An Approach to Evolving Legacy Enterprise System to Microservice-Based Architecture through Feature-Driven Evolution Rules

Safa Habibullah, Xiaodong Liu, and Zhiyuan Tan

***Abstract*—Evolving legacy enterprise systems into a lean system architecture has been on the agendas of many enterprises. Recent advance in legacy system evaluation is in favour of microservice technologies, which not only significantly reduce the complexity in deployment of enterprise systems but also enhance the availability of services to system users. However, there are technical challenges to overcome towards a successful transformation. Challenges, relating to information security, container optimisation, the performance of a new system and its deployment, are particularly under concern. To cope with these technical challenges, a new approach is proposed in this paper to govern the evolution of legacy systems into a microservice-based and cloud-hosted architecture. This approach includes a detailed working process, a set of transformation rules towards microservices and their repositories. In addition, a conceptual framework is drawn to provide a comprehensive illustration to this evolution approach. Moreover, a case study is presented to demonstrate the application of the proposed approach on the evaluation of a chosen legacy system, as well as the procedure of system architecture optimisation.**

***Index Terms*—Cloud computing, enterprise legacy system, feature-driven evolution rules, microservices.**

I. Introduction

Software system evolution is subject to the sophistication of available hardware and the cost of implementation. Over the last decade, enterprise systems were designed and developed based on specific hardware at a fixed scale until virtualization and cloud computing technologies were introduced. These technologies provide enterprise system development with greater flexibility. A new wave of revolution has been aroused by the recent introduction of microservice architecture.

The continuous development of microservice mechanism, both in academia and the industry, shows different features, such as a fast start-up, separation units, etc. This new technique allows the monolithic application to be broken down into smaller cohesive independent services with respect to their functionality. Microservices are organised as a set of small services runing in its own process and communicating with other services via an Application Programming Interface (API) gateway, that allows developers to update each service independently without affecting others.

Owing to the aforementioned technical advantages, microservices provide a new solution to evolving legacy systems nowadays. Deploying a microservice in the cloud provides a new means to modernise software applications and is adopted by enterprises for next-stage system deployment. Microservice can be developed in any programming language that means it gives a development team freedom to choose and implement the most appropriate programming languages that suits better to their characteristics and requirements. In addition, each service represents a small job, which makes it easy for the developer to understand.

Microservices, however, are not without issues and challenges. Thus, to build the most suitable foundation for enterprise, a solid understanding on this new paradigm is crucial as it is changing how enterprises handle information, and resolve legacy system weaknesses such as scaling challenges.

The rest of the paper is structured as follows. Section II presents the related work. Section III discusses the conceptual framework and the set of transformation rules. A case study is presented in Section IV. Conclusions are drawn and future work is introduced in Section V.

II. Related Work

Microservices are inspired by service-oriented computing and have evolved from business solutions for modern applications. Microservices have gained popularity in recent years, and a clear understanding of this paradigm is critical to sensible implementation of microservice-based systems. Thus, an in-depth investigation was conducted to identify state-of-the-art microservice architectures. It reveals that recent studies have focussed on the scalability, availability and performance of these architectures.

In this section, we aim to divide the related work into three parts to provide a fair evaluation of microservice architectures.

A. Systematic Study

Francesco *et al*. conducted a systematic mapping study in [1] a clear summary of microservices is presented and serves a solid foundational reference for both academia and industry professionals. This study also identifies a gap in the system quality attributes, such as security, portability and testability. Another systematic mapping study shows that some of the qualities of attributes have not be thoroughly investigated, such as security features for microservice architecture [2]. This study also provides a broad overview of the challenges facing microservice architecture and related technologies,

Manuscript received September 13, 2018; revised October 12, 2018.

Safa Habibullah, Xiaodong Liu, and Zhiyuan Tan are with School of Computing, Edinburgh Napier University, Edinburgh, UK (email: s.habibullah@napier.ac.uk, x.liu@napier.ac.uk, z.tan@napier.ac.uk).

