Analysis of Component-Based Approaches Toward Componentized 5G

Elie El Hayek and Imen Grida Ben Yahia
Orange Labs Networks
Orange Telecom
Paris, France
{elie.elhayek, Imen.gridabenyahia@orange.com}

Damian Arellanes and Kung-Kiu Lau
School of Computer Science
The University of Manchester
Manchester M13 9PL, United Kingdom
{damian.arellanesmolina, kung-kiu.lau@manchester.ac.uk}

***Abstract*— 5G is expected to be modular by design toward autonomic and agile networks. In this regards, the 5G functional architecture is designed as service-based seeking to support the concept of Network Slicing. This leads us to the question: what componentization approach to implement this modular architecture? Is there a componentization approach that is suitable for all the network functions? Which design approach will help to have autonomic and cognitive networks?**

In this paper we shed the light on the different component-based approaches. In addition, we reviewed the state of the art addressing the applicability of component-based approaches to build autonomic networks. Therefore, we present discussion, comparison and synthesis to recommend which approach to be used to implement 5G modular architecture principle.

Keywords—5G; component-based; autonomic; microservice, object

I. INTRODUCTION

A core set of 5G use cases categories has been defined by several standardization and research projects: enhanced Mobile Broadband (eMBB), Massive Internet of Things (mIoT) and Critical Communications. In this context, the concept of Network Slicing is emerging. It considers the different needs of verticals through the creation of slices which is tailored to different functional and performance requirements. The integration of these new services and business actors requires a flexible architecture and smart (intelligent) management operations; the main driver to have a modular 5G architecture.

Component-based software development [1] can bring many benefits, including reduced time-to-market, reduced production cost, increased reuse, highly factored design, compositional construction, scalability, etc. Over the past years, researchers and industries have invented a number of component-based approaches. In every approach, there is an underlying component model to define what a component is and how components are composed.

This paper proposes an analysis of the different component-based approaches in order to highlight what could be suitable to implement a modular 5G architecture that is autonomic by design. It is organized as follows. Section II summarizes the different categories of components and highlights the application of component models to autonomic computing in the literature. In Section III we propose a comparison between different componentization paradigms based on selected criteria. A discussion around these approaches in the context of 5G is made based on the comparison results. Finally, Section IV summarizes the key conclusions of the paper.

II. RELATED WORK

We present in the following a twofold state of the art (sota). A sota on component types and a sota on autonomic architecture associated to component based design.

A. Components

In Lau et al. 2007 [2], a survey of major component models was conducted. From the survey, components are identified and classified into three main categories:

- Object-based,
- Architectural unit,
- Encapsulated component

Generically, a component is a unit of design with interface(s) specifying ports representing services it requires and crucially services it provides.

Object-based component

In this category, we have EJB (Enterprise Java Beans), COM (Component Object Model from Microsoft), OSGi [3] (Open Services Gateway initiative) frameworks. Within these frameworks, *a component is an object.* A provided service is a public method. Required services however are not explicitly specified in the sense that they are not in the interface of the object.

Using objects as components entails using object composition. Indeed, an object uses method delegation i.e. method calls to directly pass message to another object. Thus, objects compose by direct message passing.

Architecture unit

Koala, Acme, SOFA, and Fractal [4] are typical component models in this category. A component in this category is an architecture unit. Each component has explicit required and provided ports representing respective kinds of services.

Generally, all required services of a component have to be satisfied so that the component can be executed.
Since components are dependent on others (which can further depend on other components), using them tends to be more challenging.
Components in this category use indirect message passing in the form of port connection for connecting components.

Encapsulated component
In this category, we have Web services and X-MAN [5, 6, 7, 8, 9, 10]. An encapsulated component as the name suggests has encapsulation of functionality and data. It does not require external services for its provided services.
An encapsulated component only has provided services. Encapsulated components have no external dependencies on one another. They do not call one another. Services, e.g. web services and Microservices [11, 12], may call one another directly and so may have external dependencies on one another. Therefore services may or may not be encapsulated components.
Encapsulated components are composed by coordinators. This kind of composition is called exogenous composition. A coordinator coordinates the control flow between the components, and manages the results returned by the components.
In X-MAN, coordination is performed by (exogenous) composition connectors which embody control structures.

