Software Defined Networking: Bridging the gap between distributed-systems and networked-systems research

Paul Mueller

Integrated Communication Systems Lab. (ICSY)
University Kaiserslautern
Paul Ehrlich Street Bld.34
67663 Kaiserslautern
pmueller@informatik.uni-kl.de

Abstract: This contribution takes a quote from Einstein, *We can't solve problems by using the same kind of thinking we used when we created them*, to take a step back and try to have another viewpoint when thinking about a new internetworking architecture. The ongoing research will not be looked at from an *application research* point of view nor from a *communication technology* point of view, but from a *software engineering* perspective. This new way of thinking leads us to realize that the Internet is a *largely distributed software system*, and thus we propose to apply software-oriented methodologies to define a new internetworking architecture. These methodologies lead to a new, flexible architecture that allows for both evolutionary and clean-slate approaches. Moreover, this approach bridges the gap between *application* and *distributed systems* research, and *networking* research to have a more holistic view and avoid suboptimal solutions.

1 Introduction

As described in the first report prepared by the FIArchitecture group [FIA11], the current Internet is challenged by ever new demands of constantly emerging applications and new capabilities of communication networks. Today these challenges result in an architectural patchwork with increasing complexity and unpredictable vulnerabilities. This patchwork of the formerly clearly layered architecture, however is not related to specific protocols or mechanisms, but is mainly caused by the (more than 40 years old) architecture of the Internet.

Under these circumstances all adaptations of the current Internet architecture (e.g., security, middle boxes) result in unintentional complexity that finally leads to an ossification of the Internet core with increased development (for applications) and maintenance (for networks) costs (e.g., security/privacy issues, distributed denial-of-service attack attacks, etc.). This means that these shortcomings cannot only be solved by an evolutionary step-by-step approach, but they also require efforts in newly designed internetworking architectures [Cl03], which should be more adaptable and flexible. Overall, a new internetworking architecture must offer flexible mechanisms to map the demands of (yet unknown, future) applications onto the capabilities of (yet unknown, future) transport networks. Especially the current uniform API of the internet has come to be a handicap to distributed applications which cannot exploit many of the useful

capabilities of the underlying network, and therefore must perform their own measurements to adapt to varying network conditions, often with cross-layer approaches (see figure 1).

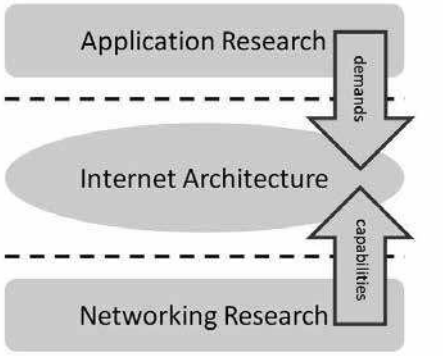

Fig. 1. The Internet Architecture is between applications and networking research (adopted from [Ro06])

Today there are sharp boundaries between *distributed systems* and *applications* research, and *networking* research; which is evident in institutions (IETF vs. OASIS) and companies (Google vs. Cisco). To overcome this situation it is essential to take advantage of the combination and improvements in the fields of distributed systems research and networking research [Ro06]. This has been discussed in many of the projects and new initiatives within the 6th and 7th EC Framework Programs, "Networks of the Future". A short overview of related approaches can be found in [MR08]. Future network architectures must be *flexible* in both the long and short term in order to evolve and adapt to changing application requirements and new transport technologies (fixed and/or mobile) with different capabilities, by enabling evolutionary changes of the network itself. *Long-term* flexibility can be seen as the capability of a system to evolve with updated protocols and network capabilities. *Short-term* flexibility is understood as the capability of a system to adapt itself and react to network conditions and application requirements [Kh12a].

In that sense, a new internetworking architecture should be able to be adapted to different transport technologies so that simple, low-cost systems can be integrated into the network just as well as high-performance end systems. From this, it follows that network functionality should be highly configurable so that it can be adapted to several transport technologies. Moreover, a new internetworking architecture should be able to handle the different requirements of various applications. Since application requirements are usually not known until run time, the network must be able to react dynamically to handle these requirements.

In addition to such external demands, there are demands for flexibility from the network itself. We assume that there is no fixed set of mechanisms that is well suited to fulfill all requirements on all transport technologies for all time. Therefore, capabilities for evolutionary changes are required in a new internetworking architecture to avoid radical, and thus costly, changes. The nodes of the Internet today are already heterogeneous, but they all have the functionality of TCP/IP in common. Relatively few improvements of these protocols haven been introduced in the past (compared to the many suggested improvements) and more radical changes like migrating from IPv4 to IPv6 are ongoing now for more than 15 years. From this we derive the requirement that there should be very few mechanisms — ideally none — that are mandatory for all nodes in the network.

