Nanoservices as Generalization Services in Service-Oriented Architecture

Sutrisno, Frans Panduwinata, Pujianto Yugopuspito
Informatics Department
Pelita Harapan University
Tangerang, Indonesia
sutrisno.fik@uph.edu, frans.panduwinata@uph.edu, yugopuspito@uph.edu

***Abstract*—In software methodology, a service is related to an application in internet technology whereas web-based application is popular in the service-oriented architecture. SOA is a framework for developing an application in the enterprise system. It builds and runs on top of a web-based technology. Each service represents a business activity, and the technology supports this services via an operation known as a web-service. Meanwhile, a microservice architecture is a fine-grained of the web service. So microservice is an approach in implementation phase of a SOA. It is a bottom-up approach in application software development that is potentially used for small company and focused on a specific product. In this paper, a nanoservice is introduced as an implication of transforming a web service into microservices that yields some similarities or intersection among microservices. We showed that a nanoservice is an existence of generalization of microservices, as web service API as an example.**

Keywords—software methodology; nanoservice; microservice; web-service; service-oriented architecture

I. INTRODUCTION

This paper is a continuation of the application migration from a monolithic application to adapt service-oriented architecture [1]. We observed that RESTful application leads to microservices, and faces a new terminology, nanoservices. As web service is an operation or functionality works in server which is invoked by client application over HTTP internet-based technology. The web services gain popularity and much relating to service-oriented architecture (SOA) or service-oriented computing (SOC). And, the service-oriented architecture raises interest for enterprise systems community in developing distributed applications over web-technology. Service-oriented architecture is an application software development approach focus on business entity. One of the issues in business-driven application is application should keep adapt to business changes, because the business change is nature of the business entity. This issue is stated as one of the issues statement in SOA manifesto [2].

Transforming a web service that works on vary different entity data domain (e.g. books, automotive, electronics) or consist some tasks into microservices, will generate number of similar microservices. Therefore, the microservices have similarities on the task, also different, they have intersection. This paper will explore and discuss on transforming a web service into microservices, also introduce a redefine term nanoservices, as part our contribution.

The structure of this paper is as follows: Section II presents different efforts related to the nanoservices terminologies. The Enterprise Systems and Service Oriented Architecture is presented in Section III. Section IV describes the application deployment as Web Services and Microservices. Section V presents our finding on nanoservices. Finally, Section VI concludes and presents several research lines that can be addressed.

II. RELATED WORKS

One of the SOA frameworks proposed which is called as Simple Object Access Protocol (SOAP). In service-oriented architecture, informally service can be described as object with business capability. Implementing SOA using web services has some critics, because difficulties and complexity [3]. Therefore, microservice architecture proposed as second generation of SOA or SOC.

In the microservice architecture, a microservice has a single functionality and loosely couple task [3] that expected to be highly cohesion within the operation. A microservice promises to support end-to-end application, and the application can develop continually to keep adapt to business changes. Furthermore, a microservice application can work or delegate the task to a group, so it focuses and responsible to evolving and continuing application development [4]. Developing an application which applies microservices architecture, developer have to take care for each service designed only one functionality and loosely couple. For each developer team, has independent to choose database management system used in the application, data structure, algorithm, and the programming language used, for implement the application.

An evolution of webservice has been predicted in [5], and nanoservice is stated as a fine-grained service. The term of nanoservice also reported in [6] a result of migrating native architecture using microservices [7]. So there are several opinion about this term.

Reference [8] stated that microservice is another word of SOA, this is related to service-oriented delivery issue. As agreed that microservice is a choice of implementing enterprise system in SOA. While [9] states microservices and

978-1-4673-9899-2 /17 $31.00 © 2017 IEEE
DOI 10.1109/ICSIIT.2017.21

nanoservices are independent deployment unit as supported by [4]. Nanoservice is smaller than microservices and they can interact among them. It can be a microservice stack like Docker and Spring Boot. A Docker can be run independently as virtual machine on top of a host server.

The idea of independent development is an open choice among available software architecture as well as the used of programming languages. So the problem of "micro" is in the size, as [10] with the GorilaStack framework as part of AWS Lambda, micro means 10 – 100 line of codes. But [11] stated nanoservices is not about an anti-pattern, as an experience in lambda function as the controller of microservices in both EC2 and AWS. Nanoservice is a single action or responsibility.

