Micro-service Architecture for Emerging Telecom Applications

Manfred Sneps-Sneppe, Dmitry Namiot

***Abstract*—In this paper, we would like to discuss software standards in the connection with the development of emerging telecom applications. Emerging (military) telecom applications present one of the biggest challenges for the developers (design and development) and telecom providers (deployment and maintenance). It is especially true in the context of transition from TDM to IP networks. This transition may bring own challenges associated with the priorities in data transmission, security, etc. We describe two biggest telecom related examples (GIG and FI-WARE), discuss the challenges and propose directions for software standards development and deployment.**

***Keywords*— circuit switch; packet switch; communications; software standards; micro-service; middleware.**

I. Introduction

In this paper, we would like to discuss the software standards in the connections to the development of emerging telecommunication services.

We want to dwell on the importance of the opinions and the emerging trends in the development during the creation of new standards. We are talking about the standards that affect software development. By our opinion, the very important point here is an adaptation (adoption) of standards by the existing development community ensures their wide distribution and use [1]. Otherwise (which is not uncommon), we are faced with a situation where standards exist in parallel and independently of the established practice. In recent history, we can recall examples of real opposition to the proposed standard from the existed approaches and practices. For example, we can point out here the confrontation of TCP/IP protocol stack and the ISO, Corba and Web services, IIOP and XML, and on the same Web Services versus REST, XML versus JSON, etc [2].

In the telecommunications world, the most interesting example is, of course, the whole story behind the Parlay. We saw a whole family of APIs: Parlay/OSA, Parlay X, which can be described as a simplified version of Parlay/OSA, then JAIN. This constant redesigning and repositioning of standards leads to a loss of meaning as to that constitutes a standard [3]. Parlay is also a great example of incompatibility for standards implementation.

We can use many parameters describing the software standards. But from our point of view, the main (determining) parameter is the answer to the main question of interest to developers. This question is the time it takes to build services (applications) using a novel approach. Time is a key factor in software development. Can we save a time with new approach? The biggest problem with above-mentioned Parlay was the conclusion about time-to-market for new development with this approach. Actually, this standard increased the time for development (time to market for new services).

By our opinion, the key point for any software development tool is the simplicity and finally, time to market for new applications. Note, that telecom projects could be heavy affected by standard problems due to high diversity in the devices and use case models. Not taking into account the interests of developers to create standards, we risk facing a parallel existence of the standard model and the actually utilized the existing approaches.

In the connection with software standards we discuss the biggest telecom-related software projects: Global Information Grid (in Section II) and Future Internet FI-WARE (in Section III). Both are too sophisticated to be successful to follow unique standards, of course, from our particular viewpoint. As a compromise we offer micro-service approach (in Section IV).

II. Modeling the Global Information Grid

The movement from circuit switching to packet switching is one of the biggest tasks for telecom companies over the world (Figure 1). The packet switching equipment manufacturers are the main engine behind this movement. And they are the first promoters of this change of the paradigm of the telecommunications industry.

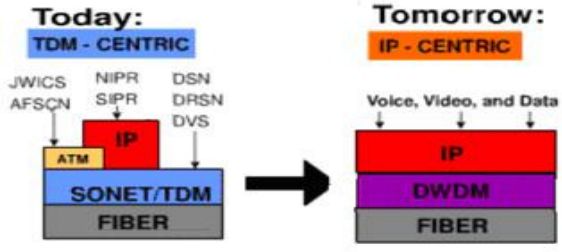

Fig. 1. TDM to IP.

In the paper, we consider the difficulties of the transition from circuit-switched to packet switched communication networks on the example of the Ministry of Defense of the United States - the world's largest private network. We hope that the experience of such a large project could help domestic operators who took the orientation to "All-over-

Manuscript received October 3, 2014.

M. Sneps-Sneppe is with Ventspils University College, Ventspils International Radioastronomy Centre, Ventspils, Latvia. e-mail: manfreds.sneps@gmail.com

D. Namiot is with Lomonosov Moscow State University, Faculty of Computational Mathematics and Cybernetics, Moscow, Russia. e-mail: dnamiot@gmail.com

IP". In 2008, significant effort began on Global Information Grid (GIG), led by Vice Admiral Nancy Brown, Joint Staff, Director for C4 Systems [4].