DOI: 10.7763/IJCTE.2018.V10.1219

which can be used by the research community to widen this field of study.

B. Pattern

To provide a guide for the process of system migration, Balalaie *et al*. [3] followed the principles of situational method engineering to design a set of preliminary repositories for microservice migration patterns. Different solutions were provided to decompose a legacy system, and the challenges of the proposed patterns were discussed, while different patterns were implemented using a variety of techniques depending on service discovery, the load balancer, and the circuit breaker. Each transform pattern has been provided by a scenario that gives a clear idea for the reasons for the evolution transform. The authors in [3] stabilised a system by applying a pattern that takes into account containerisation of the service. On the other hand, all the above techniques have limitations that must be considered. If it is not properly managed and implemented, it might become a single point of failure. Furthermore, migration patterns include certain weaknesses. For example, the proposed pattern was simple and focused on the migration planning phase.

C. Case Study

Dragoni *et al*. [4], for example, presented a real case study of a mission critical system - the Foreign eXchange (FX) core system at Danske Bank - which includes a legacy system architecture, to determine the primary issues that affect system scalability. Microservice architectures have also shown to be a promising approach to reduce the complexity of the code. Moreover, microservices can be decoupled by using service discovery. As mentioned in [4] switching to a microservices architecture leads to better scalability by applying various techniques, such as horizontal scaling, clustering, cash and load balancing techniques [4]. However, The FX system can be enhanced by implementing the above techniques to support other quality attributes. For example, including security to deliver innovative user-experience to customer.

Villamizarn *et al.* [5] promoted an enterprise case study that was developed and deployed in the cloud podium in both the monolithic and microservice architecture. However, the authors analysed the performance of both architectures in terms of response time, more features of performance need to be measured in the future.

Based on existing proposals and approaches, a research gap exists in relation to non-functional attributes of microservice architecture, such as performance and security. Thus, the current study seeks to create a transformation plan for a new architecture taking into account the situational context of the legacy application. To improve software evolution, a feature-driven, agile methodology is introduced as an effective means for developing the system. The importance of this method depends on building a detailed and prioritised features list to gain better understanding of the system and to provide guidance throughout system evolution. When the software system grows, the process of development becomes complex; thus, to overcome difficulties, feature-driven development is employed to create a set of rules. For security, performance and functionality are the primary requirements of developing and evolving a legacy system.

III. Main Approach

This research is constructed around legacy systems, cloud computing and microservices. It aims to develop a highly effective scheme, driven by evolution frameworks and rules, in order to modernise legacy systems. A novel approach is designed to evolve legacy enterprise systems into a lean system architecture with the support of microservices and cloud computing. The proposed framework, shown in Fig. 1, consists of three key components, namely core system, middle layer, and target system. It focuses on how to evolve a legacy system by using a microservice architecture and then migrating to a cloud platform.

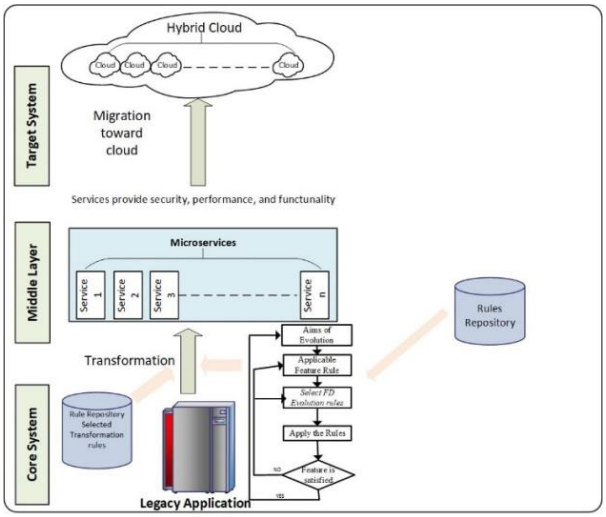


Fig. 1. Conceptual framework.

