Future Internet Architecture – A Service Oriented Approach

Future Internet Architecture – Ein serviceorientierter Ansatz

Paul Mueller, Bernd Reuther, University of Kaiserslautern

Summary The development of the Internet during the last years has shown that it becomes harder to integrate new functionality in order to fulfill the demands of new applications and the capabilities of new transport technologies. Especially the core mechanisms (TCP/IP) are hard to change. Thus the development of a new Internet architecture has been discussed for some time now. In this work we present a service-oriented approach for a new network architecture. At first it is argued which kinds of flexibility should be provided by an architecture. Then it will be shown which common principles of service-orientation support this flexibility. Further we present technical considerations for implementing network functionality by services. ►►► **Zusammenfassung** Die Entwicklung des Internets in den letzten Jahren hat gezeigt, dass es immer schwieriger wird, neue Funktionalität zu integrieren, um Anforderungen seitens der Applikationen zu erfüllen und auf veränderte Rahmenbedingungen der Transporttechnologien zu reagieren. Insbesondere die Kernmechanismen (TCP/IP) sind nur schwer veränderbar. Daher wird seit einiger Zeit die Entwicklung einer neuen Architektur für ein zukünftiges Internet diskutiert. In dieser Arbeit stellen wir einen serviceorientierten Ansatz für eine neue Netzwerkarchitektur vor. Zuerst wird begründet, welche Arten der Flexibilität eine neue Architektur mindestens aufweisen sollte. Anschließend wird gezeigt, welche Grundprinzipien serviceorientierter Architekturen diese Flexibilität unterstützen können. Des Weiteren werden grundlegende Techniken für eine Realisierung von Netzwerkfunktionalität durch Services vorgestellt.

KEYWORDS C2.1 [Computer Systems Organization: Computer-Communication Networks: Network Architecture and Design] network communications; C2.2 [Computer Systems Organization: Computer-Communication Networks: Network Protocols]

1 Introduction

Driven by the demands of ever emerging applications and the capabilities of new communication networks, the Internet has become an architectural patchwork resulting in increasing complexity and unpredictable vulnerabilities. This patchwork is the result of layer violations (e. g., cross-layer design), sub-layer proliferation (e. g., MPLS at layer 2.5, IPsec at layer 3.5, and TLS at layer 4.5), and erosion of the end-to-end model (middle-boxes, such as firewalls, NATs, proxies, caches, etc.) – see Fig. 1. This erosion of the formerly clearly layered architecture is not just because of too much functionality but because of lots of implicit dependencies, i. e., tight coupling. Under these circumstances, all changes result in a rise of complexity which finally lead to an ossified Internet.

The problems mentioned above are not related to specific protocols or mechanisms of the current Internet but are mainly caused by the inability to integrate new mechanisms. These problems are caused by the architecture of the Internet and thus could be solved by a newly designed architecture [1]. In the context of this paper we define an architecture according to [2] as the fundamental organization of a system, the relationship of components as well as the design and evolution principles (see [3] for further discussion).

Our approach for a new and open network architecture is based on principles of service-oriented architectures (SOA) [25]. By defining open, standardized and generic service interfaces, it is possible to decouple logic from implementation. This enables a simplified in-

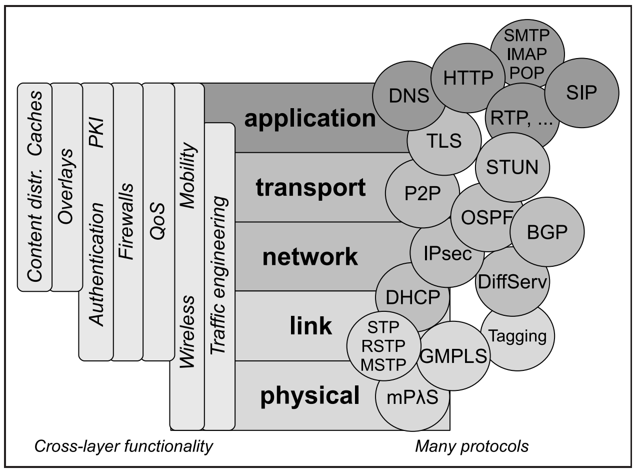

Figure 1 The Internet Patchwork.

tegration of new technologies and increases flexibility of communication systems in order to be prepared for future complex applications.