Besides these three categories of components, **Microservices** units are widely spread and investigated in telecom networks. It is an independently replaceable, upgradeable and deployable unit. It is small and focuses on completing a single task that represents a small business capability. A Microservice is designed for failure i.e. if a Microservice fails, other Microservices involved in the same application continue to run and the failed one can be re-instantiated if needed.
Microservices communicate directly by calling REST APIs over HTTP. Microservices need a client library for every service they communicate directly with. Maintaining libraries is costly. Furthermore, an HTTP connection may become a bottleneck (especially for long running Microservices) since it must be open during the entire communication. Microservices can also use a lightweight messaging bus, known as API Gateway, to communicate indirectly. In particular, Microservices are required to expose their endpoints to the API Gateway.

B. Autonomic computing and component-based approaches

In this section, we review the state of the art where autonomic computing is applied to component-based approaches. This short survey shows that the field is not active and that the topic is not a cornerstone in the context of autonomic networking and computing.
Self-configuration is the system-level property that requires the reconfiguration of an autonomic computing system by installing, updating, integrating, uninstalling, replicating and reconnecting components at runtime [13, 14]. These actions are known as reconfiguration operations. Self-configuration describes what reconfiguration operation to perform under which conditions, and is driven by high-level policies. Reconfiguration operations can be either architectural (i.e., the addition, removal or the replacement of software components and/or connectors) or parametric (i.e., modifications to the parameters of components and connectors) [15].
Some of the techniques to reconfigure software components at runtime include dynamic linking [16], dynamic object technology (including class loaders), dynamic programming languages, design patterns [16] and architectural reflection. An autonomic engine must be responsible of minimizing the disrupting of the operation by shutting down (part of) the software system, then performing the needed reconfiguration operations. This process is known as quiescence.

Self-adaptive component models have been proposed to accommodate changes in the operating environment of a component-based software system, by allowing dynamic reconfiguration operations of software components. Authors in [17] describe K-component, a self-adaptive component model that defines self-adaptive components for distributed computing systems, and provide an Adaptation Contract Description Language (ACDL) for the specification of the adaptation logic.

Likewise, authors in [18] use a procedure based on the concept of Automated Planning (an artificial intelligence area) to generate a reconfiguration plan (i.e. a sequence of reconfiguration actions) at runtime, thus allowing a dynamic reconfiguration where only the parts (components) that must be adjusted are affected, rather than the whole application.

On the other hand, the authors in [19] present SATIN, a model that supports reconfiguration by offering code migration services (logical mobility). The SATIN component model uses logical mobility primitives to provide distribution of services. Instead of relying on the invocation of remote services via the network, the component model supports the cloning and migration of components between hosts, providing autonomy to the system when network connectivity is missing or unreliable.

In [20], a self-adaptive component model called DEECo is presented. It defines components as independent and self-sustained units of development, deployment, and computation. DEECo components are made up of four major parts: knowledge, beliefs, interfaces and component processes. The knowledge part reflects the internal state and the available functionality of the component. Beliefs are copies of knowledge of other components; this part is treated with a certain level of uncertainty as it might become obsolete or invalid. A component interface is used to expose the component's knowledge so that it represents a partial view on the component's knowledge. Component processes are essentially soft real-time tasks that manipulate the knowledge of a component, whose operation is cyclic scheduled by a runtime framework.

III. Comparison of Componentization Paradigms

To the best of our knowledge, the sota lacks comparison of the component based approaches and its suitability to networking design in particular.

In order to help researchers and standardization deciding the right approach we describe in this section a comparison between five componentization approaches: X-MAN, Bundles, Fractal, SOA [21] and Microservices. For the effectiveness of the comparison, we defined a set of criteria which we will be using in Table 1.

A. Comparison results

Hereafter, we will define a set of criteria for the component-based approaches.
We will be comparing the nature of the approach by means of: component model, object-oriented framework or distributed services.

Each component model defines: (i) **software units**, (ii) **composition types** and (iii) **composition mechanisms**.
The software units could be *encapsulated components*, *objects*, *services*, *microservices* etc. We also examine if the software units have external dependencies or not i.e. whether their computation requires services from other units or not.