A. Core System

Firstly, we will start from a legacy application (i.e. legacy code). This process aims to restructure the elements from one representation to another at the same level of abstraction. The core system carries out the transformations from a legacy system to microservices where the source code of the legacy system candidate elements are identified for transformation. The primary objective of the core system is to define a set of feature driven transformation rules in order to establish a rational transformation of the legacy application into microservices. In accordance with business and technical concerns and requirements, different combinations of feature driven rules could be implemented.

This microservice-based approach can facilitate the development of new business functions rather than creating new elements as a part of the monolithic code, which may increase the complexity of the code. The microservice architecture allows services of a software system to evolve independently of other services. Services can be built using any programming language, and each of them runs as a distinct process. In addition, this microservice-based approach may reduce the size of the legacy application and create, independent and easily-maintained services [6].

B. Middle Layer

The main purpose of this layer is to move away from a legacy application and towards the microservice architecture.

The potential advantages of this architecture will lead to a better performance, the possibility of fewer system errors and enhanced functionality, security and scalability. The adoption of a microservice provides the best architecture that forms optimal solutions for enterprises. The conceptual framework of this research has given rise to the research challenge of how to transform legacy source code to a microservice, in order to achieve the intended target.

Initially, modernising the legacy application will be achieved by applying a set of transformation rules to the legacy code that takes its purpose into consideration. The transformation into the targeted architectural paradigm is accomplished by using the microservice technique. The microservice technique involves the substitution of each part of the legacy application functionality with a microservice, thus breaking the entire legacy system into services. Where to start and how to identify the candidate microservices are important points to be considered carefully, by:

1) Identifying the enterprise's capabilities and system domains. This will lead to the determination of the microservice that the enterprise needs to construct.
2) Identifying the information to be exchanged in the transaction. For instance, in e-commerce the orders, items and customers might need to be considered.
3) Analysing the code allows one to break up some features much easier than others. Microservices ought to maintain their respective database rather than to share a common one. The reason for keeping separate databases is to avoid updating every service when there is a change in a common database and to minimize coupling between the services. Besides, when splitting the monolith into services, one must also consider how these services will be deployed into the cloud platform. In order to take advantage of this technological phenomenon, we will consider containerization technologies, apart from the microservice. Containers provide portability for microservices in various platforms. Developers will be able to move the microservices seamlessly between the private cloud and the public cloud, as is shown in Fig. 2.

Fig. 2 illustrates that a legacy application is re-architectured into a set of containerised services. Each container is independent and deployed in the cloud podium. The container removes redundant resources that virtual instances need, which improves the performance.

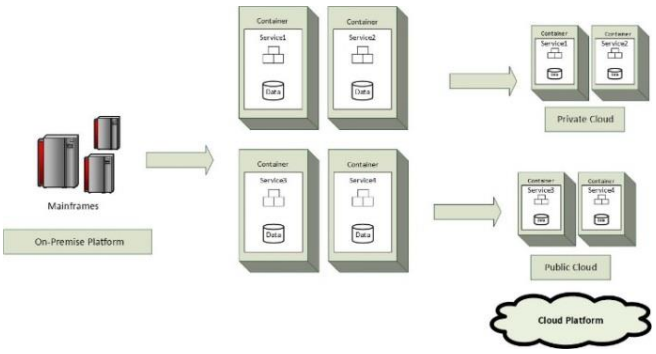


Fig. 2. The legacy application is re-architectured into a set of containerised services (microservices).

C. Target System

This layer represents the design of a target architecture and the cloud components used in migration. In order to deliver containerised microservices, a hybrid cloud platform will be considered, this is the most important step to achieve the enterprise's needs, such as better performance, security and scalability. Furthermore, for the migration plan, different requirements will be introduced as well as different satisfactory solutions will be proposed.

