2020 IEEE 15th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET) 978-1-7281-5566-1/20/$31.00 ©2020 IEEE 10.1109/TCSET49122.2020.235378

Highly-Distributed Systems Based on Micro-Services and their Construction Paradigms

Andriy Luntovskyy
BA Dresden University of Coop. Education
Saxon Study Academy
Dresden, Germany
Andriy.Luntovskyy@ba-dresden.de

Bohdan Shubyn
Institute of Telecommunications and Radioelectronics,
Lviv Polytechnic National University
Lviv, Ukraine
boshubin@gmail.com

***Abstract*—A definition for the HDS, as well as the demarcation to conventional distributed systems, were given. Typical architectures for HDS were discussed which affect increasing of QoS and of so-called QoE (Quality of Experience). The distinguishing features for HDS are clearly formulated. The advanced SWT (Software Technologies) approaches lead to use of young flexible service-oriented architectures like Micro-Services, which provide higher performance and small latencies, as well as better scalability, energy-efficiency and autarky.**

One possible option in the frame of HDS regarding security, privacy, authentication and compulsoriness of workflow steps, modules and service execution for such apps Blockchain and Smart Contracting are. The theoretical issues are proven via the represented examples and case studies.

***Key Words* — Highly-Distributed Systems, Agile Process Models, Quality of Experience, Service-Oriented Architectures, Micro-Services, DevOps, Scrum, Conway's Law, Blockchain.**

I. MOTIVATION

The main aim of the work is creation of so-called HDS (Highly-Distributed Systems), which are energy-efficient and cryptographically secured (SAML – Security Assertion Mark-up Language, firewalls, IDS/ IPS – Intrusion Detection /Prevention Systems), provide up-to-date QoS parameters (higher DR and availability, small latency) and support extended QoE (Quality of Experience) [6,7].The HDS have to possess flexible structures based on SOA and Micro-Services, as well as deploy efficient communication models (P2P, cloud-fog), which are able to solve the distribution conflicts in short time and support rapid access to the data analytics. Such HDS are often developed under use of advanced SWT (Software Technology) process models like DevOps and Scrum and are driven via Blockchain-conform cryptographic structures [7,8,11,12], which provide compulsoriness of required workflow steps and predictable execution of the deployed modules, services, Micro-Services and of other components within the internal architecture of the above-mentioned HDS.

A. Distributed Systems

The term "Distributed Systems" has been used for many years for applications, which operate in modern combined wired-wireless-mobile networks under clear co-operation goals, as well as have no centralization in memory access or synchronization in the clocking. The distributed applications are constructed on the sample n-tier and often possess redundancy in form of server and database replications. They follow to established SOA (service-oriented architecture) concept and can be often organized as cloud-centric structures. Significant architectural transformations in network services and distributed systems characterize an ongoing trend nowadays [1,2,7-10]. The clouds, clusters with explicit cooperation goal (e.g. parallelized computing) as well as grids belong to the above-mentioned systems.

B. Highly-Distributed Systems

Since 2005 the P2P systems (Internet of Things, fog) in combination with convenient C-S communication model as well as server-less structures (SLMA, robotics) have gained on popularity. After that, Cloud-based solutions became a trend (2011) under predominant use of the load-balanced "thin clients" with functionality delegation to the clouds [7,8,11,12]. Under use of fog computing the IoT solutions are constructed. The workload is shifted on the edge to the energy autarky and resource economizing small nodes. Finally, what does "Highly-Distributed Systems" mean?

The term "Highly-Distributed Systems" (HDS) must be deployed for the new mobile, frequently "quasi-offline" or server-less apps (SLMA), which extend the convenient distributed systems. They understand the use of efficient and performing networks under clear co-operation goals, as well as no centralization in memory access or synchronization in the clocking. Additionally, they possess more redundancy and possibility for replications due to use of flexible P2P structures, use of cloud and fog services. Energy autarky plays a very important role for HDS. Highly-Distributed Systems have more strata and layers in their architecture (better modularity and management with efficient conflict resolving) and are also more secured, especially for privacy and anonymity. For the development of such systems, the agile SWT methods and process models must be used [1,2].

The distinguishing features of HDS are as follows (Fig. 1):Advanced communication models (C-S with Clouds, Fog, P2P, M2M); Advanced methods for performance management and optimization as well as for QoE (Quality of Experience) increasing; Advanced SWT (agile approaches like XP, DevOps, Kanban, Scrum and so-called Micro-Services); Advanced Data Analytics regarding to solving of "Big Data" shortcomings. Therefore, the given paper possesses the

978-1-7281-5566-1/20/$31.00 ©2020 IEEE

following structure: 1) Section I contains the motivation and discusses the demarcation of convenient distribution systems to the so-called HDS; 2) Section II discusses advanced communication models for HDS; 3) Section III provides the up-to-date methods for performance management and optimization as well as defines clearly the important term of QoE (Quality of Experience); 4) Section IV contains an overview of the influence of the actual SWT processes model like DevOps and Scrum to the use of Micro-Services in HDS. Big Data problematic regarding HDS and advanced data analytics are discussed in Section V. Section VI provides an analysis for suitable deployment of Micro-Services in HDS aimed to increasing of flexibility and efficiency of the applications. Section VII offers the case studies with some proven use examples with inset of the appropriate protocols (e.g. REST, WebSockets, AMQP); 5) An important option for HDS is the deployment of Blockchain-conform structures [3,4,7] due to necessity of compulsoriness of the services and workflows for HDS; 6) Conclusions and the Outlook are given in Sections VIII and IX.