The next section provides a brief overview of related work. Then we argue that a next generation network should be flexible in several ways. Section 4 describes how basic concepts found in service-oriented architectures can meet these demands. In Section 5 we present technical considerations for implementing a framework based on service-oriented architecture principles. The paper concludes with a summary and an outlook.

2 Related Work

In the current debate, there is a lot of work related to the topic "Future Internet". Interesting clean-slate approaches which are related to the work presented here can be seen in the "Role-Based Architecture" (RBA) [4], the "Service Integration, controL and Optimization" (SILO) [5], the "Recursive Network Architectures" (RNA) [6], and the "Data Oriented (and Beyond) Network Architecture" (DONA) [7] approaches.

The RBA approach introduces a non-layered architecture to the design of network protocols and organizes communication in functional units referred to as "roles". Roles are not hierarchically organized and thus may interact in many different ways. The main motivation for RBA is to address the frequent layer violations that occur in the current Internet architecture, the unexpected feature interactions that emerge as a result, and to accommodate middle boxes. The SILO approach also introduces a non-layered design based on silos of services. Furthermore it offers a more flexible header structure than the RBA approach. The overall goal of the SILO architecture is to facilitate cross-layer interactions in a manner that meets the user requirements accurately and optimizes performance. The RNA approach examines the implications of using a single, tunable protocol for different layers. RNA reuses basic protocol operations across different protocol layers, avoiding redundancy of implementation as well as encouraging cleaner cross-layer interaction. It allows protocols and protocol stacks to adjust at runtime. This results in a more dynamic composition of services, both within stacks and in the way networks combines the stacks of individual hops into an overall network architecture. DONA takes into account that the vast majority of today's Internet usage is data retrieval and service access, whereas the architecture was designed around host-to-host applications.

RBA, SILO, RNA, DONA, and our concept are similar in that all avoid layering and aim at defining a highly flexible architecture. The main motivation for RBA, SILO, and RNA was to address the frequent layer violations and cross-layer interactions that occur in the current Internet architecture. The approach presented here has a focus on network flexibility and evolution by utilizing principles of service-oriented architectures. The challenge is to implement these ideas in the lower layers of the network. In addition to RBA, SILO and RNA, the definition of interfaces between services as well as synergistic interaction of applications with the network are the focus of the presented approach.

Approaches like active and programmable networks [21] as well as dynamic configuration of protocol stacks discussed in [22] and [23] have influenced our work, too. Especially in the former approach, the separation of the communication plane from the control plane is a major concern. Such a separation is difficult to realize in today's Internet. This is because the network nodes are vertically integrated and content providers have no direct access to the control plane. Our approach on SOA principles based on horizontally distributed functionalities promises more flexibility as described in Section 5.

Beside these approaches, a lot of new initiatives within the 6th and 7th Framework Program of the European Commission about "Networks of the Future" are already started. Within FP6, it is worth to mention the projects HAGGLE [8], ANA [9] and DAIDALOS [10]. Noteworthy, the ANA project which aims to design and develop a novel autonomic network architecture has some similarities with the presented SOA approach. ANA tries to enable a flexible, dynamic, and fully autonomous formation of network nodes as well as whole networks. Our approach uses several concepts that are also found in ANA. The major differences to our approach is that ANA is based on functional blocks (FB) which are organized in sets (compartments) instead of using abstract services and (dynamic) workflows. In continuation of the initiatives within FP6, in FP7 some new projects are already started. For example 4WARD [11], TRILOGY [12] and PSIRP [13] are projects dealing with architectural questions. Moreover within FP7, Euro-NF [14] as a network of excellence and EIFFEL [15] as a support action are platforms for discussion of the European vision of the Future Internet.

