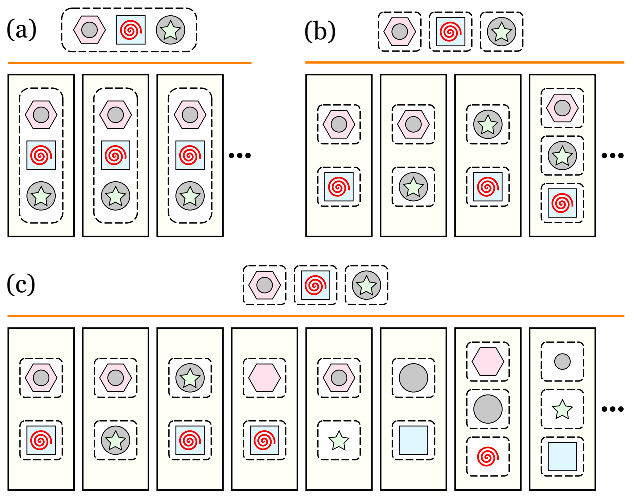


Fig. 11. Comparison of scalability.

E. Maintainability

Maintainability is a composition of three main subcharacteristics: *analyzability*, *changeability*, and *understandability* [39]. So, we perform the evaluation of maintainability based on these three quality attributes, which are interpreted as follows:

- **Analyzability.** It describes the capability of model or source code of a software to be diagnosed for deficiency. In F(X)-MAN, a composite service can be analyzed from two perspectives: the control flow can be identified by its exogenous connectors, while the data flow can be observed by data channels.
- **Changeability.** It refers to the possibility and ease of modification in an application. Because of the low coupling, we can change any service in an application without effecting others, all we need to confirm is the related data channels.
- **Understandability.** It indicates how easy to understand an application by its developers and users. It becomes obvious that an application constructed by F(X)-MAN is a tree, whose structure visualizes the hierarchical composition of services. Moreover, every connector visualizes a fixed semantic that can form a statement.

F. Evolvability

Evolution of software is inevitable in industry, due to the changing requirements must be satisfied during the life cycle. Thus, the cost of software mainly depends on the evolution in long term. Thus, evolvability is the capability of an application to enable its own evolution. In comparison with changeability, evolvability refers to the change caused by new requirements. As we presented before, the computational nature of F(X)-MAN allows us replace any part of an F(X)-MAN architecture with a new service in a simple fashion, while maintaining its architectural integrity.

VI. Conclusions and Future Work

This paper presents an algebraic and hierarchical composition model for FaaS. In the future, we plan to make an empirical study for the evaluation of F(X)-MAN model and implement a development tool.

Acknowledgments

This work has been supported by the Initial Research Funds for Young Teachers of Donghua University, the National Key R&D Program of China under Grant 2019YFE0190500 and Shanghai Engineering Research Center on Big Data Management System.

References

[1] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, and A. Ghalsasi, "Cloud computing—the business perspective," *Decision support systems*, vol. 51, no. 1, pp. 176–189, 2011.

[2] M. Stigler, "Understanding serverless computing," in *Beginning Serverless Computing*. Springer, 2018, pp. 1–14.

[3] L. Carvalho and A. P. F. de Araújo, "Framework node2faas: Automatic nodejs application converter for function as a service," in *Proceedings of the 9th International Conference on Cloud Computing and Services Science*, vol. 1, 2019, pp. 271–278.

[4] B. P. Rimal, E. Choi, and I. Lumb, "A taxonomy and survey of cloud computing systems," in *2009 Fifth International Joint Conference on INC, IMS and IDC*. Ieee, 2009, pp. 44–51.

[5] K.-K. Lau and S. di Cola, *An Introduction to Component-based Software Development*. World Scientific, 2017.

[6] S. di Cola, C. Tran, K.-K. Lau, C. Qian, and M. Schulze, "A component model for defining software product families with explicit variation points," in *Proceedings of 19th International ACM SIGSOFT Symposium on Component-Based Software Engineering*. ACM, April 2016, pp. 79–84.

[7] C. Qian and K.-K. Lau, "Enumerative variability in software product families," in *Computational Science and Computational Intelligence (CSCI), 2017 International Conference on*. IEEE, 2017, pp. 957–962.

[8] R. Arshad and K.-K. Lau, "Reverse engineering encapsulated components from object-oriented legacy code," in *Proceedings of The 30th International Conference on Software Engineering and Knowledge Engineering*. KSI Research Inc., July 2018, pp. 572–577.

[9] D. Arellanes and K.-K. Lau, "Exogenous connectors for hierarchical service composition," in *Proceedings of 2017 IEEE 10th International Conference on Service-Oriented Computing and Applications*. IEEE, 2017, pp. 125–132.

[10] D. Arellanes and K.-K. Lau, "Algebraic service composition for user-centric IoT applications," in *International Conference on Internet of Things*. Springer, 2018, pp. 56–69.

[11] I. Griffiths, *Programming C# 8.0: Build Cloud, Web, and Desktop Applications*. O'Reilly Media, 2019.