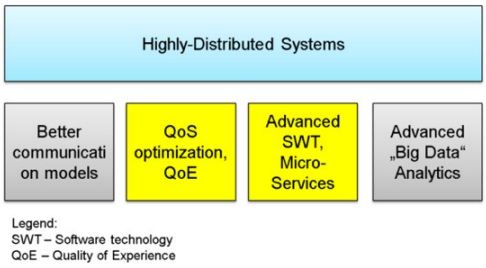


Fig. 1. To the motivation on HDS

II. Advanced Communication Models

This Section discusses the communication models for HDS like C-S, P2P, M2M as well as their efficient combinations aimed to flexible data and message exchange between the advanced architectural components under considering of performance (data rates, latencies), security and privacy and energy factors.

There are three main types of communications models, which are widely deployed in HDS: Client-server (C-S) communication model, Peer-to-peer (P2P) communication model, Machine-to-Machine (M2M) communication model.

Machine-to-Machine (M2M) communication is a communication model that involves one or more instances that do not necessarily require human interaction or intervention in the process of communication (Fig. 2). The applications are used in automated and half-automated modes. The typical scenarios are sensor networks, telemetry, IoT and robotics. The main differences to C-S and P2P models (refer the previous sections) are as follows: M2M applications generate short "bursts" on periodic data packets (so-called short telegrams); The communication is realized typically via asymmetric links; M2M communications are established through uplink channels (UL); Uplink traffic is bigger than downlink (DL) traffic (direct channels from providers) [5].Usually, M2M traffic is machine generated and does not require any human intervention: no human-to-human (so-called H2H). The QoS requirements for M2M communication model differ significantly from regular applications such as in the mobility, delay tolerance on offline and data volumes, on priorities. The automated M2M applications have, furthermore, low power consumption and significant requirements on security and privacy [3,4,7,9].

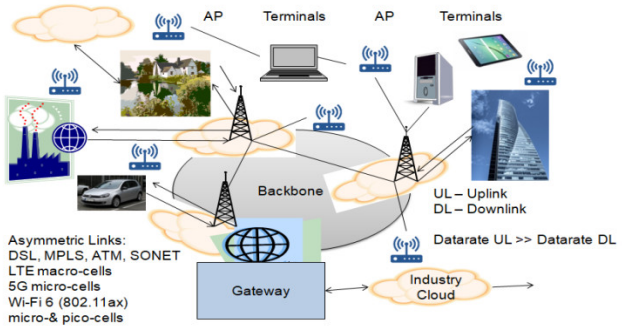


Fig. 2. M2M communication model

In opposite of conventional H2H apps based on C-S and P2P, the serving M2M devices (M2MDs) accumulate and transfer via uplinks the large volumes of sensor data to the specified engineering and industry clouds. This modus operandi can frequently lead to so-called "Big Data" problematic (s. next sections).In frame of M2M model, let's consider a wireless sensor net (WSN), which consists of 30000 sensors. Each sensor can transfer a short telegram up to 100 Bits each minute, i.e. 60 times per hour and 1440 times daily. Thereby:

- The survey for each sensor is conducted 60 times per hour: 6000 Bits/h;
- x 24h = 144000Bits/daily = 18kByte/daily for each sensor;
- In general, an average sensor accumulates experimental data for 3 years x 365 days ~ 1000 days;
- It means: 18 Mbyte for each of the sensors;
- The overall-data for the mentioned network: 18MBytes x 30000 sensors = 540GByte of raw data!

The above mentioned advanced communication model is widely involved to new industries, marketing, event, entertainment scenarios under lower operation expenditures (OPEX) and under robust energy-efficiency. A large number of communicating terminals is used (mobile, desktop) under relatively small traffic per terminal (refer Fig.3). Frequently, so-called "digital twins" for machines, automotive, plants are considered which use M2M too [5].

III. Advanced Methods for Performance Management and Optimization

This Section discusses the methods for performance management and optimization, which are typical for so-called HDS. Inter alia, the term QoE (Quality of Experience) regarding to the HDS can be widely used. Under QoE (Quality of Experience) we understand system performance using subjective and objective measures of customer satisfaction. It differs from quality of service (QoS), which assesses the performance of hardware and software services delivered by a vendor under the terms of a contract. The origins of the term

went from 5G whitepapers in last years, which discuss the advantages of IMT2020 from, e.g. LTE with typical QoS requirements [6-10].

Nowadays, IT and electronics industries apply the QoE principle more and more for wide spectra their businesses and services. Because QoE depends on durable positive customer experiences, the assessments are compiled from large user group polls [8,11,12]. The most important criteria toQoE assessment for fixed, wireless and mobile networks and devices are as follows: 1) Durable QoS within the service (covering) area as well as simultaneously on the edge of the network; 2) Application criticality, for example, simple texting versus audio/video is inacceptable; 3) Robust working environment, stable handover and roaming, as well as inter-operability (fixed or mobile).

Thus, in contrast to QoS, QoE not only depends on the technical performance, but also on a wide range of other factors, including content, application, user expectations and goals, and context of use. Understanding QoE, thus demands for a multi-disciplinary research approach that goes beyond the network level. In particular, different applications have different QoE requirements (also including different QoS-dependencies), necessitating different QoE models, monitoring and eventually, different QoE management approaches.