Furthermore worldwide, there are programs under way addressing the topic "Future Internet". Here the projects under NSF funding like GENI [16] and FIND [17] and the Clean-Slate Program at Stanford University [18] should be mentioned. In Asia, the platform programs from Korea "Future of the Internet for Korea" (u-IT836) [19] and Japan's "Collaborative Overlay Research Environment" (CORE) [20] have an impact to this topic.

3 Demands on a Flexible Network Architecture

There are many demands on a future network architecture. Here we focus on the demand for a flexible architecture. This is important because a next generation network faces various requirements from yet unknown applications. It also has to adapt to new transport technologies (wired and mobile) with different capabilities, and should enable evolutionary changes of the network itself.

Adaptation to different transport technologies is required in order to integrate a wide range of endsystems like sensor nodes and high performance systems. This adaptability can be achieved by flexible provisioning of functionality.

The network should be flexible enough to handle different requirements of various applications. Changing requirements may also be caused by users. For example, requirements on trust, price and reliability may vary according to a user's goal [27]. It should be assumed that such requirements are given at runtime of an application only. Thus the network must react fast. This can be achieved by flexible selection of provided functionality.

In addition to such demands from the outside, there are demands for flexibility from the network itself. We assume that there will not be a fixed set of protocols/mechanisms, which are well suited to fulfill all requirements on all transport technologies for all time. This implies that evolutionary changes of networks must be possible to avoid extensive and thus costly transitions in the future.

Changing network functionality in a large scale network will inevitably lead to a heterogeneous network where different nodes provide different sets of functionalities. This is because a synchronous exchange of functionality affecting many million nodes residing in many thousands of administrative domains is an infeasible task. As a consequence, the architecture of a next generation Internet should be able to handle heterogeneity.

The nodes of the Internet today rely on the protocols of TCP/IP, i. e., the mechanisms of TCP/IP are mandatory for a node in the Internet. Thus it is hard to change these mechanisms. For example, substantial changes like migrating from IPv4 to IPv6 are going on for more than 15 years now. From this, we derive the requirement that there should be as few as possible mandatory mechanisms in a next generation network.

4 SOA Approach for the Future Internet

In our approach of a Future Internet architecture, we consider the Internet as a large, distributed (software) system. Hence we address a new inter-network architecture by using software engineering methodology. A promising methodology is the service-oriented architecture (SOA) paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains [25].[1] In the rest of this section, we describe how principles of service orientation can provide solutions for requirements described in the previous section. Figure 2 illustrates some terms used within our approach.

Services are the essential building blocks of SOA. A service provides self-contained functionality, has well-defined interfaces and must not make assumptions about internals of other services (loose coupling). A service is an abstraction of specific algorithms and data structures (i. e., mechanisms) used to implement the service.[2] These principles simplify adding and removing service as well as changing the implementation of services.

Workflows provide more complex functionality by selecting sets of services and defining their interaction. A workflow can be defined explicitly by a requester or may be derived from known dependencies. The dynamic definition of workflows allows responding in short-term to user requests or changing environmental conditions.

The goal is to encapsulate (micro-)protocols by services. Then services offer an abstract view on the functionality of these protocols, in contrast to hiding mechanisms by layers [26]. For example, there may be a service for "ensuring reliable transmission" which can be implemented by different retransmission protocols. Services and applications may depend on the functionality provided, but the protocols used to implement the service are transparent to them. Thus the service concept enables modifying or even exchanging protocols transparently to other services and services users.

Communication in general requires a common basis or language;

[1] Note: SOA is a paradigm, and thus it does not rely on a specific technology (e. g., Web Services) and is not limited to specific application areas (e. g., enterprise information integration).

[2] See [24] for more principles of SOA.