The key goal is to reduce barriers to information sharing. Adm. Brown pointed out that there are too many networks and GIG 2.0 should be a framework to bring together service intranets to act as one global network (Figure 2).

Fig. 2. GIG 2.0 framework

GIG 2.0 has five characteristics that the admiral outlined in her presentation: global authentication, access control and directory services; unity of command; information and services "from the edge"; joint infrastructure; and common policies and standards. The tactical edge is in the center of the framework because GIG 2.0 is being developed to provide capability to the troops on the ground. Fig. 2. GIG 2.0 is a framework to bring together many networks to act as one global network Figure 3 illustrates the co-operation tasks between different forces [5].

The DISA (Defense Information Systems Agency) systems engineering process shown in Figure 4 was developed to ensure DISA services and applications. These solutions will be developed using a Model based Systems Engineering (MBSE) methodology in conjunction with the standards-based Systems Modeling Language (SysML), which focuses on the underlying data in the models.

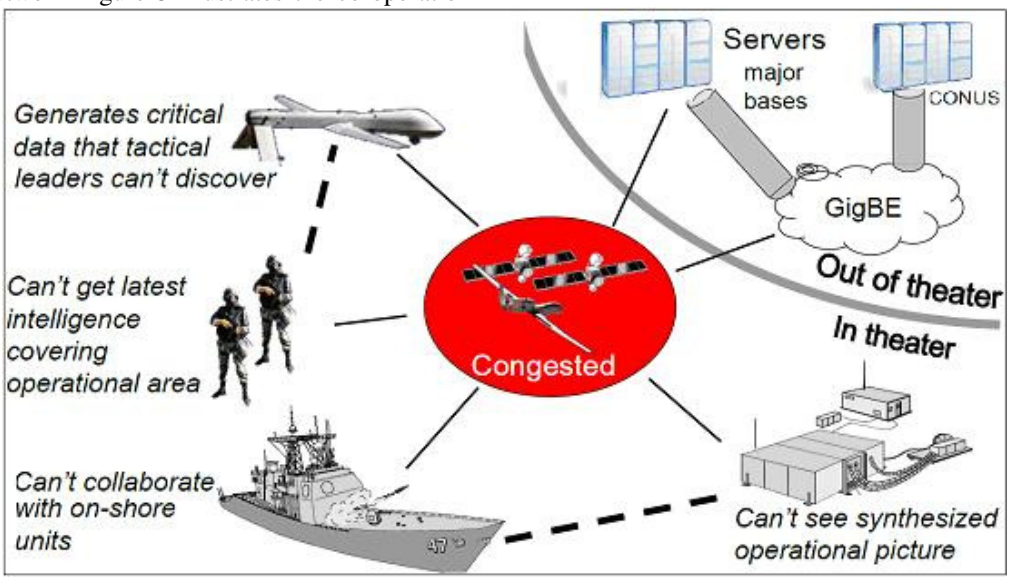

Fig. 3. Co-operation tasks

International Council on Systems Engineering (INCOSE) defined the Unified Modeling Language (UML) for Systems Engineering strategy in January 2001. INCOSE partnered with the Object Management Group (OMG) published Systems Modeling Language (SysML) specification in June 2006 [6].

Figure 4 depicts the DISA MBSE process, where the system architecting process maps to the SysML diagrams that comprise the model. Models provide precise descriptions of how systems work and include well defined interfaces, which make it possible to combine existing models into end-to-end services; the models that make up these end-to-end services can then be used as patterns to develop new services.

SysML provides 9 different types of diagrams to represent the architecture, which can be used to develop solutions: 4 behavioral, 4 Structural and one Cross-Cutting diagram. These 9 SysML diagram types map directly to DoD Architecture Framework [7] (DoDAF) models, totally 26 DoDAF models. 26 DoDAF matrix artifacts are reports that can be generated directly from SysML models.