Before moving to a hybrid cloud, we need to understand what it is. Mell and Grance [7] defined a hybrid cloud as 'a composition of two or more distinct cloud infrastructures (i.e. private, community or public) that remain unique entities, but are bound together by standardised or proprietary technology which enables data and application portability (e.g. cloud bursting for load balancing between clouds)'. According to Mazhelis and Tyrvainen [8], the hybrid cloud's purpose is to provide the efficient distribution of the load between the clouds

Private and public clouds will be used in this research in order to provide the deployed environment for our new architecture. It is presumed that part of the enterprise software will be deployed in a public cloud specifically, whereas other types of subsystems will be deployed in a private cloud due to confidential information. Thus, there will be two types of deployment environments: containerised microservices hosted by the public cloud and containerised microservices hosted by the private cloud.

The system could be moved from the private to a public cloud and vice versa, depending on the enterprise's requirements. In general, this layer has two parts: the functional part and the technical part. From the functional perspective, the target system not only represents the functionalities that will be delivered, but also covers non-functional aspects, such as performance and security. From the technical viewpoint, a decision will be made among several possible selections of technologies that will be used.

In order to further advance the state of the art of microservice-based legacy system migration, it has been decided to employ the idea of the feature driven, based in legacy system evolution and we set out to propose a feature-driven microservice transformation rule repository. These rules define transformation that can be applied to software architecture in order to maintain the system goals, requirements and objectives. The migration rules will help to deal with software architecture during implementation and operation.

D. Feature Driven Microservice Rules

A set of features driven microservice rules is proposed in this section. The rules are documented in the pseudo code format. The title of each rule is given based on its context. Each rule consists of an assumption, condition and impact on feature.

An assumption part reflects a main problem of a system. A condition is a mirror of the solution. Lastly, an impact on a feature explains the concerns that might arise when the rule is applied.

TABLE I: RULE1- DECOMPOSING LEGACY APPLICATION TO MICROSERVICE ARCHITECTURE

Assumption	There is a monolithic system which needs to scale and improve modularity, and individual parts of a modular application may be

	independently deployed. Ls = legacy system;

Condition	Decomposing the legacy system to a microservice architecture through building an application from internal light weight services will improve modularity and simplify scaling a particular service to meet new demands and requirements.
	Decomposition can happen as a result of different non- functional requirements and the size of the service depends on the complexity of the problem domain.
	Ls = legacy system;
	Mo = modularity;
	Sc = scalability;
	Ms = microservice architecture;
	While (Ls)
	{
	Create Ms;
	Mo_max ∧ Sc_max;
	}

Impact on features	▪ separate processes add complexity and new problems, including network latency ▪ the management of dependencies and deployment is more complex

TABLE II: RULE 2- SINGLE RESPONSIBILITY PRINCIPLE

Assumption	A monolithic application has suffered from tight coupling and dependencies between modules. Ls _D= legacy system dependency;

Condition	Decomposing a system into small services minimise dependencies and become loose coupling by applying the Single Responsibility Principle (SRP) concept. In addition, separating the dependent service can improve scalability of each service [9].
	This allows the developer to change the implementation or modify the service and replace them without any downstream impact.
	Ms_SRP = microservice architecture based on the single responsibility principle;
	Ls _ D = legacy system dependency;
	D = dependency;
	Sc = scalability;
	While (Ls_D)
	{
	Create Ms _SRP;
	D_min ∧ Sc_ max;
	}

Impact on features	The core complexity of this approach is increasing memory consumption.

TABLE III: RULE 3- SEPARATE DATABA

Assumption	A monolithic application has been decomposed into a set of services to ensure that the services are loosely coupled (i.e. they can be developed, deployed and scaled independently from other services). Ls = legacy system; Ms = Microservice architecture;

Condition	Keeping separate data store for each service.
	Ms _ DB = microservice architecture with separate data store;
	Ls_TC = legacy system _tightly coupled;
	D = dependency;
	While (Ls_TC)
	{
	Creat Ms_DB;
	D_min;
	}

Impact on features	Breaking data can make data management more complicated.
	Implementing queries that join data is becoming a new challenge.