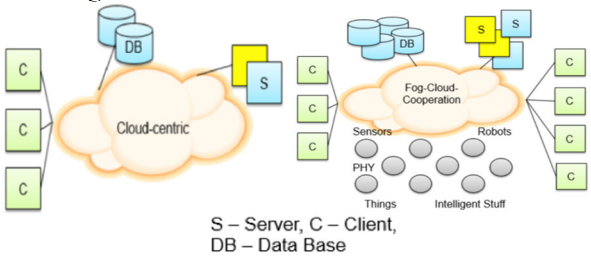
A. Performance Optimization. QoS Parameters

Modern networking uses widely management of QoS-parameters aimed to Performance, Reliability and Scalability optimization [8,11,12]. So-called IoT is based nowadays on IPv6. This brings more freedom in addressing of immense quantity of available devices: sensors pico-nets, Embedded, Wearable, Cyber-PHY, robots, intelligent stuff etc. Huge as well as heterogeneous data volumes (approx. 100PB to 100EByte) are acquired additionally causing “Big Data” shortcomings [8,11,12]. The processing and upload- and download-functionality for sensor pico-nets and robotics is offered via Cloud and Fog systems in their cooperation. The small intelligent nodes in IoT-scenarios communicate via energy-autarky gateways with the capable server part. The considered approaches are able to increase the performance, reliability and scalability in desktop applications and IoT both. Which further approaches can be applicable? Let’s discuss how these affect the following QoS criteria like throughput, response time, and probability of failure, availability, and reliability? Table I represents the influence if the listed approaches on the QoS parameters within an IoT system in detail [11,12].Further performance optimization can be reached via the analytics migration into the clouds. The cloud-centric systems can discharge the energy-critical mobile nodes [8,11,12].A good balance between C and S parts brings use of fog systems. The sensors, robots, intelligent stuff as well as further Cyber-PHY operate partially autonomously (SLMA – server-less mobile apps). Such autarky is possible via energy-efficient communication protocols and software (cp. Automation ML, OPC UA, MQTT, AMQP). An explanation of such important parameter like reliability, which is expressed per average downtime and availability classes 1-7, is given in [11,12].

B. Analytic Placement for the Performance and Scalability

The up-date highly-distributed apps (desktop as well as mobile) are characterized via multi-layer horizontal and vertical architecture. These layers can be as follows PHY world and hardware; Software with interfaces (heterogeneous and adaptive); Middleware components and web services; Analytic blocks and mobile agents. The layers and tiers are combined and balanced between client, cloud and fog part considering the QoS parameters like performance (throughput, response time), reliability (probability of failure, availability) and scalability too. As appropriate examples, OPC UA (OPC Unified Architecture) and programming framework, standardized by IEC 62541-2015, as well as ROS (Robot OS) and programming framework [11,12] can be considered. The IoT and robotic applications can be classified into three following groups: 1) Conventional IoT and robots; 2) Cloud-Centric IoT and robots; 3) Distributed (Fog-Cloud-cooperating) robots.

Therefore, we are talking about replaceable and customizable IoT and robotic algorithms in various fields of application (industry, medicine, communication and telecommunication, entertainment) which can acquire, then process and retrieve voluminous heterogenic “Big Data” in the given area [8,9,11,12]. The new approach is depicted in Fig. 3: performance optimization is possible due to migration from clouds to fogs.



a) Less performance for Analytics: servers, DB and clients
b) More performance for analytics: servers, DB and clients

Fig. 3. New Approach: Performance Optimization due to Migration in Clouds and Fogs

Only the cloud-centric solution (2) and further distributed, fog-cloud-cooperating solution (3) both are able to overcome the discussed problems in full measure. The analytic blocks, migration agents as well as further adaptive interfaces are delegated to the clouds and, possibly, after pre-processing and clustering backwards to the so-called “fog” under use of the mentioned solutions and protocols. The virtual analytical components (middleware, web services, and mobile software agents) are placed in the cloud and fog environment. Virtual cloud and fog solutions contain software components for the robots that implement reboot-able (virtual) business processes

IV. Advanced Software Technology for HDS

This Section investigates several advanced SWT (Software Technology) methods and process models, which can optimize the construction of modern HDS, increase their efficiency and reduce the expenditures. A process model (Fig. 4) organizes a workflow for creative production and SW development into various, structured sections and phases, which are assigned to

corresponding technologies, tools, languages, protocols and methods of the enterprise, organization or industry. The first of them, the mostly used agile approaches like RAD, XP, DevOps, Kanban, Scrum must be mentioned and, the last but least, so-called Micro-Services. Such advanced languages and notations like UML, XML, BMPN as well as Automation ML are widely used for HDS construction [1,2,8,10,11,12].

As agile model, DevOps (2009) uses a close correlation between software development and operation teams. The similar successful agile techniques are known as Kanban (with origins by Toyota) and widely spread Scrum (refer Fig. 4).