The comparison of enterprise frameworks (EAF) is presented in [8]. Authors provide a methodology for the comparison. They chose several aspects of EAF. The planner view includes the concepts for the final product. It may encompass items such as the relative size, shape, and basic intent of the final structure. The owner view is that of the owner which may represent an architect's drawings in which the owner agrees that the architect has captured what he has in mind. The designer view is the architect's final product. The builder view represents the view in which the architect's final plans are modified to reflect how construction will proceed. The subcontractor view represents drawings of parts or subsections of the plans. The product

view represents the final product, building, or project. In their study they have mentioned, for example, the lack of maintenance phase support in DoDAF. On the implementation phase DoDAF just describes the final product. There are no implementation tools.

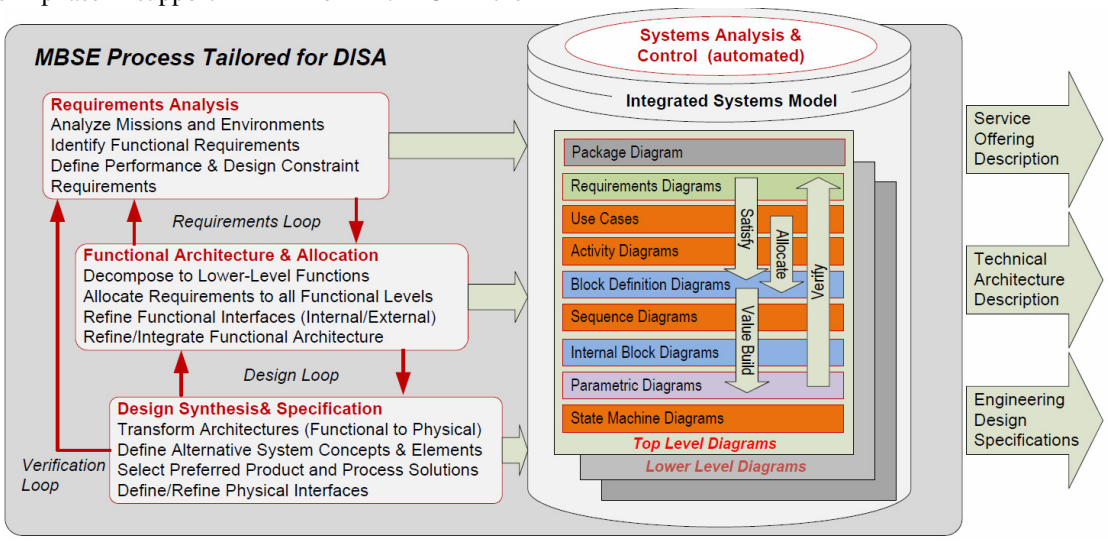

Fig. 4. DISA Model

Thus, this is an extremely complex design, which will require the work of many thousands of programmers and is unlikely to be brought to an end, as there are new tasks, such as problem of cyberwar, which have already led to a revision of GIG: GIG 3.0. These considerations were the basis for treatment Micro-service architecture discussed below.

III FI-WARE PROJECT

Military applications are similar to M2M communications or, from the wider viewpoint, similar to Internet of Things. The most interesting in the area of Future Internet, from the developer's point of view, is FI-WARE project [9]. FI-WARE will deliver a novel service infrastructure, building upon elements (called Generic Enablers) which offer reusable and commonly shared functions, making it easier to develop Future Internet Applications in multiple sectors – building a true foundation for the Future Internet.

The project will develop public and royalty-free Open Specifications of Generic Enablers, together with a reference implementation of them available for testing. This way, it is aimed to develop working specifications that influence Future Internet standards. FI-WARE is the cornerstone of the Future Internet Public Private Partnership (PPP) Program, a joint action by the European Industry and the European Commission.

The FI-PPP follows an industry-driven, user-oriented approach that combines R&D on network and communication technologies, devices, software, service and media technologies; and their experimentation and validation in real application contexts. The platform technologies will be used and validated by many actors, in particular by small- and medium-sized companies and public administrations. FI-WARE architecture is shown in Figure 5.