According to the size, [12] argued that LOC size is not proper for microservice, because this term will lead to a new term nanoservice, less than 10 LOC, but it depends on the programming language. It also observed that nanoservice tends to finest-grained service and becomes an anti-pattern. So the term nanoservice is good to be implemented in iterative design phase, and tight related to its functionality.

III. Enterprise and Service Oriented Architecture

Enterprise system businesses characterizes with dispersing unit business also geographically location. Therefore, applications in an enterprise system also distributed, their applications are connected through network. Integrity and consistency data on the enterprise system is one of being main concerns. As an implication, centralized design approach is chosen and applied to assure obtaining those concerns.

Internet technology by means of web technology is de facto standard applications built up on top. From simple applications to complex applications use the web technology. E-commerce is one of the applications which are built on top web technology. E-commerce players in supply side or sellers ran by personal to enterprise scale. One of the e-commerce players is Amazon. They provide API's, such application developers can use them.

Distributed is the nature of enterprise systems, and in another hand, to consolidate data integrity and consistency. Meanwhile the interactions among applications through robustness applications are the main issues in building an enterprise system. Business changes are also nature of the businesses and should be faced and adapted by enterprise systems. The business changes will impact to the software application as the supporting technology embedded within their systems. Versioning and continuing development are implications in software development for always adapting to the business changes. Many software development methodologies and frameworks are proposed which are expected to solve the enterprises problem, such as model driven architecture, rational unified process, and service-oriented architecture [13].

Service-oriented architecture (SOA) or service-oriented computing (SOC) is a framework in developing application approach has popularity in web-based technology environment. In the SOA approach, it is top-down design approach, works start from conceptual design to implementation action. The SOA concerns start from the business perspective, and technology follows the business. The SOA communities declare principles which are called as SOA manifesto [2]. Some points in the SOA manifesto are shared services, evolutionary refinement, and sustainable business through adapting to the business changes.

In the technology side, the SOA service accommodated in which is called as web services (or just *services*). Web service is an operation or functionality on top web-technology, HTTP, REST, and SOAP. Client request an operation works in remote server within application through invoking a web service. A service provider provides APIs, and an application can request services through embedding within the application. Example of the requesting web service will be discussed in the next section.

Generating a web service, it starts from the business activity concern within the context of enterprise system. An enterprise system typically has many and wide product lines, therefore, each web service is designed to handle and accommodate some product lines. A single web service (operation) handles some varied product types (domain), such as book, apparel, music, electronic, etc. To ensure integrity of data or information, an enterprise tends to use homogeneous technology infrastructure, such as DBMS. Besides maintaining data, it is also forcing the usage of the same application software, or the same programming language, in all phases of the development including the application development.

IV. Web Services and Microservices

In [3], microservice is defined as a minimal independent process interacting via messages. And, the term "minimal" represents functionality concern only on the service. In the microservice architecture, a microservice is an independent component. All components in microservices architecture are microservices. Microservices itself is second generation of SOA and SOC.

In Service-Oriented Computing (SOA), a program is called as service, also named as web service, which offers functionality to other components via message passing, or parameter. Componentization is the main concern in service-orientation, and closely related to object-orientation. Object in object-orientation concerns on encapsulation and information as shared-memory, while service focus on independent deployment and message-passing, and also added by business capability, or domain problem. Encapsulation and information in object-orientation represents a domain problem, so between object-orientation and service-orientation are closely related.

A web service is an operation or functionality which provides on a server over network, the service does a certain task. An application which needs a server doing a task, should request appropriate web services via web service API, and the application called as web services requester. Common syntax of a web service API consists of service name, parameters, and result back from the service. Each web service application that interest a service should inserts this API into the application within its HTML in the REST environment. In the following is an example of requesting a

service or operation (e.g. ***ItemSearch***, from AWS), and result back from the service called [14]. Fig. 1 and Fig. 2 show the typical fragment of request and response services, as an example.