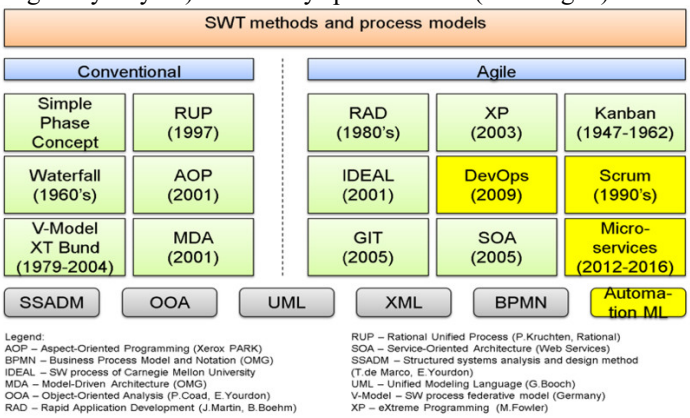


Fig. 4. Overview of the process models

Scrum (1990's) is a virtual analogy to rugby sports and describes the successful teams, which are working on the development of a project or SW product together and involved into close human communication. As an iterative SWT process model, Scrum does not describe classic project phases, but instead of attaches the values of so-called continuously usable results (artefacts') right from the beginning of the project. The following slogans are used in the industry:

- *Slogan 1: "Do twice in half time" (refer XP).*
- *Slogan 2: "One for all and all for one" (a musketeer slogan).*

Automation ML (2006-2009) is widely used as markup language (XML, *ML) and established protocol for sensor pico-nets and IoT solutions. Automation ML supports a domain-specific exchange format for IoT and can cooperate with advanced SOA as well as Micro-Service architectures (refer Fig. 7). Automation ML allows explicitly incomplete or inconsistent descriptions of the PHY devices, machines or, even, a plant and can provide benefit in early stages of the planning. Automation ML is supported via a number of the vendors for automation tools and wireless networks. However, the technique can be nowadays understood as WIP.

The following use cases are typical for Automation ML: Data and message transfer from robot simulation system to robot specific programming system; Data and message transfer from mechanical design to functional (electrical and PLC) engineering; Data exchange between CAD systems; Interface from CAD system to documentation system.

V. Advanced Data Analytics regarding to "Big Data" problematic

"Big Data" accumulation is nowadays typical for trading and marketing, electronic payments, production process, for the traffic from mobile providers, international justice and forensics, for public fiscal authorities, pharmaceutical and advertising industry. A large number of research institutes, organizations and universities accumulate, store and process large amounts of technical and scientific information [8,11,12].

In the conditions of modern industrial development, so-called "Industry 4.0", there are even more "Big Data" sources: home automation, patient health data, M2M and robotic data, business intelligence, pharmacological research, networking and experimental data [11,12].The mobile networks and aps for the 5G will definitely take an active part in the process of receiving and processing of large data amounts [8,11,12] too. By year 2020, the new 5G networks will use more than 50-100 billion sensors to download comprehensive information about how we interact with things that surround us or that are even inside of us?!As one of the most interesting further topics the Blockchain technology occurs. This is nowadays exponentially increasing and enables modern crypto-currencies: e.g. Bitcoin, Monero, Ethereum etc.[3,4,8]. The technology of Blockchain and its associated applications like crypto-currencies are so-called "resource eaters" due to their enormous energy and memory consumption. Large amounts of chained crypto blocks are causing surely "Big Data problematic" [8,11,12]. For the processing of "Big Data" the usual statistical concepts can be deployed like S, R, SPSS, Oracle R, SAP Hana, IBM SPSS, Netezza, Grafana. On the other hand, there is a Big Data Appliance is NoSQL-Cluster from application servers for massive-parallel analysis based on the integrated R tools and Apache Hadoop. An advanced concept for overcoming of "Big Data" complexity, which uses proven freeware for Linux-clusters and connectors to the conventional DB was discussed in [11,12]. The complex poly-structured and redundant retrieved data can be processed with higher performance within an enterprise or institution data center. Some in Java implemented modules allow even real-time control. The expenditures in the form of investments CAPEX (Capital Expenditures for hardware, cable infrastructure, premises) and OPEX (Operational Expenditures for licenses, personnel, electricity, ongoing maintenance) are significantly reduced. The discussed concept delivers agility, possesses no rigidity due to only small license costs. The components were [11,12] as follows: Apache Hadoop, Apache HBASE, Apache Phoenix, Apache Hive, Tableau, Talend and SCADA (Supervisory Control and Data Acquisition).

VI. Evolution and Transition to Micro-Services

Monolithic and quasi-monolithic architectures is factually the term of 1960's: no compilers, no OS, early modularization, dependencies conflicts. So-called (quasi-)monolithic software application is a software application composed of modules that are not independent from the application to which they belong. The main disadvantages of such approach are as follows: Complex maintenance; Poor scalability; Unified deployment

configuration; Dependencies conflicts. Then, as a challenge: more flexibility, SWT compromises necessary and less coupling is necessary!

A. Early Architectures

The evolution of distributed applications followed historically from OOP through Web Services and SOA to Micro-Services. The slogan in software technology (SWT) was always as follows: *„Always loosely coupling!" (the new Slogan 3).*

The mostly appropriate approach is SOA (2005 -2010) based on Web Services. The Web services are not the same as SOA, but they can be used in a SOA. Web services are not synonymous with Web applications, but they can be used by Web applications. The Web application additionally provides a GUI for entering or displaying information for or from the Web Services. The next stage: Micro-Services with more flexibility and less coupling (refer next sections, refer Fig.5).