Moreover, the composition type is also compared. Algebraic composition means that it defines composite components such that they have the same type as their sub-components or not; an algebraic composition defines hierarchical composition. Algebraic is related to the semantics of the composition (e.g. if we have a set of architectural units components. If the composition of these components is algebraic, the result of the composition is also algebraic). Hierarchical composition is enabled by algebraic semantics.
As far as a composition operator is used, this type of composition can be defined (and implemented) as a mathematical operator for example.

In addition, we compared the composition mechanism that is used by the approach. These mechanisms could be control coordination (composition connectors, orchestration) or message passing (direct or indirect):

- Orchestration (e.g. in Service Oriented Architecture) specifies the execution order of selected operations in services, so the assembly of services is a workflow and not a composite service. Indeed, the orchestration engine itself defines a sequence of invocations. As such, the orchestration is done by a separate server.
  On the other hand, X-MAN uses composition connectors as control coordination. These composition connectors compose components into composite components.
- In indirect message passing the interactions between components is mediated by a messaging bus while in direct message passing the interactions between components are made with no mediator between them.

Table 1 shows a comparison of the aforementioned componentization approaches.

B. Discussion on components & autonomics for future networks

An autonomic system [13, 22] is composed of an autonomic manager (control loop e.g. MAPE – Monitor Analyze Plan Execute) and a managed element. If we consider the componentization facets, we can imagine the four following views:

- The managed element is not implemented with the componentization paradigms, and the autonomic manager is implemented using a componentization paradigm (view#1).
- The managed element is implemented using componentization paradigms, then two options are possible, the autonomic control loop could be implemented also as a component (view#2) or not (view#3).
- We could also have the manager, managed element developed as one integrated component (view#4).

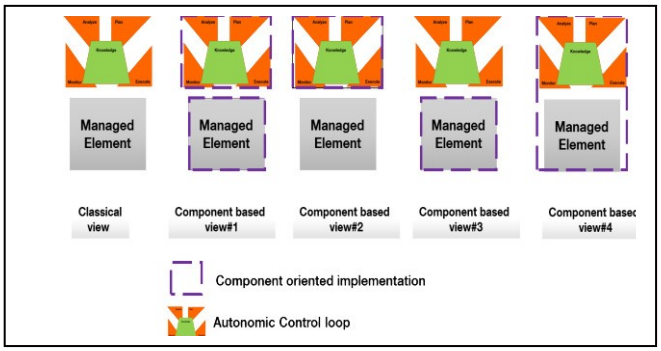

Fig. 1. Potential views of an autonomic component

The previous four views (Figure 1) are not exhaustive and are purely theoretical. Componentization paradigms could be indeed beneficial to building autonomic systems. However, which paradigm from Table 1 can we recommend for example? And on which basis can we make this choice?

For more than 20 years, autonomic based systems have adopted several designs that sometimes were imposed by constraints issued from the managed element itself as well as the MAPE functions. Usually wrappers and envelopes were developed, however, as these mechanisms were not of general purpose, they were not extensible and as a consequence their integration was time consuming and costly.

For future networks such as 5G, where the network will be based on SDN, NFV and Cloud principles, do we need a reference design approach for such heterogeneous network elements (IoT – Internet of Things – paradigm) and for management functions? Is it a feasible approach?

Reference models and design approaches including componentization paradigms, as well as frameworks such as RM-ODP (Reference Model of Open Distributed Processing)

and Zachman, were popular in the 80's and 90's and less nowadays with open-source communities and de facto standards.

Since 1968 and for more than 40 years, several componentization paradigms have appeared, the latest ones are more and more "lightweight" to ease their adoption. This also proves that no reference component design approach was sufficient and suitable to any software development.
In the telco case, we will surely need a component type for a given network function e.g. a virtual HSS (vHSS), another type of component for a fault management operation, and a third type for an SDN controller. This heterogeneity of componentization types will lead to costly integration of all these types.
In these regards, for 5G networks and their management, if we adopt the componentization paradigms, we will end up with several and heterogeneous types of components to cover the different network and management requirements, which will bring us to the initial problem of how to manage heterogeneous devices, resources, infrastructure in 5G networks?