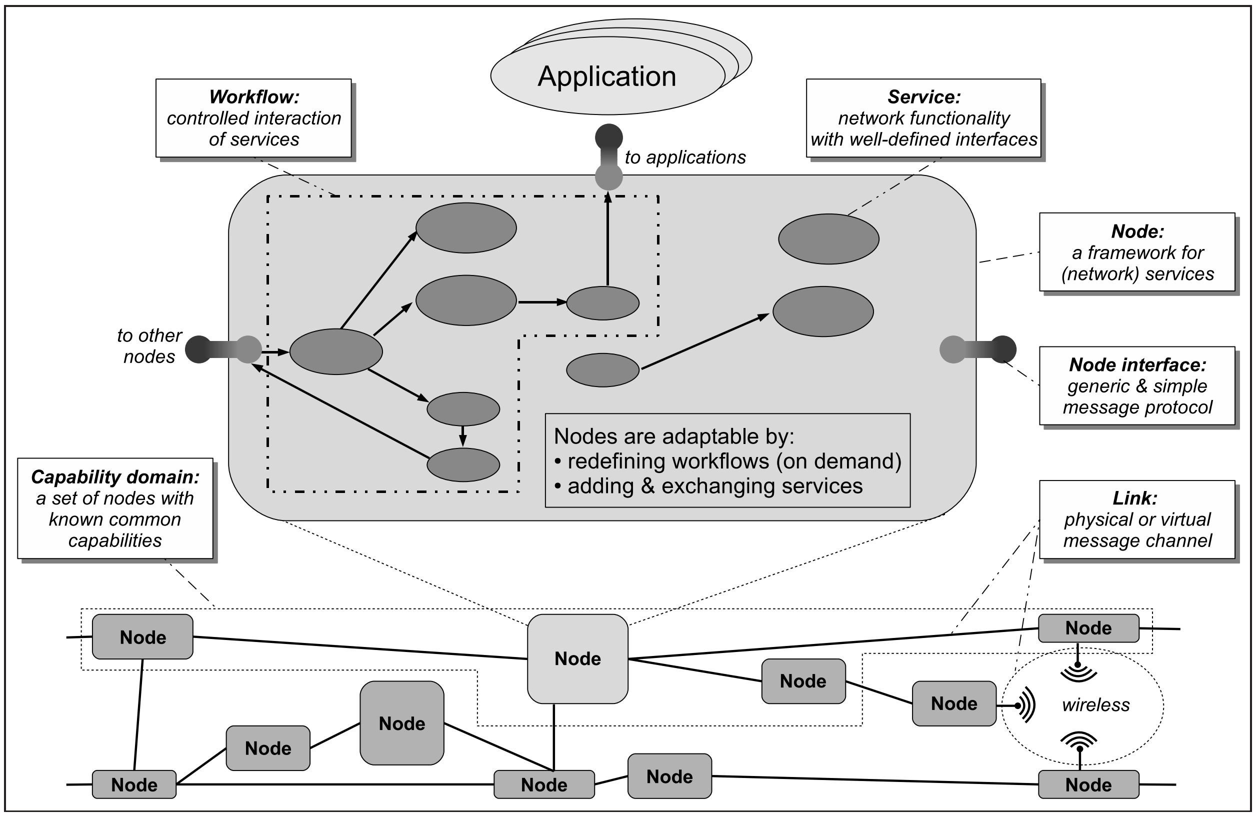
Application
Workflow: controlled interaction of services
Service: network functionality with well-defined interfaces
to applications
Node: a framework for (network) services
to other nodes
Node interface: generic & simple message protocol
Nodes are adaptable by:
• redefining workflows (on demand)
• adding & exchanging services
Capability domain: a set of nodes with known common capabilities
Link: physical or virtual message channel
Node
wireless

Figure 2 Basic concept of a service-oriented approach for a Future Internet architecture.

in case of computer networks common protocols are required. Thus different instances of the same service on different nodes have to use the same protocol. But changing and introducing services in a large scale network will inevitably lead to a heterogeneous network.

From the point of view of a single node, heterogeneity causes an uncertainty about the protocols that are supported by other nodes. A node can basically handle this in two ways: removing uncertainty in advance of communication or handling such problems during communication.