Please notice, we as the authors are rigorous against the use of the term "monolithic architectures" [1,2] regarding to the conventional systems, but offer the use of so-called "quasi-monolithic", or, correspondently, "modular", "object-oriented", "component-oriented", "the architectures with macro-services" etc. They are anyway not "monolithic" (refer Fig.5).

B. From quasi-monolithic architectures to Micro-Services

The discussed SWT methods and process models can be classified into conventional and agile. The agile SWT methods and process models are widely used for distributed systems and HDS too. Nowadays the Micro-Services gained on the meaning under use of firstly DevOps and Scrum. The both process models support service-oriented approach: SOA and Micro-Services.

Typically, UML and BPMN for workflow notation are used. For IoT aps the Automation ML is mostly attractive. What does it mean Micro-Services? The Micro-Services are a concept for modularization, a specific organization and SWT approach simultaneously. The components of such concept are single Micro-Services, which work independent and technically oriented (cp. SOA).Micro-service architecture can be represented as a large entity across all individual Micro-Services The both mostly popular slogans for Micro-Services deployment are given below:

- *Slogan 4: Unix-Philosophy („Do One Thing and Do It Well!")*
- *Slogan 5: Two pizzas teams (6-8 people can be with satisfied two big pizzas)!*

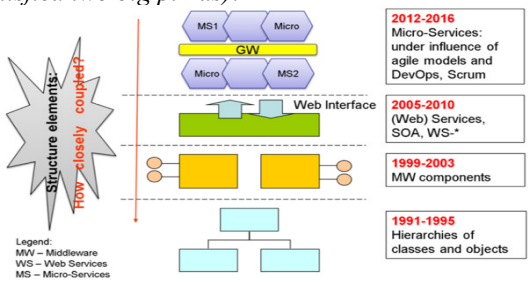


Fig. 5. From quasi-monolithic architectures to Micro-Services

Under use of above-mentioned concept, a quasi-monolithic application is composed of several inflexible modules or macro-services that are not fully independent from the application to which they belong and one each other. The deployment is carried-out with complexity, that means less scalability and complicated configuration is only available. So-called dependencies conflicts occur often too [1, 2]. On the other hand, the application, which is based on Micro-Services consider the necessary technological trade-offs and offer more flexibility, loosely coupling of the modules and components, as well as is easily configurable (Fig. 6). A comparison of Micro-Services to so-called quasi-monolithic architectures with internal look is given in Fig. above. A demarcation of the Micro-Services to conventional SOA and Web Services is represented in Table I.

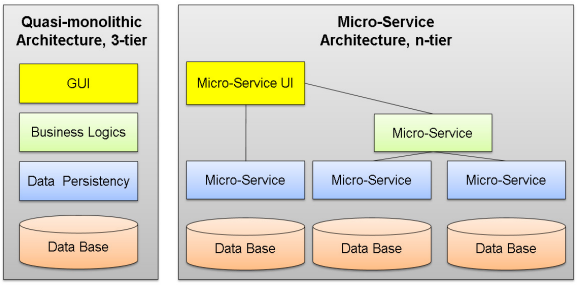


Fig. 6. Micro-services: internal look

TABLE I. DEMARCATION OF MICRO-SERVICES TO SOA

Both SOA and Micro-Services use services as architectural elements.	
SOA uses the services for integration of different apps (EAI).	Micro-services bring a structure to an App under use of the services.
The combination of the services is "orchestrated" or "choreographed" (so-called Orchestration or Choreography), and the portals can provide a common GUI for all services.	Each Micro-Service can include a GUI and implement the business processes in similarity to a SOA with Orchestration.
Conclusion: loosely coupled, more flexibility.	

C. Differences of Micro-Services to Conventional Architectures

Differences between the traditional architectures and Micro-Services are represented in Fig.7 on the example of a supermarket with established (macro-)services for customers, products and carts.

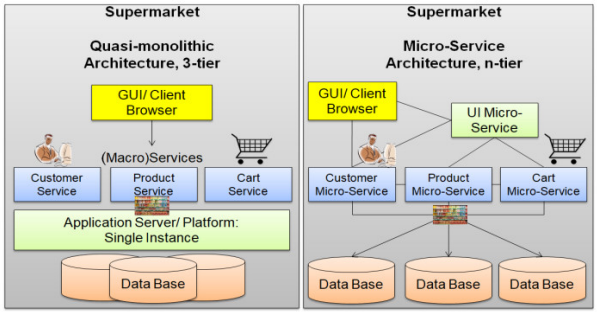


Fig. 7. Differences between (Quasi-)Monolithic Architecture and Micro-Services given on the example of a supermarket application

The main difference that we can observe in Fig. is that all the features initially were under a single instance (a platform or

an application server) sharing a database with further replications. That means an additional performance loss due to consensus support functionality: commit protocol between replicated DB required. Then, under use of Micro-Services, each feature can be deployed via a dedicated Micro-Service, handling their own data and performing different functionalities (refer Fig. 7).

D. Important Paradigms for HDS

The deployment of Micro-Services is mainly based on the three following concepts [1]: CAP Theorem (1), Conway's Law (2), Domain Driven Design (3). Let us to discuss these paradigms more in detail. So-called Conway's Law plays a very important role for Micro-Services deployment and need a more detailed overview [1]. Conway's Law possesses the following distinguishing features:

1) Organization of the development department, which is separated by functional groups.