There are more than 60 FI-WARE Generic Enablers (GE) as common building blocks across Use Case projects, and more than 100 Specific Enablers as dedicated building blocks coming from the Use Case projects so as to support their proof of concept and build prototypes. Each enabler presents a set of components and some unified API (Application Program Interface). The specifications for enablers are open.

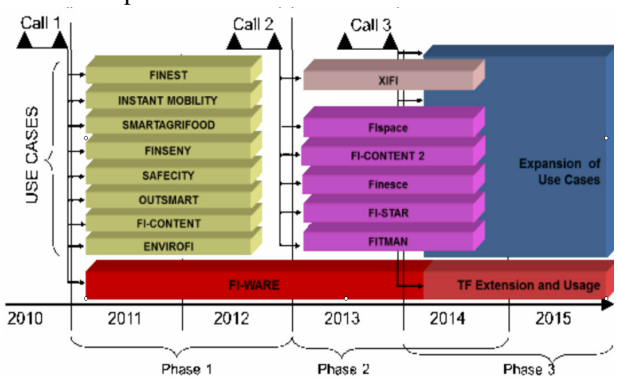

Fig. 5. FI_WARE project

As per official document, FI-WARE will enable smarter, more customized/personalized and context-aware applications and services by the means of a set of assets able to gather, exchange, process and analyze massive data in a fast and efficient way [10]. In general, FI-WARE contains the following chapters:

cloud hosting,
application/service delivery framework,
data/context management,
Internet of Things enablement,
interfaces to network devices, and
security.

It is mentioned for example, that the Apps Generic Enablers supports (should support) managing services in a business framework across the whole service life cycle from creation and composition of services to monetization and revenue sharing (see above the remarks about maintainability in DoDAF).

Data in FI-WARE refers to information that is produced, generated, collected or observed that may be relevant for processing, carrying out further analysis and knowledge extraction. A basic concept in FI-WARE is that data elements are not bound to a specific format representation.

FI-WARE proposes also an interesting approach for Applications/Services Ecosystem and Delivery Framework. It is based on the heavy usage on USDL [11]. Universal Service-Semantics Description Language (USDL) can be used by service developers to specify the formal semantics of web-services. Thus, if WSDL can be regarded as a language for formally specifying the syntax of web services, USDL can be regarded as a language for formally specifying their semantics. USDL is as formal service documentation that will allow sophisticated conceptual modeling and searching of available web-services, automated composition, and other forms of automated service integration. For example, the WSDL syntax and USDL semantics of web services can be published in a directory which applications can access to automatically discover services.

FI-WARE approach is modular and it is very close to the micro-services architecture. The whole success for the platform depends on the above mentioned enablers. E.g. 17 Specific Enablers relate to the OUTSMART project (Smart City project) but only a few are implemented by now.

The biggest problem, by our opinion, is “all or nothing” approach with FI-WARE based way. E.g., FI-WARE spec de-facto sets mandatory data sharing to some cloud environment. Do we really need it for all imaginable scenarios? There are many use case for sensors (tags) inspection from end-user devices (e.g. smartphones or other personal devices). Mandatory data sharing just adds the complexity [12-13].

IV The micro-service architecture

The micro-service approach is a relatively new term in software architecture patterns. The micro-service architecture is an approach to developing an application as a set of small independent services. Each of the services is running in its own independent process. Services can communicate with some lightweight mechanisms (usually it is something around HTTP) [14]. Such services could be deployed absolutely independently. Also, the centralized management of these services is a completely separate service too. It may be written in different programming languages, use own data models, etc.

An opposite approach is so-called monolithic architecture. Internally, the monolithic application may have several services, components, etc. But it is deployed as a united solution. For its scalability we can run several copies of this application, but they are identical. What are the advantages?

Unless the application is getting too big, it is easier to develop. No doubt, it is easier to deploy the monolithic application. It is, probably, the biggest advantage of the monolithic solution.