Invoking the ***ItemSearch*** service, an application should submit at least one parameter to this operation is ***SearchIndex***, it is mandatory parameter. Other parameters such as "***Keyword***", "***Conductor***", "***Actor***", "***ItemPage***", "***Sort***", are optional parameters. In SOC, web services apply concept of separation between interface the operation, and how the operation is implemented. This concept has similar idea with the concept of representation independence which is mentioned in [15]. The ***ItemSearch*** web service is provided by Amazon, and http site the service provider should be declared within the application, Fig. 1. The result response is shown in Fig. 2. In this example, to limit the book searched, the application adds some keywords supply to the machine.

```
http://webservices.amazon.com/onca/xml?
  Service=AWSECommerceService&
  AWSAccessKeyId=[AWS Access Key ID]&
  AssociateTag=[Associate ID]&
  Operation=ItemSearch&
  Keywords=the%20hunger%20games&
  SearchIndex=Books&
  Timestamp=[YYYY-MM-DDThh:mm:ssZ]&
  Signature=[Request Signature]
```

Figure 1. Typical ***ItemSearch*** Request API

As written in [14], when use the ***ItemSearch*** operation in an application, one suggests to aware the parameter, whenever specify ***Actor*** parameter, would not use the ***Automotive*** search. It can be understood, because it can conflict and confuse the search work of the service. ***Actor*** and ***Automotive*** parameter values are different entity domains (product categories). On the ***SearchIndex*** parameter, an application supplies product category as parameter value such as books, apparel, beauty, electronic, and others.

The ***ItemSearch*** provides a number of valid optional parameters for accommodating predefined AWS's product categories. Some optional parameters seem that relate movie product category such as "***AudienceRating***", "***Actor***", "***Artist***", and "***Director***". Other predefined parameters relate to "***Music***" such as "***Conductor***", "***Composer***", and "***Orchestra***". Other parameters seem unrelated to certain product categories, these relate for tuning the service, such as "***ItemPage***", "***Sort***", "***VariationPage***", and "***Power***".

Response result of the ***ItemSearch*** service will either valid, or invalid, which is represented by element of returned response groups such as "***isValid***", and "***Message***". The invalid operation may occur in some conflicting cases, for example when "***Actor***" is submitted in ***ItemSearch*** parameters, and "***Automotive***" as a ***SearchIndex*** value, it potentially raises error.