2) Functional-specialized departments are divided to the following Expert Groups: Experts for GUI; Experts for Business Logic; Experts for DB.

3) Functional-separated architecture, which includes: Web App with Backend Access; Backend Apps with logics for business functions; Regional DB with fixed Data scheme;

4) Domain limited cross-functional teams contains: Registration and management team; Rating and Matching Team; Team for Chat function

5) Architecture based on domain-oriented enclosed entities includes: Independent Registration Component; Independent Component for Rating and Matching Functionality; Independent Chat Component.

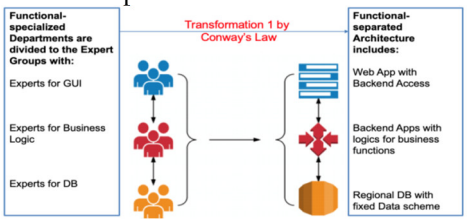


Fig. 8. Conway's Law: Transformation 1

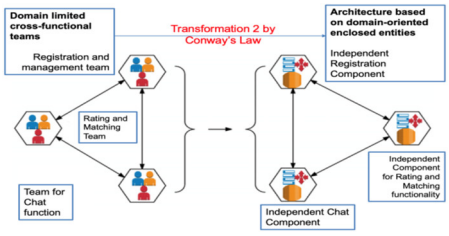


Fig. 9. Conway's Law: Transformation 2

Furthermore, Conway's Law offers two basic transformations for the software development process which are derived from DevOps and Scrum process models (refer previous sections). The first transformation T1 is shown in Fig.8. With T1, a quasi-monolithic system is transformed for each app type and for different teams of software engineers: it means the experts for GUI, experts for business logic as well as for persistency layer. The second transformation T2 is shown in Fig. 9.The whole development department is structured by several functional teams via the discussed T2, which assigns an independent component to each team (Registration, Rating, Matching, Chat etc.).

Domain Driven Design [1] is the third important paradigm to deploy Micro-Services and construct the flexible and performing HDS. This paradigm is deployed mostly tighter with Conway's Law (refer Transformations 1 and 2 above). The most important positions of the paradigm, which the discussed approach can distinguish lucratively, are as follows: Creation of special domains for SWT; Use of bounded contexts; Breaking complexity of the developed projects

E. Micro-Services Platforms and Frameworks in comparison

The following famous Internet and cloud services are known as steady consumers of the Micro-Services, and namely: Google, Amazon, Twitter, eBay, Spotify, Otto, Azure Service Fabric, The Guardian, Spring Cloud, Kubernetes [1]. The development of the Micro-Services can be supported via the well-known frameworks (FW) as follows: Spring Cloud, Netflix OSS (Open Source System), Kubernetes. The implementation of a Micro-Service architecture is a complex software-technological problem. However, there are a few difficulties, which must be nearly considered (refer Table II).

TABLE II. MICRO-SERVICES PLATFORMS AND FRAMEWORKS

Netflix	Spring	Kubernetes
Micro-service Framework Primary support of internal Netflix-Apps Partially available as OSS Comparable implementation for multiple similar functionalities within the referenced FW	Micro-service Framework Several tools have been adopted from Netflix Rebranding von Netflix OSS as Spring Cloud Implementation of multiple functionalities is comparable Expanding ecosystem Java-based technologies	Micro-service Framework A little bit other concept Multi-language runtime platform Expanding ecosystem Implementation of multiple functionalities is comparable

VII. CASE STUDIES

This section represents some examples on the discussed subjects. Under use of agile SWT process models, Micro-Services, available tools and variety of application protocols the modern apps appear which corresponds to HDS paradigms and possess advanced features like QoE, performance, small latencies and high flexibility as well as are autarky and energy-efficient. Such apps are, in addition, better scalable and more secured via a better management. One of possible options regarding to security, privacy, authentication and compulsoriness of such apps for HDS is use of Blockchain infrastructures [3-5,8].An option is deployment of Ethereum, which provides authentication and compulsoriness of execution (or not execution) of some modules, (macro-)services and Micro-Services within an established SW system. A first example for such deployment of cryptographic-supported workflow Smart Contracting represents.

A. Example 1. A Simple 1x1 Implementation

Fig. 10 depicts use of Micro-services 1 and 2 with simple Request-Response functionality. The MS1 and MS2 can use HTTP and WSDL are interoperable to Web Services. Otherwise, they can be deployed for flexible construction of the IoT app via GW under use AMQP (Advanced Message

Queuing Protocol) for asynchronous messaging between MS1 and MS2.

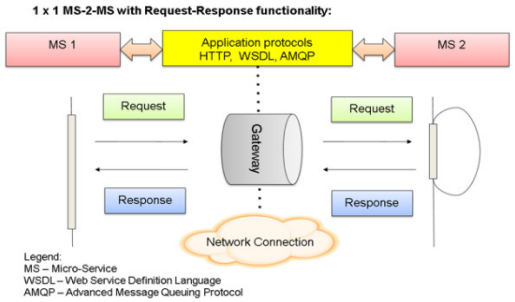


Fig. 10. MS-2-MS: 1 x 1 -integration with Request-Response functionality

B. Example 2. Micro-Services with n x n -functionality integration

An alternative application is provided via Fig. 11. The architecture components for Micro-Services are depicted below. The depicted GW enable to communicate under use of variety of protocols, inter alia, HTTP, RPC (JSON-coded) and WebSockets. WebSockets (2010-2011) is a L5-7 application protocol (refer Fig. 11), which operates over TCP-connections and is oriented to message exchange between Web-browsers and Web-servers in real-time.