Removing uncertainty in advance of communication in turn requires communication between two or more nodes. Nodes may negotiate their capabilities with each other or may consult a registry immediately before user data will be exchanged. This causes a delay before communication can start, even if caching techniques are used. Thus these methods should be used rarely, e. g., when using highly specific protocols. Nodes that interact often could decide to negotiate their capabilities in advance. Such nodes can build capability domains to share information about common capabilities (see Fig. 2). A node may be a member of several capability domains covering different types of capabilities. For example, a set of nodes may share the capability of supporting specific address types. Capability domains could be defined statically or dynamically. The latter requires distribution of information. This might be implemented similarly to routing protocols and will also produce overhead by additional communication.

The second method is to handle uncertainty at runtime. This can be done at least by three different methods. (1) For some protocols it is appropriate if these are just ignored by nodes which do not support the protocols. For example having flow-control support within the network fosters efficiency, but it is not necessary that all nodes support flow-control mechanisms. (2) If there is uncertainty only about few alternatives then it might be possible to offer all alternatives. An example is to use two address types for the same destination. (3) Finally it is possible to delegate the processing to another node which is able to handle the protocol. The delegation may also inform the origin host to change its behavior.

5 Technical Considerations

Building a network based on principles of service-oriented architectures requires specific supporting techniques. Web Services and XML are obviously inappropriate to implement services on a network level. Thus we will investigate techniques and principles suitable for service-oriented network architectures.

As shown in Fig. 2, we assume that a network is made up of interconnected nodes and that each node provides a framework for network services. The common generic protocol provides a technique for message exchange between the same services on different nodes. All other techniques enable interaction of different services within the same node.

5.1 A common generic protocol

Communication between nodes takes place between instances of the same service only. Thus the task of a generic protocol is to separate messages of different services and to identify the service a message belongs to. This can be achieved by a sequence of simple type, length, value (TLV) structures, whereby the type identifies the service and the length identifies message boundaries. The values of a TLV can be a service specific data structure or may be a sequence of TLV again. Advantages of using TLVs are:

- Messages can be ordered arbitrarily. This supports loose coupling, e. g., services send data only when needed instead of using fixed protocol headers.
- The message length is variable. This enables adding extended or redundant data and thus supports heterogeneity.
- TLVs with an unknown type can be forwarded transparently. This also supports heterogeneity (ignore or delegate messages).

A sequence of TLVs can carry the same information as any other protocol because the type and length are additional meta-data only.

5.2 Flow and Context

The concept of flows and context is closely related to the technique for data exchange between services within a node. The term "flow" usually denotes a sequence of packets which are semantically related to each other. Within our approach, a flow denotes a sequence of semantically related messages.[3] Flows are separated by a specific *flow-ident* message. It is assumed that all messages arriving at the same interface belong to the same flow until a new flow-ident message is received. Thus on unreliable links, each data unit (e. g., packet or frame) must begin with a flow-ident message to ensure that messages are processed in the correct context. The value of a flow-ident message can contain arbitrary data types, typically some kind of addresses or labels. Such a flow identifier is valid only on one link. Nodes sharing the same link are free to change flow identifiers as needed.

All messages that belong to the same flow are processed within the same context. The context is implemented as a tuple space (a set of <name, value> pairs). Whenever a service receives a message for processing, it also gains read and write access to the associated context. Thus the context enables data exchange between services with temporal and referential decoupling of services [28]. For example, a routing service may provide information about an outgoing port without having knowledge when or which service may read that information from the context. All status information of a service is kept in the context so that each service is stateless. This fosters the autonomy of services and improves their stability.

Each message is associated with a flow and thus also with a context. But this does not imply a connection-oriented behavior. Connection-less behavior (i. e., using stateless protocols) is implemented by deleting a context whenever a new flow-ident message is received (i. e., the context is transient). Connection-oriented behavior (i. e., using stateful protocols) is implemented by storing the context for later reuse whenever a new flow-ident message is received (i. e., the context is persistent).

Flows can be designed to transport payload transparently. Such a flow may be used as a virtual link by other flows, i. e., a flow can be embedded into another flow. This way, a flow might offer common functionality like reliable and secure transmission which can be reused by several other flows. The concept of embedding flows is similar to layering, but the functionality of a flow is not fixed and can be defined as needed.