The path for the scalability is also clear. We can run multiple copies of the application behind a load balancer. But this approach has got the serious drawbacks too.

The monolithic application could be difficult to understand and modify. It is especially true, when the application is growing. With the growing application it is difficult to add new developers, or replace leaving team members.

The large code base slows the productivity. Very often we it will lead to the declined quality of the code. The original modularity will be eroded. The monolithic application prevents the developers from working independently. The whole team must coordinate all development and redeployments efforts [15].

It makes the continuous development very difficult. The monolithic application makes the obstacles to the frequent updates. In order to update some small component, we have to redeploy the whole application.

Scaling the application can be actual difficult too. But there is another reason. A monolithic architecture can only scale in one dimension. We can increase transaction volume by running more copies of the application. But on the other hand, this architecture can not scale with an increasing data volume. Each our copy of application instance will access all of the data. It makes caching less effective. Also, this solution increases memory consumption and input/output traffic. At the same time, different application components may have different resource requirements. One might be CPU intensive while another might be memory intensive. With a monolithic architecture, we can not scale each component independently.

The next biggest issue is a technology stack. With the monolithic architecture, it is very difficult to change it. E.g., there is almost no way to change development framework, etc. It can be difficult to incrementally adopt a newer technology. And all components within the application will be sticking to technology being selected at the beginning.

Micro-services architecture gets our attention in the connection with M2M applications. We declare many times, that in our opinion “no one size fits all” in M2M applications. So, we think that the unified (monolithic) framework for M2M (IoT) is not a realistic solution. By this reason we think that micro-services are the natural fit for M2M (IoT) and hence development. As an example, we can mention our paper [16].

V Discussion on programming methodology

Micro-services architecture may cause some changes in the used programming paradigms. Let us name some of the modern approaches in this connection. We think that some of them could be the true future for telecom programming.

Reactive programming (functional reactive programming - FRP) [17] is a paradigm for programming hybrid systems (systems containing a combination of both continuous and discrete components) in a high-level, declarative way. The key ideas in FRP are its notions of continuous, time-varying values, and time-ordered sequences of discrete events. The most important concept underlying functional reactive

programming is that of a signal: a continuous, time-varying value. That is, a value of type Signal is a function mapping suitable value of time to a value of a given type.

The next interesting concept is Abstract Task Graph [18]. The Abstract Task Graph (ATaG) is a data driven programming model for end-to-end application development of networked sensor systems. An ATaG program is a system-level, architecture-independent specification of the application functionality. ATaG model maps the network graph to an application graph.

ATaG provides a methodology for architecture-independent development of networked sensing applications. Architecture independence here is the ability to specify application behavior for a generic and parameterized network architecture. The same application may be automatically adopted for the different network deployments. Application will work as nodes fail or are added to the system. Furthermore, it allows development of the application to proceed prior to decisions being made about the final configuration of the nodes and network.

As the next model we would like to mention in this context is the Computational REST [19]. In this model the traditional content resources are replaced with computational resources. The key moments behind the Computational REST are:

- Computations and their expressions are explicitly named.
- Services may be exposed through a variety of URLs which offer perspectives on the same computation.
- Interfaces may offer complementary supervisory functionality such as debugging or management.
- Functions may be added to or removed from the binding environment over time or their semantics may change.
- Computations may be stateful and stateless.
- Potentially autonomous computations exchange and maintain state.
- A rich set of stateful relationships exist among a set of distinct URLs.
- The computation is transparent and can be inspected, routed, and cached.
- The migration of the computation to be physically closer to the data store is supported thereby reducing the impact of network latency.

In this context we should mention also an interesting model CoReWeb [20]. It presents a web of linked computational resources.

And at the end, we will describe Flow-Based Programming (FBR) [21] and the Actor Model [22]. Both models are based on components where the messages are the only entities which can affect processes. FBR is actually very close to the extensions of M2M API proposed in our paper [19]. Also Actors are very close to the basic primitives for micro-services.