TABLE I. COMPARISON BETWEEN COMPONENTIZATION APPROACHES

	Componentization model	***X-MAN***	***Bundles***	***Fractal***	***SOA***	***Microservices***
Approach	**Approach nature**	Component model	Object oriented framework	Component model	Component model	Distributed services
Software units	**Units**	Encapsulated components	Objects	Architectural units	Services	Micro services
	External dependencies	No	Yes	Yes	Yes	Yes
Composition types	**Algebraic**	Yes	No	Yes	No	No
	Hierarchical	Yes	No	Yes	Yes	Yes
	Through operators	Yes	No	No	No	No
Composition mechanisms	**Control coordination (composition connectors)**	Yes	No	No	No	No
	Control coordination (type workflow)	No	No	No	Yes	Yes
	Direct message passing (method calls)	No	Yes	No	Yes	Yes
	Indirect message passing (port connectors)	No	No	Yes	No	No

So what are the key drivers if the heterogeneity of resources remains?

Componentization paradigm will be a good practice for a precise software development and not for general purpose ones, like for example 5G management functions (involving autonomic principles) that are different in nature and roles in the context of 5G (verticals, IoT, diverse devices, etc.). Unifying the network using only one component approach is not possible. Each approach/paradigm may be optimized to be suitable for one use case but not another.

From Table 1, we can observe the extreme differences between the componentization paradigms: X-MAN approach is covering all the theoretical properties of a component model while Microservices are a lightweight approach design. Thus, Microservices approach is becoming very common in current implementations while we can find that in theory X-MAN is the best approach, but for adoption there is a need for software libraries, for strong community and for extensive tooling.

At the same time, the differences in Table 1 are being ameliorated by combining the strengths of componentization paradigms. For instance, there is a tendency of using control coordination for Microservices [23, 24] whilst X-MAN has evolved into a distributed service-oriented model [25, 26].

IV. Conclusion

This paper discusses component-based approaches as a key design principle to implement a flexible and modular 5G architecture capable of answering to the diversity of 5G use cases. The application of autonomic to the componentization paradigms is also presented.

The article shows that componentization approaches are suitable if a certain degree of flexibility is allowed i.e. the approach is not rigid or with too many formal constraints (e.g. Microservices). This also applies to autonomics where the most important aspects are the programmable interfaces so that a given autonomic manager and its managed element can communicate smoothly. Moreover, we argue that componentization paradigms are not suitable for unification; each approach may be suitable for one use case but not for another.

This analysis is theoretical but fundamental as the sota lacks such study. We aim in a future work to push the barriers of this study through prototyping.

References

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

[2] K.K. Lau and Z. Wang. “Software component models”, IEEE Transactions on Software Engineering 33, 10, pp. 709-724, 2007

[3] OSGi Alliance Specification,“The Dynamic Module System for Java”. https://www.osgi.org/

[4] T. Coupaye, J.B. Stefani, “Fractal Component-Based Software Engineering Report on the WS Fractal at ECOOP’06”, 5th ECOOP Workshop on Fractal, France, 2006

[5] K.-K. Lau, P. Velasco Elizondo, Z. Wang, “Exogenous Connectors for Software Components”. Proc. 8th Int. Symp. on Component-based Software Engineering, LNCS 3489:90—106, 2005

[6] K.-K. Lau, M. Ornaghi, Z. Wang, “A Software Component Model and Its Preliminary Formalisation”. Proc. 4th Int. Symp. on Formal Methods for Components and Objects, LNCS 4111:1—21, 2006

[7] P. Velasco Elizondo, K.-K. Lau, “A Catalogue of Component Connectors to Support Development with Reuse”. The Journal of Systems and Software, 83:1165—1178, 2010

[8] N. He, D. Kroening, T. Wahl, K.-K. Lau, F. Taweel, C. Tran, P. Rümmer, S. Sharma, “Component-based Design and Verification in X-MAN”. Proc. Embedded Real Time Software and Systems, 2012

[9] K.-K. Lau, C. Tran, “X-MAN: An MDE Tool for Component-Based System Development”. Proc. 38th EUROMICRO Conference on Software Engineering and Advanced Applications:158-165, 2012