TABLE IV: RULE 4- CIRCUIT BREAKER

Assumption	If one or more services is not available or might suffer from high latency that will result in a cascading failure.
Condition	The circuit breaker is the best solution to prevent this failure across multiple services in order to improve the stability and resiliency of an application [3].
Impact on features	How to handle the exceptions if the services are not available. For example, a remote service might be crashed.

TABLE V: RULE 5- COMPUTATION TASK

Assumption	Decomposing the legacy application to microservice architecture has a serious effect on the response time. Ls= legacy system; P= performance; LS = P_max.

Condition	The response time of a request is an important indicator for the user perceived performance of the system. The response time should decrease compared to the monolithic application, in order to improve the computation task of the service.
	Response time ↔ Computation Task (trade off)
	Ms_t = microservice response time;
	Ms_ct = microservice computation task
	P_m = performance max;
	Ls_t = legacy system response time;
	
	While (Ms_t > Ls_t)
	{
	Increased Ms_ct;
	Increased P_m;
	}

Impact on features	Improved computation task and minimised response time by distributing the workload between services.
	ct = computation task;
	t= response time;
	∀ ct, t;
	ct_max ∧ t_min;
	Services are independently deployed and developed.

TABLE VI: RULE 6- I/O PROCESSING

Assumption	In order to enhance the legacy system performance precisely response time, the system will be decomposed to a set of services. P= performance; Ms = microservice architecture; Ms= p_max.

Action	The response time of a request is an important indicator for the user-perceived better performance of the system.
	Response Time ↔ I/O Processing (Trade off)
	Ms_t = microservice response time;
	P_m = performace max;
	Ls_pf= legacy system I/O processing function;
	While (Ms_t > Ls_pf)
	{
	Increased P_m;
	}

Impact on features	Improved performance of the system by reducing the number of unnecessary I/O processing

	should be reducing. P = performance; Pf= i/o processing function; Ms= microservice architecture; ∀ p, pf; Ms = P_max ∧ pf_min

TABLE VII: RULE 7- THROUGHPUT

Assumption	In order to improve the legacy system network throughput, the system will be constructed into a set of services. Th = network throughput; Ms = microservice architecture; Ms = Th_max.

Condition	It is important to rise the number of the throughput (i.e. messages processed by) of microservices, by comparing with the monolithic application and reducing the number of the memory intensive function.
	Throughput ↔ Memory Intensive (trade off)
	Th= network throughput;
	Ms= microservice architecture;
	Mi = memory intensive function;
	P = PERFORMANCE;
	While (Ms_Th > Ms_mi)
	{
	Increased P_Ms;
	}

Impact on features	Improved throughput by making sure the system has sufficient use of the memory spaces. Ms= microservice architecture; Mi = memory intensive function; Th= network throughput; ∀ Th, Mi; Ms_P = Th_max ∧ Mi_min

IV. CASE STUDY

The framework presented in Fig. 3 shows an existing legacy system architecture. Given that, the legacy system includes a user interface, business logic and a file system. This architecture has all the application modules, and files in one package. To apply microservice architecture, the system should be reconstructed and separated into module based on the above rules.

In this case study, we used a simple open source legacy code (i.e. ferry booking system) from GitHub [10]. The system had been designed as a console application and the system provided booking and management services. As well as providing information about ferry crossing, ferry module and types and different routes for each type of ferry. The system also presented an information about each port which includes, port origin, port destination, departure time and journey time. To construct a microservice architecture, the system was analysed by understanding the whole system architecture. Redesigning the legacy system was a difficult process; the legacy code was inefficient and there was a shortage of the system documentation.