In Fig. 3, we present diagram steps of invoking a web service interaction between application and web service provider. This diagram uses for studying and analyzing how a web service invoked by application, and web service provider handle the requested service. We use ***ItemSearch*** as study case. An application uses ***ItemSearch*** web service, and parameters values are obtained from user which interacts through HTML. The application invokes the service after setting up relevant parameters that needed for executing the service. The web service provider receives a known requested service API. Parsing and analyzing a script may consist of identity, operation (***ItemSearch***), mandatory parameter (***SearchIndex***), and other parameters. If all pass, it will continue to the next phase. Web service provider will execute search operation based-on the contextual data, and return result back to the application. The contextual data means to a relevant entity data domain, such as movie entity domain, automotive entity domain, and music.

```
<TotalResults>2849</TotalResults>
<TotalPages>285</TotalPages>
<MoreSearchResultsUrl>https://www.amazon.com/gp/redirect.html?
 linkCode=xm2&SubscriptionId=[AWS Access Key ID]&
 location=https%3A%2F%2Fwww.amazon.com
 %2Fgp%2Fsearch%3Fkeywords%3Dthe%2Bhunger%2Bgames%
 26url%3Dsearch-alias
 %253Dstripbooks&tag=[Associate ID]&creative=386001&camp=
 2025</MoreSearchResultsUrl>
<Item>
  <ASIN>0545670314</ASIN>
  <DetailPageURL>https://www.amazon.com/The-Hunger-Games-Trilogy-
   Mockingjay/dp/0545670314%3FSubscriptionId%3D[AWS Access Key ID]%
   26tag%3D[Associate ID]%26linkCode%3Dxm2%26camp%3D2025%26creative
   %3D165953%26creativeASIN%3D0545670314</DetailPageURL>
  <ItemLinks>
    <ItemLink>
      <Description>Technical Details</Description>
      <URL>https://www.amazon.com/The-Hunger-Games-Trilogy-
       Mockingjay/dp/techdata/0545670314%3FSubscriptionId
       %3D[AWS Access Key ID]%26tag%3D[Associate ID]%26linkCode
       %3Dxm2%26camp%3D2025%26creative%3D386001%26creativeASIN
       %3D0545670314</URL>
    </ItemLink>
    <ItemLink>
      <Description>Add To Baby Registry</Description>
      <URL>https://www.amazon.com/gp/registry/baby/add-item.html
       %3Fasin.0%3D0545670314%26SubscriptionId
       %3D[AWS Access Key ID]%26tag%3D[Associate ID]%26linkCode
       %3Dxm2%26camp%3D2025%26creative%3D386001%26creativeASIN
       %3D0545670314</URL>
    </ItemLink>
    <ItemLink>
      <Description>Add To Wedding Registry</Description>
      <URL>https://www.amazon.com/gp/registry/wedding/add-item.html
       %3Fasin.0%3D0545670314%26SubscriptionId
       %3D[AWS Access Key ID]%26tag%3D[Associate ID]%26linkCode
       %3Dxm2%26camp%3D2025%26creative%3D386001%26creativeASIN
       %3D0545670314</URL>
    </ItemLink>
    <ItemLink>
      <Description>Add To Wishlist</Description>
      <URL>https://www.amazon.com/gp/registry/wishlist/add-
       item.html%3Fasin.0%3D0545670314%26SubscriptionId
       %3D[AWS Access Key ID]%26tag%3D[Associate ID]%26linkCode
       %3Dxm2%26camp%3D2025%26creative%3D386001%26creativeASIN
       %3D0545670314</URL>
    </ItemLink>
    <ItemLink>
      <Description>Tell A Friend</Description>
      <URL>https://www.amazon.com/gp/pdp/taf/0545670314
       %3FSubscriptionId%3D[AWS Access Key ID]%26tag
       %3D[Associate ID]%26linkCode%3Dxm2%26camp%3D2025%26creative
       %3D386001%26creativeASIN%3D0545670314</URL>
    </ItemLink>
    <ItemLink>
      <Description>All Customer Reviews</Description>
      <URL>https://www.amazon.com/review/product/0545670314
       %3FSubscriptionId%3D[AWS Access Key ID]%26tag
       %3D[Associate ID]%26linkCode%3Dxm2%26camp%3D2025%26creative
       %3D386001%26creativeASIN%3D0545670314</URL>
    </ItemLink>
  </ItemLinks>
  <ItemAttributes>
    <Author>Suzanne Collins</Author>
    <Manufacturer>Scholastic Press</Manufacturer>
    <ProductGroup>Book</ProductGroup>
    <Title>The Hunger Games Trilogy: The Hunger Games /
     Catching Fire / Mockingjay</Title>
  </ItemAttributes>
</Item>
```

Figure 2. The Typical Response

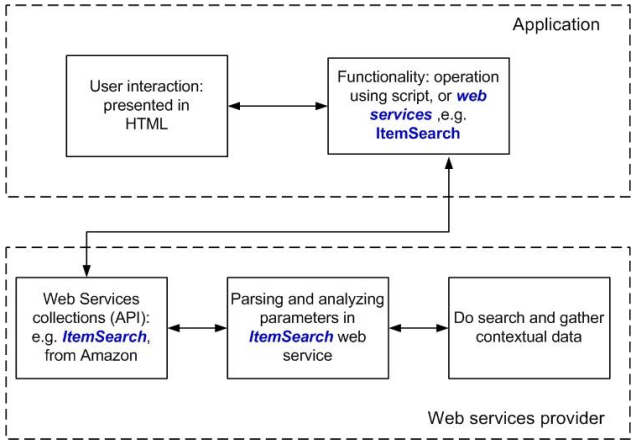

Figure 3. Interaction between Application and Web Services Provider

In order to assure a web service receives conflict free parameters, the application has to provide relevant parameters as requested. An application should do a proper design of user interaction and anticipation by avoiding possibility user selects irrelevant parameters to the entity domain.