[10] S. Di Cola, C. Tran, K.-K. Lau, “A Graphical Tool for Model-Driven Development Using Components and Services”. Proceedings of 41st Euromicro Conference on Software Engineering and Advanced Applications (SEAA) 2015:181—182, 2015

[11] S. Newman, “Building Microservices”, 1st Ed., O’ Reilly Media, 2015

[12] M. Fowler, J. Lewis, “Microservices: A definition of this new architectural term”, http://martinfowler.com/articles/microservices.html, 2014

[13] J. O. Kephart, D. M. Chess, “The vision of autonomic computing”. Computer, 36(1):41–50, 2003

[14] M. Salehie, L. Tahvildari, “Self-adaptive Software: Landscape and Research Challenges”. ACM Trans. Auton. Adapt. Syst., 4(2):14:1–14:42, 2009

[15] P. Oreizy, M.M Gorlick, R.N. Taylor, D. Heimbigner, G. Johnson, M. Medvidovic, A. Quilici, D.S. Rosenblum, A.L. Wolf, “An Architecture-Based Approach to Self-Adaptive Software”, IEEE Intelligent Systems, 14(3):54-62, 1999

[16] J. Dowling, T. Schäfer, V. Cahill, P. Haraszti, B. Redmond, “Using Reflection to Support Dynamic Adaptation of System Software: A Case Study Driven Evaluation”, In Cazzola61, W., Stroud62, R. J., and Tisato63, F., editors, Reflection and Software Engineering, number 1826 in Lecture Notes in Computer Science46, pages 169–188. Springer Berlin Heidelberg, 1999

[17] J. Dowling, V. Cahill, “Self-managed Decentralised Systems Using K-components and Collaborative Reinforcement Learning”, In Proceedings of the 1st ACM SIGSOFT Workshop on Self-managed Systems, WOSS ’04, pages 39–43, New York, NY, USA. ACM, 2004

[18] M. Eugênio, M. Di Beneditto, C. Maria, L. Werner, “Using a model to generate reconfiguration plans at runtime”, In Proceedings of the 14th ACM conference on CBSE, Marcq-en-Baroeul, France, 2014. ACM

[19] S. Zachariadis, C. Mascolo, W. Emmerich, (2004). “satin: A Component Model for Mobile Self Organisation”, In Meersman81, R. and Tari82, Z., editors, On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE, number 3291 in Lecture Notes in Computer Science62, pages 1303–1321. Springer Berlin Heidelberg

[20] T. Bures, I. Gerostathopoulos, P. Hnetynka, J. Keznikl, M. Kit, and F. Plasil, “DEECO: An Ensemble-based Component System”, In Proceedings of the 16th International ACM Sigsoft Symposium on Component-based Software Engineering, CBSE ’13, pages 81–90, New York, NY, USA. ACM, 2013

[21] M. P. Papazoglou, P. Traverso, S. Dustdar and F. Leymann, “Service-Oriented Computing: State of the Art and Research Challenges”, Computer, Vol. 40 No. 11, IEEE Computer Society, pp. 38-45, November 2007

[22] S. Dobson, S. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, F. Zambonelli, "A survey of autonomic communications"- ACM Transactions on Autonomous and Adaptive Systems 1(2), December 2006

[23] Netflix. Conductor, https://netflix.github.io/conductor/, 2016

[24] R. Oberhauser, “Microflows: Automated Planning and Enactment of Dynamic Workflows Comprising Semantically-Annotated Microservices”, In: Shishkov B. (eds) Business Modeling and Software Design. BMSD 2016. Lecture Notes in Business Information Processing, vol 275. Springer, Cham, 2017

[25] D. Arellanes and K.-K. Lau, “DX-MAN: A Platform for Total Compositionality in Service-Oriented Architectures”, in Proceedings of the 7th International Symposium on Cloud and Service Computing (SC2 2017), IEEE Computer Society, 2017

[26] D. Arellanes and K.-K. Lau, “Exogenous Connectors for Hierarchical Service Composition”, in Proceedings of the 10th International Conference on Service Oriented Computing and Applications (SOCA 2017), IEEE Computer Society, 2017