The process that used to decompose the system into microservice architecture are:

1) Understanding the code and defining system boundaries.
2) Building a class diagram for the existing system helps to understand the relationships between the system class and plays an essential role in determining the service.
3) Recognising and determining which rule is going to apply. In this case, three rules have been considered to implement, which are decomposing legacy to microservice, SRP and separate database rules.

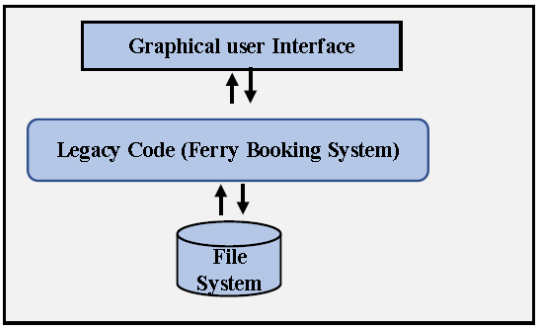


Fig. 3. Existing legacy system architecture.

The first step was to reconstruct the existing system to determine the service elements from the legacy code. Building up the microservice was achieved by applying the first and second rules: decomposing the legacy system into microservice as mentioned on Table II and using the Single Responsibility Principles (SRP) rules as presented on Table II. Implementing a set of persistent and cohesive services will be achieved by using SRP rule.

As a result, three services were selected from the legacy java code; Table VIII indiscates the three services and the functional task that relates to each service.

TABLE VIII: SERVICES CANDIDATE

Service	Functionality
Booking	This service allows the user to make a booking and can easily cancel or amend the booking.
Ferry	This service provides different types of ferry and different routes the associate with each one
Port	A port origin, port destination, departure time and journey time will be determined by this service.

In addition, the ferry system uses a file system to manage and manipulate the data; this leads to many problems, like data integrity, and this type of issue can be avoided by using an isolated database based on the separate database rule as described on Table III. This rule will be applied to the already identified candidate services to ensure that each data service is independent of the other as shown in Fig. 4.

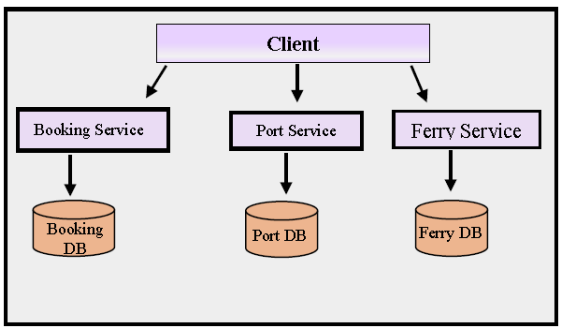


Fig. 4. Microservice architecture.

The approach of the system evolution discussed in this paper led to reduced coupling, improved modularity, scalability and improved response time by reducing the unnecessary I/O processing.

V. Conclusion and Future Work

The research aims to develop an approach to evolve legacy systems, which will incorporate the microservices technique, through developing framework architecture to determine how effective the new service is in terms of security, performance and functionality. This framework focuses on the microservice rule repository and how these rules support these three features. At a later stage, more rules will be added to satisfy the system's needs.

The future work will include applying the above and enhanced evolution rules to a medium to large enterprise system to evaluate the rules and make sure that the approach is scalable for the transformation to industrial scaled microservice-based architecture and meanwhile maintains acceptable performance, functional and security.

References

[1] P. D. Francesco, I. Malavolta, and P. Lago, "Research on architecting microservices: Trends, focus, and potential for industrial adoption," in *Proc. 2017 IEEE International Conference on Software Architecture (ICSA)*, 2017.

[2] N. Alshuqayran, N. Ali, and R. Evans, "A systematic mapping study in microservice architecture," in *Proc. 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, pp. 44-51, 2016.

[3] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices migration patterns," Technical Report no. 1 TR-SUTCE-ASE-2015-01. Automated Software Engineering Group, Sharif University of Technology, Tehran, Iran, 2015.