In case of ***ItemSearch*** service, the service provides and accommodates various product categories. It means the service should be smart to identify parameters and connect to a relevant certain product category. The service also has to manage all databases for all product categories. This example, the ***ItemSearch*** independently provides a specific task, searching. But the service seems as a "fat web service", because, it has wide entity domains to handle. It manages many different entities domain data (e.g. movie, music, automotive, electronic, and so on). In fact, each entity domain has specific properties.

A microservice conducts a functionality task on the relevant data independently. Other property of a microservice is low coupling and high cohesion [3]. But ***ItemSearch*** might be low cohesion service. Because ***ItemSearch*** has to handle different and vary entity domains, it means should handle vary and wide databases.

Back to a structured design methodology, a conventional software design methodology is also known a structured design program or modular programming technique. A module can be categorized such as functional module, sequence, communication, and control. A module has properties do a single task (functionality) and work only on relevant data. It tends to have a high cohesion and low coupling, and is called as a functional module. So, it seems that a microservice has similar concept with the functional module. Whenever, a module consists of several tasks which each task may use data to communicate to each other, it called as communication module. And, module that do the task driven by control (tag, or switch), it called as control module. The control module, may consists of several tasks or several kinds of data. A selected task or data which is based-on input control.

Behavior task of ***ItemSearch*** depends on entity domain through on the inquiry parameters (e.g. ***SearchIndex***, and other option parameters) supplied in the application. Search functionality may vary and have to adapt, such as search behavior for automotive may different behavior if search for music, or movie, also in accesses database. The search functionality can be lighter if the search service focus only on one entity domain; task and data bind as a bundle (encapsulated). It is true, implication of this approach, the search service has to decompose, and has to generate many search services; each service dedicated to a specific entity domain.

Advantages of implement a single search service for one specific entity domain, it opens the opportunity the service can focus to adaptation of business changes. The service can evolve and continually developing, inherent and synchronize with business changes. It is one of the main objectives of the microservices. When an entity domain changes, the service easier to revise, and locally modifies.

There is challenge issues to tackle a problem with many search services, e.g. to keep managing robustness compare. A single service, even in wide entity domains, is still easier than the multi-search. An application that request a service, must submit the mandatory parameters and some additional (optional) information, e.g. identity, and timestamp as part of the service requesting. Therefore, it occurs additional information duplication.

One of the solutions is to apply the overriding and overloading concept; these are well known in object-orientation concepts. The other solution might be in using a template in object-orientation or specifically parameterized template, as generalization.

The ***ItemSearch*** service is one of the services part of an application construct. Actually, microservices and its architecture is on developer side issue. From the perspective of developing end-to-end application, a microservice architecture enables to support evolutionary and continuity application development, adaptation and synchronization with business changes [4]. For continuity development, there is classical issue, how to handle and manage a service version which represents business changes, is it only one newest version as current valid service, and discard previous versions?

V. NANOSERVICES: FINDING THE GENERALIZED

A web service is a microservice when the service does single task functionality for single entity domain. The solution for a web service which is not in single task functionality, a web service has to be converted into a microservice through decomposition. It decomposed into some single tasks, and each task binds to the related entity domain. Each microservice application manages each own database, and it is independent application. So, it may generate a number of microservices that all basically have same functionality. For example, ***ItemSearch*** (for movie, books, automotive, and others) behave according to the entity domains data. Then, it is possible there is a microservice on different applications has similar behavior (algorithm) but it is applied on different data. Therefore, there is a challenging issue need to be solved, the issue is,

how to avoid or reduce the duplication of the generated microservices.

We propose adopting some concepts or terminologies in object-orientation, may be used to consider for solving the duplication problem. The concepts are overriding, overloading, and template (generics). The overriding is a concept that each entity domain allows to redefine own behavior (algorithm), also still connected to ancestor (core behavior). The overriding concept associates with inheritance relation among classes, and overriding concept has relationship with polymorphism. In the overloading concept, it used when different data object will be processed by same algorithm. Both of overriding and overloading, still there a number of decomposed services, but their services name can use same name. In the overloading function, an algorithm works on different data (object) argument. And, template or parameterized template, can be used, to cover overloading function. Using the template concept, the application provides single algorithm (service) that can absorb and works on vary entity data domain as its argument.