REFERENCES

[1] Sneps-Sneppe, M., and Namiot, D. (2012, April). About M2M standards and their possible extensions. In Future Internet Communications (BCFIC), 2012 2nd Baltic Congress on (pp. 187-193). IEEE. doi: 10.1109/BCFIC.2012.6218001

[2] Namiot, D., & Sneps-Sneppe, M. (2014, June). On software standards for smart cities: API or DPI. In ITU Kaleidoscope Academic Conference: Living in a converged world-Impossible without standards?, Proceedings of the 2014 (pp. 169-174). IEEE.

[3] Hawkins, R., and Ballon, P. (2007). When standards become business models: reinterpreting “failure” in the standardization paradigm. Info, 9(5), pp.20-30.

[4] The Global Information Grid (GIG) 2.0 Concept of Operations Version 1.1//11 March 2009, Joint Staff J6, Washington, D.C.

[5] John Chapin Reengineering the GIG to Support the Warfighter// IEEE COMSOC, 8 January 2009.

[6] SyncML Initiative. "SyncML Architecture, version 0.2." (2001).

[7] DoD Architecture Framework Working Group et al. DoD architecture framework version 1.0 //Department of Defense. – 2003.

[8] Urbaczewski, L., & Mrdalj, S. (2006). A comparison of enterprise architecture frameworks. Issues in Information Systems, 7(2), pp.18-23.

[9] Tuominen, L. (2013). Future Internet in the European Union-Case FI-WARE.

[10] Future Internet Public Private Partnership. Outcomes, achievements, and outlook. Phase 1, Final Report, European Commission, Sept 2013.

[11] Barros, Alistair, and Daniel Oberle. Handbook of Service Description: USDL and Its Methods. Springer Publishing Company, Incorporated, 2012.

[12] Sneps-Sneppe, Manfred, and Dmitry Namiot. "M2M Applications and Open API: What Could Be Next?." Internet of Things, Smart Spaces, and Next Generation Networking. Springer Berlin Heidelberg, 2012. pp. 429-439.

[13] Namiot, Dmitry, and Manfred Sneps-Sneppe. "On M2M Software." International Journal of Open Information Technologies 2.6 (2014): 29-36.

[14] Uckelmann, Dieter, Mark Harrison, and Florian Michahelles. "An architectural approach towards the future internet of things." Architecting the internet of things. Springer Berlin Heidelberg, 2011. pp.1-24.

[15] Namiot, D., & Sneps-Sneppe, M. (2014). On Micro-services Architecture. International Journal of Open Information Technologies, 2(9), 24-27.

[16] Namiot, Dmitry, and Manfred Sneps-Sneppe. "On M2M Software Platforms." International Journal of Open Information Technologies 2.8 (2014): pp. 29-33

[17] Hudak, P., Courtney, A., Nilsson, H., & Peterson, J. (2003). Arrows, robots, and functional reactive programming. In Advanced Functional Programming (pp. 159-187). Springer Berlin Heidelberg.

[18] Bakshi, A., Prasanna, V. K., Reich, J., & Larner, D. (2005, June). The abstract task graph: a methodology for architecture-independent programming of networked sensor systems. In Proceedings of the 2005 workshop on End-to-end, sense-and-respond systems, applications and services (pp. 19-24). USENIX Association.

[19] Erenkrantz, J. R. (2009). Computational REST: A New Model for Decentralized, Internet-Scale Applications DISSERTATION (Doctoral dissertation, University of California, Irvine).

[20] Monnin, A., Delaforge, N., & Gandon, F. (2012, June). CoReWeb: From linked documentary resources to linked computational resources. In Proceedings of the WWW2012 Conference Workshop PhiloWeb 2012: “Web and Philosophy, Why and What For”.

[21] Morrison, J. P. (1994). Flow-based programming. In Proc. 1st International Workshop on Software Engineering for Parallel and Distributed Systems (pp. 25-29).

[22] Esposito, A., & Loia, V. (2000). Integrating concurrency control and distributed data into workflow frameworks: an actor model perspective. In Systems, Man, and Cybernetics, 2000 IEEE International Conference on (Vol. 3, pp. 2110-2114). IEEE.