5.3 Dispatcher

Dispatchers are responsible for assigning incoming messages to services. Therefore services must register to the types of messages they are willing to process. Dispatchers also control the sequence of message processing. The trivial case is sequential processing in order of arrival. In addition to that, independent services might be processed in parallel. In case of known dependencies, a dispatcher might even reorder messages. Such dependencies might be specified explicitly by an administrator or may be derived from read and write accesses to the context of a flow (e. g., there must a be routing decision before data can be forwarded). Defining a processing order corresponds to the definition of a workflow in SOA.

A service is called with three input parameters: a message that has triggered the service, a context and an optional list of additional messages. The output parameters are a list of messages and an indicator for success or failure. If a service fails, all succeeding messages will be skipped up to the next flow-ident message. For example, an expired time-to-live or a CRC error will cause an interruption of the current workflow.

5.4 Notification Broker

Notifications enable dynamic control of service processing. Notifications are handled by a notification broker. Services can publish named-notifications and can subscribe for specific notifications or classes of notifications. In contrast to a message arriving on a link, notifications can be delivered to zero, one or multiple services, whereas a message is expected to be handled by exactly one service. Just like messages, notifications are associated to a context.

[3] It is possible to associate several messages within a packet with different flows.

An example for using notifications is a queuing service that detects the lack of memory resources. The queuing service may then publish a notification. The notification broker can forward the notification to a flow control service and to a resource monitoring service if these have subscribed for this type of notifications in advance.

6 Summary and Outlook

In this article, we argued that a next generation network must be highly flexible by design. Then we discussed how typical principles of SOA can offer a viable approach to such an architecture. These are autonomous services, dynamic generation of workflows, handling heterogeneity of the network, loose coupling, and well-defined interfaces.

In order to investigate service-orientation on a network level, our approach is to implement an experimental infrastructure that provides several techniques to enable or at least foster the implementation of the mentioned SOA principles: a generic protocol based on TLV structures fosters loose coupling of services and thus supports autonomous services. Managing the status of a flow in a context which is handled by the service framework enables stateless services. A dispatcher and a notification broker enable dynamic workflows and thus allow short-term flexibility. Heterogeneity is supported by flexible TLV message structures. Supporting heterogeneity by removing uncertainty could be implemented through the use of specific services. Well defined interfaces and loosely coupled services abstract from service mechanisms and thus avoid the definition of mandatory mechanisms, which are hard to change in the future.

We plan to use the described techniques to implement several communication scenarios in order to investigate how service-orientation can be supported on a network level. Particularly the mechanisms of TCP/IP will be adopted to demonstrate how well known functionality can be organized according to service-oriented principles. This will also offer the opportunity to provide a seamless interaction between the current Internet and an experimental network architecture.

References

[1] D. Clark et al. New Arch: Future Generation Internet Architecture. Final Technical Report, http://www.isi.edu/newarch/.

[2] IEEE Std. 1471-2000 IEEE Recommended Practice for Architectural Description of Software-Intensive Systems – Description.

[3] How Do You Define Software Architecture? *Software Engineering Institute, Carnegie Mellon*. http://www.sei.cmu.edu/architecture/definitions.html.

[4] R. Braden, T. Faber, and M. Handley. From Protocol Stack to Protocol Heap – Role-Based Architecture. In: *Proc. of the First Workshop on Hot Topics in Networking (Hotnets-I), ACM SIGCOMM*, Princeton, NJ., Oct. 2002.

[5] R. Dutta, G. N. Rouskas, I. Baldine, A. Bragg, and D. Stevenson. The SILO Architecture for Services Integration, Control, and Optimization for the Future Internet. In: *IEEE Int'l Conf. on Communications*, ICC apos 2007.

[6] J. D. Touch, Y. S. Wang, and V. Pingali. A Recursive Network Architecture. *ISI Technical Report ISI-TR-2006-626*, Dec. 2006.