[12] I. Kalb, "If, else, and elif statements," in *Learn to Program with Python 3*. Springer, 2018, pp. 103–141.

[13] M. Ogihara, "The switch statements," in *Fundamentals of Java Programming*. Springer, 2018, pp. 245–261.

[14] S. Kedar, *Principles Of Programming Languages*. Technical Publications, 2008.

[15] C. Snow, *Concurrent Programming*, ser. Cambridge Computer Science Texts. Cambridge University Press, 1992.

[16] W. Haque, "Concurrent deadlock detection in parallel programs," *International Journal of Computers and Applications*, vol. 28, no. 1, pp. 19–25, 2006.

[17] R. H. Netzer and B. P. Miller, "What are race conditions? Some issues and formalizations," *ACM Letters on Programming Languages and Systems (LOPLAS)*, vol. 1, no. 1, pp. 74–88, 1992.

[18] E. Organick, A. Forsythe, and R. Plummer, *Programming Language Structures*. Elsevier Science, 2014.

[19] K.-K. Lau and C. Tran, "X-MAN: An MDE tool for component-based system development," in *Proc. 38th EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE, 2012, pp. 158–165.

[20] B. Smith, *Beginning JSON*. Apress, 2015.

[21] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski *et al.*, "Serverless computing: Current trends and open problems," in *Research Advances in Cloud Computing*. Springer, 2017, pp. 1–20.

[22] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta, "Comparison of json and xml data interchange formats: a case study." *Caine*, vol. 9, pp. 157–162, 2009.

[23] S. J. Russell and P. Norvig, *Artificial intelligence: A modern approach*. Malaysia; Pearson Education Limited,, 2016.

[24] P. Hamet and J. Tremblay, "Artificial intelligence in medicine," *Metabolism*, vol. 69, pp. S36–S40, 2017.

[25] F. J. Martínez-López and J. Casillas, "Artificial intelligence-based systems applied in industrial marketing: An historical overview, current and future insights," *Industrial Marketing Management*, vol. 42, no. 4, pp. 489–495, 2013.

[26] S. Parsaeefard, I. Tabrizian, and A. Leon-Garcia, "Artificial intelligence as a service (AI-aaS) on software-defined infrastructure," in *2019 IEEE Conference on Standards for Communications and Networking (CSCN)*. IEEE, 2019, pp. 1–7.

[27] X. Wang, J. Li, and C. Qian, "Semantic label prediction of mammography based on CC and MLO views," in *Proceedings of 18th IEEE International Conference on Machine Learning and Applications*. IEEE, 2019, forthcoming.

[28] D. King, *Current practices in software development: a guide to successful systems*. Prentice Hall, 1984.

[29] F. Coda, C. Ghezzi, G. Vigna, and F. Garzotto, "Towards a software engineering approach to web site development," in *Proceedings of the 9th international workshop on Software specification and design*. IEEE Computer Society, 1998, p. 8.

[30] A. J. Offutt, M. J. Harrold, and P. Kolte, "A software metric system for module coupling," *Journal of Systems and Software*, vol. 20, no. 3, pp. 295–308, 1993.

[31] M. Hitz and B. Montazeri, "Measuring coupling and cohesion in object-oriented systems," 1995.

[32] G. Feuerlicht, "Simple metric for assessing quality of service design," in *International Conference on Service-Oriented Computing*. Springer, 2010, pp. 133–143.

[33] M. Mattsson, H. Grahn, and F. Mårtensson, "Software architecture evaluation methods for performance, maintainability, testability, and portability," in *Second International Conference on the Quality of Software Architectures*. Citeseer, 2006.

[34] P. Leitner, E. Wittern, J. Spillner, and W. Hummer, "A mixed-method empirical study of function-as-a-service software development in industrial practice," *Journal of Systems and Software*, vol. 149, pp. 340–359, 2019.

[35] M. Anastaopoulos and C. Gacek, "Implementing product line variabilities," Fraunhofer Institut Experimentelles software Engineering, Kaiserslautern, Germany, Technical Report IESE-Report No. 089.00/E, Version 1.0, Nov. 2000.

[36] M. Laitinen, M. E. Fayad, and R. P. Ward, "Thinking objectively: The problem with scalability," *Communications of the ACM*, vol. 43, no. 9, pp. 105–107, 2000.

[37] D. F. Garcia, G. Rodrigo, J. Entrialgo, J. Garcia, and M. Garcia, "Experimental evaluation of horizontal and vertical scalability of cluster-based application servers for transactional workloads," in *8th International Conference on Applied Informatics and Communications (AIC'08)*, 2008, pp. 29–34.

[38] S. Thakral and S. Sagar, "Reusability in component based software development - a review," *World Applied Sciences Journal*, vol. 31, no. 12, pp. 2068–2072, 2014.

[39] E. Bagheri and D. Gasevic, "Assessing the maintainability of software product line feature models using structural metrics," *Software Quality Journal*, vol. 19, no. 3, pp. 579–612, 2011.