The reason why our concern is reducing number of services having the same functionalities is to get robust services. An application which is developed in microservices architecture, it is independent application. The development style of microservices architecture is likely a bottom-up approach. Another approach is top-down, it usually used in the centralized system development method. In enterprises system perspective, an application is being part and member of the whole big applications, therefore for an enterprises system, it still requires communicating and exchanging data inter applications at least to a coordinator (pooler) application, and ensuring integrity. Some applications may have same functionalities, but handling on different data entity domain.

In conceptual model of microservices application. Let a microservice application $\boldsymbol{M}$, it consists two tuples are set of microservices $\boldsymbol{S}$, and set of data $\boldsymbol{D}$ that is used in microservice application $\boldsymbol{A}$. A set of data $\boldsymbol{d_i}$ used in microservice $\boldsymbol{s_i}$ is subset of $\boldsymbol{D}$. Between two subset data of two different microservices $\boldsymbol{d_i}$ and $\boldsymbol{d_j}$, may or may not have intersection of data used in $\boldsymbol{D}$; $\boldsymbol{d_i} \cap \boldsymbol{d_j} \neq \emptyset$, or $\boldsymbol{d_i} \cap \boldsymbol{d_j} = \emptyset$. Therefore, a matrix which represents relation between the set of services $\boldsymbol{S}$ and set of data $\boldsymbol{D}$, forms a sparse matrix, Eq. 1.

$$\begin{aligned} M &= \langle S, D \rangle \\ S &= s_1, s_2, ..., s_k \\ D &= d_1, d_2, ..., d_k \\ D &= \bigcup_{i=1}^{k} d_i, d_i \subseteq D \end{aligned} \tag{1}$$

In the enterprise system of microservices architecture, it consists of number of applications. Each microservice application are able independently to be developed continuous and autonomous, for always adapt to the business change. Each application has freedom to choose and implement database uses and algorithm uses for their services.

Let enterprise system $\boldsymbol{E}$ is pair of two tuples, a set of business process $\boldsymbol{B}$, and a set of microservices $\boldsymbol{A}$. The set of microservice application $\boldsymbol{A}$ consists of $\boldsymbol{p}$ microservices, which represents of the set $\boldsymbol{B}$. And $\boldsymbol{Q}$ is collection sets of microservices, subset of $\boldsymbol{A}$. Subset $\boldsymbol{q}^{\text{th}}$ is element of $\boldsymbol{Q}$, and a microservice application $\boldsymbol{A_q}$ is subset of $\boldsymbol{A}$. Therefore, a microservice application is set of microservices, subset of microservices on set $\boldsymbol{A}$. Each business process $\boldsymbol{b_i} \in \boldsymbol{B}$ is represented by set of application $\boldsymbol{A_i}$, $\boldsymbol{A_i} \subset \boldsymbol{A}$, $\boldsymbol{A_i} \in \boldsymbol{Q}$. The conceptual model of this system is defined in Eq. 2.

$$\begin{aligned} E &= \langle B, A \rangle, \\ & A = \{1, ..., p\}; Q = \{1, ..., m\}; \\ & 1 \leq q \leq m; q \in Q, \\ & A_q \subseteq A \\ B &= \{(b_i, A_i)\}, 1 \leq i \leq n\} \end{aligned} \tag{2}$$

A business process $\boldsymbol{b_i} \in$ B is implemented by a set of microservice applications $\boldsymbol{A_i}$, and the services which are used to construct of $\boldsymbol{A_i}$, is subset of $\boldsymbol{A}$. Then a microservice application $\boldsymbol{a_{i,k}} \in \boldsymbol{A_i}$, it consists of microservices in $\boldsymbol{A}$, $\boldsymbol{a_{i,k}} \subset \boldsymbol{A}$. So, in the enterprise system most probable, there is at least one microservice used in some microservice applications, the microservice takes form for reuse or may to extend.