One of the drawbacks of the current Internet architecture is tight coupling of functionalities (within a layer and cross layer) that make it difficult to integrate new functionalities. For example, expanding the IPv4 address space by increasing address lengths requires modifying the TCP implementation. Also the UDP checksum incorporates parts of the IP header, tightly coupling UDP and IP. Likewise, making other changes can encounter similar problems. We can mitigate maintenance costs by exploiting the long-term flexibility of a flexible internetworking architecture by, for instance, upgrading a protocol to fix a security issue.

Nevertheless, a more flexible internetworking architecture also has to deal with increasing, but inherent, complexity resulting in increasing overhead costs. "The greater the flexibility required, the higher the complexity of the code (i.e., overhead cost) needed to support that flexibility" [Mc04]. This inherent complexity and its overhead costs (i.e., processing of the increased complexity and code length) is countered by reduced network maintenance costs and by ever-increasing hardware capabilities. On the other hand, increasing flexibility decreases maintenance cost, as "flexibility increases insight into the systems secrets causing general maintenance cost to decrease. Finding system bottlenecks and optimizing performance will also be much easier once the systems architecture is flexible" [Ec04]. As hardware advances will mitigate overhead costs, it is worth investing the overhead costs of increased flexibility in order to reduce long-term maintenance costs. In contrast, the maintenance costs for inflexible network architecture will increase: "As the maintenance backlog of the inflexible software system increases, cost compound, development is deferred, customer dissatisfaction grows, and competitive position declines" [Jo05].

According to the above assumption, we propose a new and open internetworking architecture based on the principles of service-oriented architectures (SOA) [Oa06] by defining open, standardized, and generic service interfaces which make it possible to decouple logic from implementation**.** This approach prepares for the complexity of future applications by simplifying integration of new technologies and increasing communication system flexibility. Such a flexible architecture may also offer new *business models*, as any certain functionality is not related to a fixed layer, but can be integrated and offered as a new service when needed. Of course there are serious methodological problems associated with researching new internetworking architectures. In particular, it is hard to claim success in this area without building a successful follow up to the current Internet.

2 Service Oriented Network Architecture

As mentioned above, we consider the Internet as a largely distributed software system facing similar problems as software development processes, which has developed several concepts (e.g., maintenance, integration of new functionality, time and task management) to manage the complexities of the development process. Applying these concepts directly affects the cost, quality, and development time of software. The Internet has similar kinds of problems that are not addressed by the current design principle(s). To deal with the inflexibility and complexity issues of the Internet, the principles and techniques from software development can be learned and implemented.

Software architecture has advanced from structured programming to a service-oriented paradigm based on SOA1. The design of a future internetworking architecture can benefit from software architecture paradigms to make a more flexible and easy to maintain network architecture rather than having an ossified architecture (i.e., the current Internet). The following sections present an argument on how SOA can be a suitable methodology for a future internetworking architecture. Before arguing about why service-orientation could be a promising paradigm for a future internetworking architecture, the fundamental principles [Er06] of SOA are described:

- ***Service contract:*** The explicit specification of the provided service. It avoids defining (implicitly) the delivered service by algorithms/protocols or even code, fostering loose coupling.
- ***Loose coupling:*** Refers to the degree of dependency and bounding between two components. The coupling of two services is considered loose when they are independent of the implementation of one another. Loose coupling simplifies the exchange of service implementations and the introduction of new services. This supports long-term flexibility.
- ***Abstraction:*** Services are independent in that the logic they use is hidden from the outside world. Abstraction fosters reusability as well as loose coupling.
- ***Reusability:*** A service should be independent and fine-grained enough to promote reusability. Networks often have to support a wide range of applications, which requires reusing network functionality.
- ***Autonomy:*** Characterized by the control of a service over the logic it encapsulates. Autonomous services have fewer requirements on their environment and can be deployed more easily, thus fostering system evolution.
- ***Statelessness:*** Service should not keep the state of a request after it has been processed. This principle cannot be fulfilled in general, since some protocols have to be stateful.
- ***Discoverability:*** A service should be descriptive enough for automatic service discovery. This applies to services residing within a network infrastructure. These kinds of services should be addressed dynamically so that various instances of a service may be used.
- ***Composability:*** The ability of a service to be coordinated with other services in a manner that they can form a composite service. Composability is the precondition to avoiding the necessity of implementing large complex communication by reusing several more fine-grained services instead.