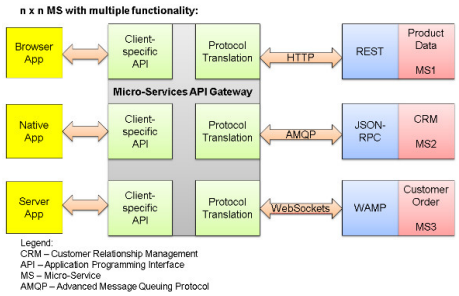


Fig. 11. Multiple MS with n x n -functionality integration

WebSocketsare quite different from well-known HTTP but the both are compatible. WebSockets protocol was designed to work over HTTP ports 80 and 443 as well as to support HTTP proxies. Nowadays W3C develops the stable standard for API Web Sockets because of many existing drafts. Since 2010-2011 the mostly known versions v6, v7, v13 are used. The beta-version of the stable standard is known as IETF / RFC 6455. WebSocket API in Web IDL is being standardized by the W3C. Furthermore, WebSocketsare used to implement different client or server apps. WebSockets protocol enables a close interaction between browsers and servers with interactive multimedia presentation and provides a full-duplex communication. WAMP Stack means the server under Windows OS, which consists of Apache, MySQL and PHP (refer Fig. 11) and provides WebSockets/HTTP interoperability.

C. Example 3. Multimodal X-platform Implementation based on REST

Firstly, it enables so-called Continuous Delivery approach [1] and can be deployed for large and complex applications. At second, there are improved maintainability: i.e. each service is relatively small and so is easier to understand and change it. It enables you to organize the development effort around multiple, autonomous teams. Each team (under so-called two pizza team slogan) owns and is responsible for one or more services. Each team can develop, test, deploy and scale their services independently of all of the other teams. Each Micro-Service is relatively small, i.e. easier for a software engineer to understand and optimize it. On the other hand, IDE is faster making the software engineering works more productive [1].The application starts faster, it speeds up the deployment and improves the fault isolation. For example, if there is a memory leak in one service then only that service will be affected.

The other services will continue to handle requests. In comparison, one misbehaving component of a quasi-monolithic architecture can bring down the entire system. The approach eliminates any long-term commitment to a technology stack, when developing a new service you can pick up a new technology stack [1]. Fig. 12 depicts a solution, which possesses a number of such benefits.

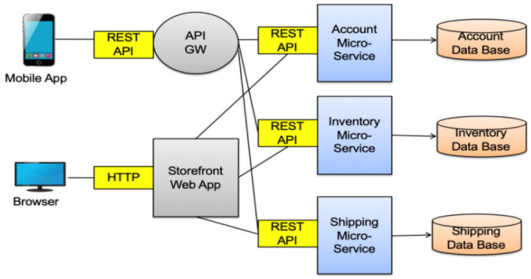


Fig. 12. Example on implementation

D. A Short Assessment

A short assessment for the above discussed Micro-Services architectures is represented in Table III. The further successful examples of Micro-Services implementation are as follows: Netflix, eBay, Amazon, the UK Government Digital Service, Twitter, PayPal, The Guardian, and many other large-scale websites and applications have all evolved from monolithic to Micro-Services architecture [1].

TABLE III. ADVANTAGES AND DISADVANTAGES OF MICRO-SERVICES

Pros	Cons
Quasi-parallel execution extended toolset and management tools flexible use of technologies easier refactoring and reduced risks for source code adaptation high load availability for the system with integrated Micro-Services attractive employer sustainable software with better modular scaling and code reusability resource-efficient hosting	Complexity is hidden behind distribution on micro-level Distributed complexity of conflict resolution by Micro-Services intercommunication Replications are often required Navigation over the entire system is often too complex RPC/RMI-based communication (remote method calls) is time-critical, lack of performance must be considered Software tests are becoming more expensive Testing and deployment are more complicated, therefore, reusability of code is restricted Falling innovation factor, some disappointment and frustration.

VIII. Use of Blockchain in HDS. Smart Contracting

One of the mostly important Blockchain applications after the mining of the crypto-currencies are so-called Smart Contracts [3-5,8]. Historically, Smart Contracts (SC) doesn't require exceptionally Blockchain, but certain consensus algorithms (protocols), which are cryptographically conditioned via hashes, private and public keys and signatures. A "smart" contract is a software-based agreement that allows and can contain a variety of contract terms. In the course of the usual contract processing (transactioning), certain linked actions can be executed automatically if there is a corresponding trigger. The contracts are offered and signed within and via the Blockchain or other Blockchain-like infrastructure. The evident advantages of the discussed approach are as follows:

- Digitality and legal openness of the platform;
- Transparency, costs and time savings;
- Automation of the workflow step processing;
- Deployment at the HDS solutions for compulsoriness.

Smart Contracts within the HDS are driven via Blockchain-conform cryptographic structures, which provide compulsoriness of required workflow steps and predictable execution of the deployed modules, services, Micro-Services and of other components within the internal architecture of the above-mentioned HDS. The mostly appropriate environment for SC is a private Ethereum-Blockchain. However, Ethereum doesn't work completely conform to European Laws. An example of a Smart Contracting application for an e-Vallet for the ICE trains of DB.de is given in Fig. 13.