For example, such as illustrate in Table I, each business process represented in a single microservice application. Each application focuses only on one single entity domain e.g. book, music, and apparel. This application has different characteristics represented by their properties. Therefore, in searching a certain product item they need search operation, and in this example named it as ***ItemSearch*** service, as illustrated in Fig. 4. These applications require search operation, and the behavior of these operations are differentiated by their properties depend on each entity domain. The row of the table represents list of properties, and the column represents search operations for each entity domain. The pivot represents relationship between properties and operations; it forms a sparse matrix. The search operation (e.g. named as ***ItemSearch***) is one of the other services in an end-to-end application for each entity domain (product line). It means search operation exists in many applications.

TABLE I. MATRIX OF RELATIONSHIP BETWEEN SEARCH OPERATION AND PROPERTIES DOMAIN

	ItemSearch: book	**ItemSearch: music**	**ItemSearch: apparel**
Keyword	Y	Y	Y
Price	Y	Y	Y
Size			Y
Dimension	Y		
Song list		Y	
Gender			Y
Singer		Y	
ISBN	Y		
Season			Y
Author	Y		
Publisher	Y	Y	

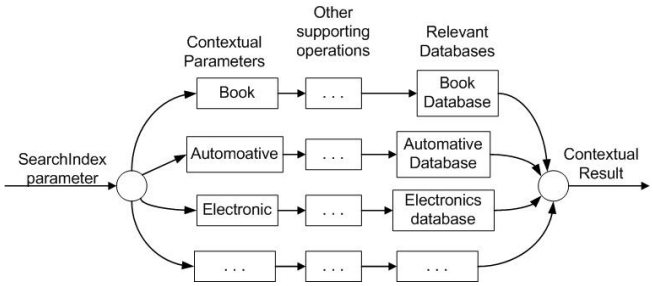

Figure 4. The BNF of ***ItemSearch***

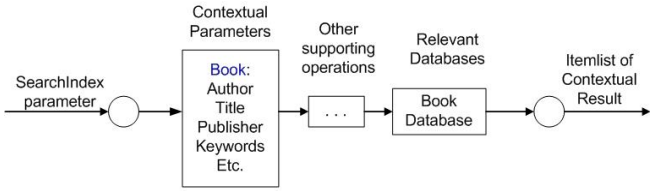

Figure 5. The BNF of ***ItemSearchBook*** Microservice

In the relationship on the Table I, there are some common characteristics, also have differences. The search operation on the different entity domains, they need some arguments (parameters) such as Keyword and Price. These properties play role as commonalities elements among them. Other properties make differentiation entity domains. Fig. 5 shows an example of ItemSearchBook.

Through example on the illustration has been discussed, finding commonality in some similar operations, is one of the challenging issues in web services architecture also in microservices architecture. Decomposing a web service imply of generating some microservices, and logically have relationship among these services. And, in this paper we called as nanoservices. If a big and complex applications have to develop adopt microservices architecture, will face many relationships among many similar services, it is a challenging issue need to solve; it beyond cover yet in this paper.

An enterprise system, handles many product lines, means manage many applications. As illustration in Table-1, even, in the implementation phase, each application may be developed independently as microservices architecture. But, in some extent, the applications still need to collaboration, and coordinate to the upper level. Therefore, in analysis and design phase, have to consider integration among applications.

For a small system, which handles single product entity domain, an application may develop independently, from analysis to implementation phase. It also, has flexibility space for evolving the application, to adapt and accommodate to business change; it is one of the points mentioned in SOA manifesto [2].

In enterprise system, it seems, use hybrid approach in software development phases, and may a good choice need to consider. Analysis and design phase, worked in centralized approach, and in implementation, operation, and maintenance, are worked in decentralized; it can apply microservices architecture.

VI. Conclusion and Further Works

Microservice architecture is a bottom-up approach in software development methodology. The microservice architecture is one of the choices strategy in implementation phase of application software development method. Microservice architecture provides for accommodating evolving and continuity application development to adapt to business environment changes. And, in nanoservices is a concept which provide to finding generalized services and relating among the similar services.

Finding nanoservices, is an exploration to find out core and common service from number of similar services. Developing model or mathematical model for representing and mapping all services exist within a system. Then, algorithms should be developed based-on this model. The algorithms will evaluate all services, and construct them as nanoservices.

Acknowledgment

This work is supported by Research Grant No. 0419/K3/KM/2017 of the Ministry of Research, Technology and Higher Education of the Republic of Indonesia, managed by LPPM of Pelita Harapan University, Research Contract No. 199/LPPM-UPH/VI/2017.