Most of the issues in the Internet arise because of inflexibility and rigidness of the network architecture, which is built upon a protocol stack. SOA provides a new perspective for building a future internetworking architecture as it addresses service loose coupling, re-usability, and autonomy; which are fundamental requirements of a flexible architecture. As the protocol stack can be decomposed into various functionalities, described with formal contracts (i.e., service descriptions), the functionalities become autonomous and self-descriptive. Self-descriptive functionalities have the ability to be discovered, as they carry descriptions that can be processed by discovering entities. Abstraction is another key point to be taken into account.

Decomposing the protocol stack into various functionalities should be done at an appropriate abstraction level, where implementation details are hidden from the consumer. Other characteristics of a functionality, such as autonomy, description and re-usability, are important for the composition of higher level services. Nevertheless, the statelessness principle of SOA might not be appropriate for all functionalities of a network architecture, as some network functionalities require state (e.g., reliable transmission). In the following we describe what services are and show their composition in protocol graphs.

2.1 Communication Services

Services are the essential elements of a SOA. A service reflects the effects of an activity rather than the algorithms and data structures that implement it. Thus a service could be implemented using different algorithms. It is necessary that there are explicit *service* descriptions that include the service semantics and interface definitions [Kh10a]. A building block is the implementation of an atomic service. A building block could implement a protocol such as retransmission, encryption, or monitoring. Each building block usually has several effects, such as increasing end-to-end delay or reducing maximum payload size, in addition to its main function. All the effects of a building block represent the service it provides. The interfaces of a building block should reflect the provided service while hiding its implementation details. Building blocks should also use *generic interfaces* [Li11] so that interaction between building blocks do not require adapters. In order to fulfill the SOA principles, it is crucial to design services and building blocks appropriately, keeping in mind the principles of SOA.

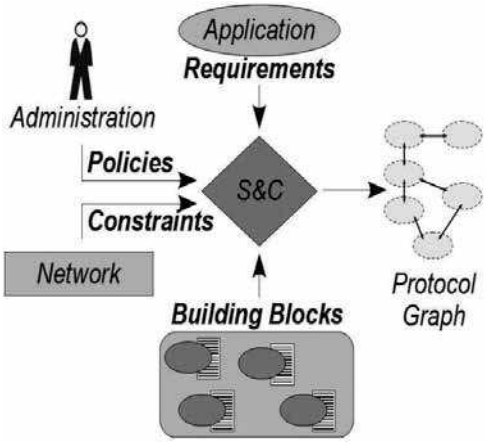

Fig. 2. Parameters for the definition/generation of protocol-graphs

2.2 Protocol Graphs

In order to make use of flexible networks as mentioned above, it is necessary to define protocol graphs and to select a communication service to be used.

When a suitable protocol graph is not already defined, a new one can be created by determining which building blocks should be used and how these building blocks have to interact in order to provide an appropriate communication service. This selection and composition of building blocks (see figure 2) uses the service descriptions of the available building blocks to define an optimal protocol graph with regard to application

requirements, network constraints, and administrative policies. Several approaches for selection and composition are possible, ranging from manually defined protocol graphs to automatic composition at runtime [Si11]. A simulation model which evaluates the impact of network communication services in order to decide if a block or a set of blocks composed into a protocol graph is able to fulfill the given application requirements and system constraints is given in [Gu12a].

It is likely that flexible networks will offer several similar communication services to applications, each of which could be implemented with different protocols. In such a scenario, applications must not be aware of the utilized protocols. To achieve this, applications must only be aware of the provided service, keeping the service implementation transparent to applications. This can be achieved by introducing a service broker, which selects an appropriate service implementation at runtime (see figure 3). These implementations can range from the conventional protocol stack (TCP/IP), to pre-composed or template-based protocol graphs, to a completely dynamic setup that composes a suitable protocol graph on demand. A service is appropriate if it fulfills all mandatory application requirements. In addition, optional application requirements are used to determine the optimal service. A service broker might consider services provided by different sources. There may be standard protocol stacks, pre-composed services as well as dynamically composed services. This way, a service broker also enables the simultaneous use of concurrent selection and composition approaches.