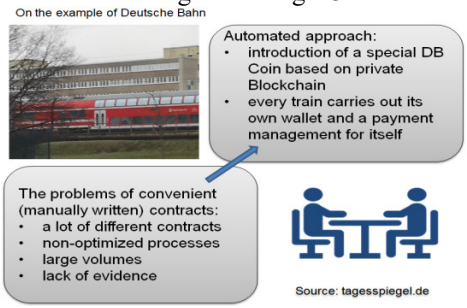


Fig. 13. Smart Contracting on the example of DB.de

Furthermore, the Blockchain is used for Smart Contracting applications for the following world-wide leading companies deployed: Walmart, Maersk, Alibaba, CartaSense, Kuehne + Nagel (aimed to logistics, sea freight, stock exchanges, marking of containers), Nestlé, Tyson Foods, Unilever (aimed to food delivery), Everledger (the registers for diamond certification) [3-5,8]. Belorussia is the first national economy nowadays, which has recognized SC completely. The criticizers speak about the wrong ethic side: SC leads often to reducing of the available jobs and distortion (corruption) of the one of the most ancient social institutes of contracting law over the world.

IX. Conclusions and Outlook

A definition for the HDS was given, as well as the demarcation to conventional distributed systems offered. Typical architectures for HDS were discussed. The QoS requirements were specified; the mentioned parameters affect the increasing of QoE. The distinguishing features for HDS are clearly formulated and were described. Deployment of modern SWT approaches lead to use of flexible service-oriented architectures, inter alia, of Micro-Services, which provide advanced features like QoE, higher performance, small latencies, better flexibility as well as are autarky and energy-efficiency. Such apps are, in addition, high scalable and more secured, as well as offer better management and reconfiguration. One possible option in the frame of HDS regarding security, privacy, authentication and compulsoriness of workflow steps, modules and service execution for such apps Blockchain and Smart Contracting are. The theoretical issues are proven via the represented examples and case studies. This work can be positioned as a Work-In-Process. There a lot unclear positions with Micro-Services due to necessity of conflict solving between them and some difficulties with use of development tools.

Acknowledgement

Authors' acknowledgements to the students of BA Dresden, SAP Dresden, the director of BA Dresden Prof. A. Haensel, the supervisor for international cooperation Mrs. I. Scherm, colleagues from Lviv Polytechnic National University Prof. M. Klymash, Dr. T. Maksymyuk, Dr. M. Beshley and colleagues from TU Zilina Prof. T. Zaitseva and Prof. V. Levashenko, as well as to friends for technical support, inspiration and challenges by fulfilling of this work.

References

[1] S. Newman. Building Micro-Services, Publishing by O'Reilly Media, USA, 2015, ISBN: 978-1-491-95035-7, 473 p.
[2] B. Fekade et al., "Clustering hypervisors to minimize failures in mobile cloud computing," Wireless Communications and Mobile Computing, vol. 16, no. 18, 2016, pp. 3455-3465.
[3] MIT Blockchain Course (Online 20.10.19): http://executive-education.mit.edu/MIT-Blockchain/Online-Course/.
[4] A. Antonopoulos. Mastering Ethereum: Building Smart Contracts, 2019, O'Reilly Media, 345p., ISBN978-1491971-949.
[5] Machine Type Communication (MTC) in 3GPP (Online 20.10.19): http://www.3gpp.org/
[6] T. Maksymyuk et. al., "Cooperative channels allocation in unlicensed spectrum for D2D assisted 5G cellular network, " IEEE Int. Conf. on Advanced Information and Communication Technologies (AICT), July, 2017, Lviv , Ukraine, pp. 197-200.
[7] A. Luntovskyy, M. Klymash. Software Technologies for Mobile Apps, Apps for Fog Computing, Robotics and Cryptoapps, Lviv, 2019, 247 p. (Monograph, Ukrainian, ISBN 978-617-642-399-7).
[8] A. Luntovskyy, J. Spillner. Architectural Transformations in Network Services and Distributed Systems: Service Vision. Case Studies, Springer Nature/Vieweg, 2017, 344p. (ISBN: 9-783-6581-484-09).
[9] T. Maksymyuk et al., "Blockchain-based intelligent network management for 5G and beyond," 3rd IEEE International Conference on Advanced Information and Communications Technologies, (AICT'2019), July, 2019, Lviv, Ukraine, pp. 36-39.
[10] A. Luntovskyy, D.Guetter, I.Melnyk. Planung und Optimierung von Rechnernetzen: Methoden, Modelle, Tools fuerEntwurf, Diagnose und Management imLebenszyklus von drahtgebundenen und drahtlosenRechnernetzen, Handbook, Springer Vieweg+Teubner Wiesbaden, 2011, 435 p. (ISBN 978-3-8348-1458-6, German).
[11] A. Luntovskyy, L. Globa. Performance, Reliability and Scalability for IoT, IEEE Conf. IDT-2019, Zilina, Slovakia, June 2019.
[12] R. Minerva et al., "Towards a definition of the Internet of Things (IoT)," IEEE Internet Initiative, vol. 1, no. 1, pp. 1-86, 2015.