References

[1] P. Yugopuspito, F. Panduwinata, Sutrisno, and J. Pangaribuan. "Toward Service Oriented Design for Reservation-Based Parking," Proc. Int. Conf. on Information and Communication Technology Convergence (ICTC), Jeju (South Korea), Oct. 2016, pp. 201–205, doi: 10.1109/ICTC.2016.7763468.

[2] A. Arsanjani, et al., "SOA Manifesto." [Online] Available: http://www.soa-manifesto.org.

[3] N. Dragoni, S. Giallorenzo, A.L. Lafuente, M. Mazzara, F. Montessi, R. Mustafin, and L. Safina, "Microservices: Yesterday, Today, and Tomorrow," OSFHOME, Jun. 18, 2017, doi: 10.17605/OSF.IO/FNEUG.

[4] J. Lewis and M. Fowler, "Microservices," Mar. 25, 2014. [Online] Available: https://martinfowler.com/articles/microservices.html.

[5] H. Wang. M. Kessentini, and A. Ouni. "Prediction of Web Service Evolution," Proc. 14th Int. Conf. on Service-Oriented Computing (ICSOC), Banff (Canada), Oct. 2016, LNCS, vol. 9936, Q.Z. Sheng, E. Stroulia, S. Tata, and S. Bhiri, Eds., pp. 282–297, doi: 10.1007/978-3-319-46295-0_18.

[6] A. Balalaie, A. Heydarnoori and P. Jamshidi, "Migrating to Cloud-Native Architectures Using Microservices: An Experience Report," Proc. European Conf. on Service-Oriented and Cloud Computing (ESOCC) Workshop, Taormina (Italy), Sep. 2015, CCIS, vol. 567, A. Celesti and P. Leitner, Eds., pp. 201–215, doi: 10.1007/978-3-319-33313-7_15.

[7] M. Villamizar, O. Garcés, L. Ochoa, H. Castro, L. Salamanca, M, Verano, R. Casallas, S. Gil, C. Valencia, A. Zambrano, and M. Lang, "Infrastructure Cost Comparison of Running Web Applications in the Cloud Using AWS Lambda and Monoltithic and Microservice Architectures," Proc. 16th IEEE/ACM Int. Symp. on Cluster, Cloud and Grid (CCGrid), Cartagena (Colombia), May 2016, pp. 179–182, doi: 10.1109/CCGrid.2016.37.

[8] S. Ansems, "Microservices: Just Another Word for 'Tiny-SOA'?," TMNS, Jul. 2, 2015. [Online] Available: https://www.tmns.com/microservices-service-oriented-architecture/.

[9] E. Wolff, Microservices Primer: A Short Overview, CreateSpace Independent Publishing Platform, 2016. [Online] Available: http://microservices-book.com/primer.html.

[10] C. Fuller, “Serverless Architecture: Monoliths, Nanoservices, Microservices & Hybrids,” GorillaStack, Jun. 20, 2016. [Online] Available: https://blog.gorillastack.com/serverless-architectures-monoliths-nanoservices-microservices-hybrid/.

[11] R. Benefield, “How I Decided to Use Serverless/Nanoservices Architecture with AWS to Make CAPI,” LinkedIn, Jun. 1, 2016. [Online] Available: https://www.linkedin.com/pulse/how-i-decided-use-serverlessnanoservices-architecture-benefield.

[12] A. Rotem-Gal-Oz, “Services, Mircoservices, Nanoservice – Oh My!,” Mar. 25, 2014. [Online] Available: http://arnon.me/2014/03/services-microservices-nanoservices/.

[13] K. Channabasavaiah, K. Holley, and E.M. Tuggle. “Migrating to a Service-Oriented Architecture”, IBM White Paper, G224-7298-00, Apr. 2004.

[14] Amazon, “Product Advertising API: Developer Guide,” API version 2013-08-01. [Online] Available: http://docs.aws.amazon.com/AWSECommerceService/latest/DG/Welcome.html.

[15] B. Meyer, Object-Oriented Software Construction, 2nd ed., Prentice Hall, 1998, pp. 83–86.