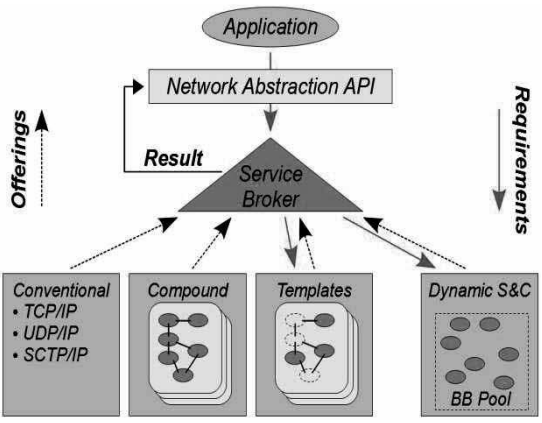

Fig. 3. Selecting an appropriate communication service at runtime

As mentioned above, flexible protocol graphs are made up of building blocks. Each block implements certain network functionality like encryption, loss reduction, routing, or compression. From a pool of available functional blocks, appropriate blocks are selected by looking at certain criteria, including application requirements, and then connected via their ports [Sc11] to create a requirement and network specific protocol graph. To reduce the naturally occurring complexity in creating an entire protocol graph, we use a template-based functional composition (TFC) approach.

The idea of this TFC approach is to split the functional composition process among different time-phases (i.e., design-time, deployment-time, and run-time) so that the time consuming activity is performed at the least time-critical phases (e.g., design-time). Potentially less time consuming activities are performed at run-time. The most time

consuming activities are the selection of functionalities, but not the actual functional blocks, and connecting them together in appropriate order so that they can interact with each other. To utilize the less time critical phases and still provide enough flexibility, TFC utilizes placeholders to represent the functional blocks that will be selected at run-time. TFC may produce more than one protocol graph that fulfills the application's requirements. Selecting which of these protocol graphs to use is a Multi- Criteria Decision Analysis (MCDA) problem that is solved by using an adaptation of the Analytic Hierarchy Process (AHP) [Kh06]. A description language is used to describe the application requirements and functional block capabilities. These descriptions are necessary for the selection and composition process. A detailed description of the proposed service description language can be found in [Kh12b]. Moreover, for demonstration purposes, an interface, GAPI (G-Lab Application-to-Network Interface [Li11]), was created to allow applications to specify their requirements.

2.3 Service Description

Service selection and composition requires the description of communication services. The task of service selection is to select a suitable or the best service from a set of candidate services. Service composition takes a set of application requirements as input and composes a set of fine-grained services considering constraints from the user, an administrator, and from the network. Service selection is a prerequisite of service composition.

Selection and composition take the requirements from the application, a set of policies from the administrator, constraints from the network, and a set of effects provided by the building blocks. Then it composes the selected building blocks, considering all of the inputs, and produces a protocol graph as output. As is seen in the figure 2, all of the inputs and outputs are nothing but descriptions. Having a service description of the composed protocol graph facilitates the process of service selection as shown in figure 3. A service provider might provide services of the conventional protocol stacks (i.e., services provided by TCP/IP, UDP/IP, and SCTP/IP).

A compound service can be seen as a service composed during design time. A compound service provider will provide design time composed services. Another service provider can supply a template during design time and the rest of the composition can be done during run-time. The dynamic selection and composition uses the requirements and other inputs at runtime (where the most information about the communication is available) to compose services.

A communication service description language must be able to describe fine-grained functionality (i.e., the capacities or capabilities of a service), constraints from the network, policies from the administrator, and application requirements. Moreover, the output of the selection and composition (S&C) engine is a protocol graph that should be described using the same language. The selection of a suitable, or the best, service requires the description of requirements and offerings, as shown in figure 2, which should also be described without changing the language. All of these requirements, constraints and offerings require specification of effects, influence, interfaces, data types

and dependencies. Effects and interfaces are required to hide internal implementation mechanism from an application or a user and show only the parts that are required during selection and composition. Whether a building block or a service influences a header of a packet, the payload of a packet, or the flow of data needs to be known during service selection and composition. During service composition, the compatibility of the connections between interfaces is checked by using data types. An interface can only accept a connection from the building block X if the building block offers data of a particular type. Dependencies are required to assist the functional composition process.

3 Can SOA based network architectures provide flexibility?

Using services as the basic elements for the design of a system instead of algorithms or protocols fosters loose coupling and abstraction. Service descriptions represent the service contracts and are also used to discover services. Building blocks should be largely independent of their environment to achieve autonomy. Generic building block interfaces make composition much easier. The layerless architecture implies higher probability for reuse of functionality. Statelessness cannot be achieved in general because some functionality can only be implemented using state-full protocols. In addition, there may be generic states, for example, in the connection setup or release phase or states for failure or debug modes.