[4] N. Dragoni, S. Dustdar, S. T. Larsen, and M. Mazzara. (2017). Microservices: Migration of a mission critical system. [Online]. Available: https://arxiv.org/abs/1704.04173

[5] M. Villamizar, O. Garcés, H. Castro, M. Verano, L. Salamanca, R. Casallas, and S. Gil, "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud," in *Proc. Computing Colombian Conference* , pp. 583-590, 2015.

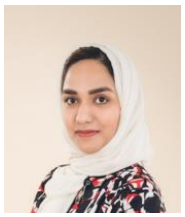
[6] A. Levcovitz, R. Terra, and M. T. Valente, "Towards a technique for extracting microservices from monolithic enterprise systems," in *Proc. 3rd Brazilian Workshop on Software Visualization, Evolution and Maintenance (VEM)*, pp. 97-104, 2015.

[7] P. Mell and T. Grance, "The NIST definition of cloud computing," NIST Special Publication 800-145, U.S. Department of Commerce, 2011.

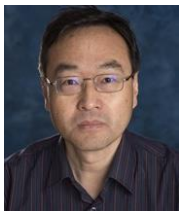
[8] O. Mazhelis and P. Tyrväinen, "Economic aspects of hybrid cloud infrastructure: User organization perspective," *Information Systems Frontiers*, vol. 14, no. 4, pp. 845-869, 2012.

[9] C. Richardson. Microservice architecture. [Online]. Available: http://microservices.io/patterns/microservices.html

[10] Github. [Online]. Available: https://github.com/Kirschstein/legacy-ferry-booking-system



Safa Habibullah is a PhD candidate in School of Computing in Edinburgh Napier university, UK. She holds a master degree in software technology for the web from Edinburgh Napier University, UK. She is a lecturer at the School of Computing, King Abdulaziz University, Saudi Arabia.



Xiaodong Liu received his PhD in computer science from De Montfort University and joined Napier in 1999. He is currently leading the software systems research group in the SoC, Edinburgh Napier University. He is an active researcher in software engineering with internationally excellent reputation and leading expertise, focusing on its emerging themes including pervasive systems (internet of things), services-oriented architecture, evolution of cloud services, and intelligence-driven software engineering. Prof. Liu has led 10 externally funded projects as the PI, and published over 100 papers in established international journals and conferences, 5 book chapters and 3 research handbooks. He is the inventor of 1 patent registered in UK and USA and the founder of a spin-out company. He has been the chair, co-chair or PC member of a number of IEEE and IASTED international conferences. He is the editorial board member of 4 international journals and editor of 3 research books and 2 journals special issues. He is a member of IEEE Computer Society.

Zhiyuan Tan holds a Ph.D. degree in computer systems from the University of Technology, Sydney, Australia. He is a lecturer at the School of Computing, Edinburgh Napier University (ENU), the United Kingdom. Prior to joining ENU, Dr. Tan held a postdoctoral research fellowship at the University of Twente, the Netherlands and the University of Technology Sydney, Australia respectively. His research has been supported by various funding agencies, including the Commonwealth Scientific and Industrial Research Organisation, Australia. His recent research findings have been published in leading journals, including IEEE Transactions on Parallel and Distributed Systems (TPDS), IEEE Transactions on Computer (TC), IEEE Transactions on Cloud Computing (TCC), Future Generation Computer Systems (FGCS), Computer Networks (CN) etc. Due to his research achievements in cybersecurity, Dr. Tan has been granted different research awards including the National Research Award 2017 from the Research Council of the Sultanate of Oman. Dr. Tan has been invited to serve as a reviewer for top-rated international journals, including IEEE TPDS, IEEE TC, IEEE Transactions on Information Forensics & Security (T-IFS), IEEE Transactions on Dependable and Secure Computing (TDSC), FGCS, Journal of Network and Computer Applications (JNCA), etc. He also has engaged in various research venues as a technical committee member, an organising chair, and a guest editor respectively.