[7] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A Data-Oriented (and Beyond) Network Architecture. In: *ACM SIGCOMM Computer Communication Review*, Vol. 37, issue 4, Oct. 2007.

[8] J. Scott, P. Hui, J. Crowcroft, and C. Diot. Haggle: A Networking Architecture Designed Around Mobile Users. In: *IFIP WONS 2006*, Jan. 18–20, Les Menuires, France.

[9] ANA: Autonomic Network Architecture. http://www.ana-project.org/.

[10] C. Werner, Y. Liang, J. Jähnert, and M. Ebner: Daidalos – A scenario based approach from scenarios towards integration.

[11] 4ward. http://www.4ward-project.eu.

[12] Trilogy. http://www.trilogy-project.org.

[13] Publish-Subscribe Internet Routing Paradigm. http://www.psirp.org.

[14] Euro-NF. http://euronf.enst.fr/en_accueil.html.

[15] EIFFEL: Evolved Internet Future for European Leadership. http://www.fp7-eiffel.eu/.

[16] L. Peterson, T. Anderson, D. Blumenthal, D. Casey, D. Clark, D. Estrin, J. Evans, D. Raychaudhuri, M. Reiter, J. Rexford, S. Shenker, and J. Wroclawski: GENI design principles. In: *IEEE Computer Magazine*, 2006.

[17] Future INternet Design (FIND). http: //www.nsf.gov/pubs/2007/nsf07507/nsf07507.htm.

[18] Cleanslate. http://cleanslate.stanford.edu/.

[19] u-IT839 Future of the Internet for Korea. http://eng.mic.go.kr/eng/user.tdf?a=user.index.IndexApp&c=1001.

[20] CORE. http://www.nict.go.jp/index.html.

[21] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. A survey of active network research. In: *IEEE Communications Magazine*, Jan. 1997, Vol. 35, issue 1.

[22] S. W. O'Malley and L. L. Peterson. A Dynamic Network Architecture. In *ACM Trans. on Computer Systems*, Vol. 10, No. 2, May 1992, pp. 110–143.

[23] M. Zitterbart, B. Stiller, and A. N. Tantawy. A Model for Flexible High-Performance Communication Subsystems. In: *IEEE Journal on Selected Areas in Communications*, Vol. 11, No. 4, May 1993.

[24] T. Erl. Service-Oriented Architecture Concepts, Technology, and Design. *Prentice Hall*, 2005.

[25] OASIS Reference Model for Service Oriented Architecture 1.0. Official OASIS Standard, Oct. 12, 2006.

[26] J. Day. Patterns in Network Architecture – A Return to Fundamentals. *Prentice Hall*, 2008.

[27] D. Clark, J. Wroslawski, K. Sollins, and R. Braden. Tussle in Cyberspace: Defining Tomorrow's Internet. In: *ACM SIGCOMM*, Aug. 2002.

[28] D. Gelernter. Generative communication in Linda. In: *ACM Transa. on Programming Languages and Systems*, Vol. 7, No. 1, Jan. 1985.

1

2

1 **Prof. Dr. Paul Mueller** is a Professor for computer science and head of the computing department at the Technical University of Kaiserslautern, Germany. His research group on Integrated Communication Systems (www.ICSY.de) focuses on communication systems and service-oriented architectures (SOA), with special interests in Future Internet and Grid architectures.
Address: University of Kaiserslautern, Gottlieb-Daimler-Straße 47, 67663 Kaiserslautern, Germany, Tel.: +49-631-2052263, Fax: +49-631-2052161,
E-Mail: pmueller@informatik.uni-kl.de

2 **Bernd Reuther** is a research assistant in the department of computer science at the University of Kaiserslautern. His research interests include middleware, service-oriented architectures, computer networks and concepts for evolution of networks. He graduated in computer science 1996 at the University of Kaiserslautern.
Address: University of Kaiserslautern, Gottlieb-Daimler-Straße 47, 67663 Kaiserslautern, Germany, Tel.: +49-631-2052161, Fax: +49-631-2053056,
E-Mail: reuther@informatik.uni-kl.de