A new internetworking architecture should be flexible in two ways. First, networks should be able to adapt to specific customer or application needs and changing environmental conditions. Second, networks should be able to evolve, meaning that functionality is added, changed, or even removed. This flexibility is achieved by composing several (smaller) services into more complex and specialized services.

Today's networks organize complex protocols in layers, building a nearly static protocol graph [OP92]. Service oriented network architectures aim to support dynamic composition

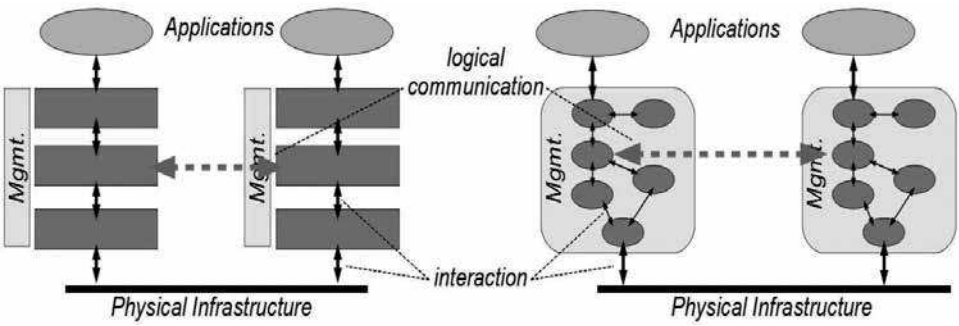

Fig. 4: Layer vs. Service Oriented Paradigm

of services (i.e., dynamic protocol graphs). When dynamic protocol graphs are used, it is easier to make use of new protocols (i.e., building blocks) and to reuse functionality on different levels. Having dynamic protocol graphs implies that there is no static

placement of functionality as defined by the layers of the OSI reference model (see Figure 4). In this sense such networks will be *layerless* such as compression or encryption can be used for application payload only or also for some protocol headers. Furthermore it is not necessary that building blocks be processed in sequence. For example, there might be different branches in the protocol graph to handle different, but related, data types within one flow (e.g., signaling and streaming media). In order to enable dynamic protocol graphs the interaction between building blocks is not defined by executable code but by description that can be changed easily. Dynamic adaptation (i.e., dynamical adaption to requirements and constraints) can be achieved by service composition and service selection that supports short term-flexibility. A short description of a first prototype of this approach based on the SONATE (service-oriented network architecture) framework [Kh10a] can be found in [Gu12b]. A first method for generating requirement-specific protocol graphs can be found in [Gu12c].

4 Conclusions

Because of the inherent inflexibility (ossification) in the current Internet architecture, it is hard to integrate new functionalities required from the application- or network-site. This paper argues that short-term and long-term flexibility is the key for designing new internetworking architectures. To achieve such a required flexibility it is necessary to overcome the barriers between “distributed-systems” and “networked-system” research based on the ideas of software defined networking. Taking this into account, our approach for a new and flexible internetworking architecture is based on a new software engineering methodology namely the service-oriented architecture principles.

Such an approach of fine-grained functionality can enable more flexible networks, so that the network can adapt to new application's needs and new network capabilities and/or constraints. This also allows the introduction, alteration, and removal of functionalities. Moreover, as any certain functionality is not related to a fixed layer but can be integrated and offered as a new user oriented, application oriented, or network oriented service when needed (potentially by a third party), this approach may also offer new business models.

While this paper gives a design foundation for future network architectures there are still several open issues remain for applying the SOA paradigm in network architectures. One open issue e.g. is service granularity. Because SOA does not describe the granularity of a service design guidelines are required for the extent of logic every service must contain. Another open question is service composition. Approaches for selection and composition face a tradeoff between ’composition time’ and ’information availability’. On the one hand, at design time, there are nearly no time limitations for the composition process and requirements of application(s) are already known. On the other hand, at runtime there are hard time constraints for selection and composition, but most of the specific user requirements network constraints might be available. A further challenging issue is how to handle different types of network nodes, to simultaneously make use of their provided services, and to ensure a conflict-free service composition at the same time. The challenge here is to develop mechanisms or protocols to handle heterogeneous services offered by different devices on a communication path.

Although the presented approach does not offer a short-term product oriented approach, it offers a long-term research methodology to foster research and innovation in this area.

References

[Cl03] Clark, D.: New arch: Future generation internet architecture, Final Technical Report. [Online]. Available: http://www.isi.eu/newarch/

[Ec04] Eckstein, J.; Marchesi, M.; G. Succi, G.; Baumeister, H.: Extreme programming and agile processes in software engineering, LNCS 3092, 2004.

[Er06] Erl, T.: Service-oriented architecture, 2006.

[FIA11] EC FIArch Group, Fundamental limitations of current internet and the path to future internet, Draft Ver.: 0.9. Available: http://www.future-internet.eu/uploads/media/FIArch_Current_Internet_Limitations_March_2011__FINAL_.pdf

[Gu12a] Günther, D.; Kerr, N.; Mueller, P.: A Simulation Model for Evaluating the Impact of Communication Services in a Functional-block-based Network Protocol Graph, 15th Communications and Networking Symposium (CNS'12) at SpringSim'12, 2012.

[Gu12b] Günther, D.; Schwerdel, D.; Siddiqui, A.; Khondoker, M.R.; Reuther, B.; Mueller, P.: Selecting and Composing Requirement Aware Protocol Graphs with SONATE. In: 12th Wurzburg Workshop on IP: ITG Workshop "Visions of Future Generation Networks" (EuroView2012), 2012.

[Gu12c] Günther, D.; Kerr, N.; Mueller, P.: A Method for Producing Requirement-Specific Protocol Graphs in a Flexible Network Architecture, in Mathematical and Computer Modelling, 2012.

[Jo05] Johnson, B.; Woolfolk, W.; Miller, R.; Johnson, C.: Flexible software design - systems development for changing requirements, Book, 2005.

[Kh10a] Khondoker, M.R.; Reuther, B.; Schwerdel, D.; Siddiqui, A.; Mueller, P.: Describing and Selecting Communication Services in a Service Oriented Network Architecture, ITU-T Kaleidoscope, Pune, India, 13-15 Dec 2010.

[Kh10b] Khondoker, M.R.; Reuther, B.; Schwerdel, D.; Siddiqui, a.; Mueller, P.: Describing and selecting communication services in a service oriented network architecture, In: Proceedings of the 2010 ITU-T Kaleidoscope: Beyond the Internet? - Innovations for Future Networks and Services, Pune, India, 2010.

[Kh12a] Khondoker, M.R.; Siddiqui, A.; Reuther, B.; Mueller, P.: Service Orientation Paradigm in Future Network Architectures, Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS-2012), 2012.

[Kh12b] Khondoker, M.R.; Veith, E.M.; Mueller, P.: A Description Language for Communication Services of Future Network Architectures, Network of Future (NoF), Paris, France,

2012.

[Li11] Liers, F.; Volkert, T.; Martin, D.; Backhaus, H.; Wippel, H.; E. Veith, E.M.; Siddiqui, A.; Khondoker, M.R.: Gapi: A G-Lab application-to-network interface, In Proceedings of the 11th Wuerzburg Workshop on IP: Joint ITG and Euro-NF Workshop on Visions of Future Generation Networks (EuroView 2011), 2011.

[Mc04] McConnell, S.: Code complete: A practical handbook of software construction, Book, 2004.

[MR08] Mueller, P.; Reuther, B.: Future Internet Architecture - A Service Oriented Approach, In: it – Information Technology, Jahrgang 50 (2008) Heft 6, S. 383-389 6/2008.

[Oa06] OASIS: Oasis reference model for service oriented architecture 1.0, Official OASIS Standard, 2006.

[OP92] O'Malley, S.W.; Peterson, L.L.: A dynamic network architecture, ACM Transactions on Computer Systems, vol. 10, pp. 110–143, 1992.

[Ro06] Roscoe, T.: The end of Internet architecture, In: Proceedings of the fifth workshop on hot topics in networks (HotNets-V), Irvine, USA, November 2006.

[Sc11] Schwerdel, D.; Günther, D.; Khondoker, M.R.; Mueller, P.: A building block interaction model for flexible future internet architectures, In: 7th Euro-NF Conference on Next Generation Internet, Kaiserslautern, Germany, 2011.

[Si11] Siddiqui, A.; Khondoker, M.R.; Reuther, B.; Henke, C.; Backhaus, H.; Mueller, P.: Functional Composition and its Challenges, Proceedings of The first international workshop on Future Internet and Next Generation Networks (FINGNET 2011), Seoul, Korea, 2011.