Design Process and Development Tools for Concurrent Future Networks

Lars Völker, Denis Martin, Timo Rohrberg,
Helge Backhaus, Peter Baumung, Hans Wippel, Martina Zitterbart
Institut für Telematik, Universität Karlsruhe (TH), Germany

I. Introduction

Building a Future Internet with Network Virtualization as basis [1] results in a very flexible and adaptive solution. A variety of virtual networks can be instantiated with different networking protocols; thus, allowing to use protocols tailored to the applications' and users' needs. Nodes within such a Future Internet have to be structured accordingly [2], [3] and require specialized functionalities, e. g. enabling the selection of the "best" available network and protocols for fulfilling specific communication requests [4].

However, the possibility to add tailored protocols to a specialized node is not enough since conceiving and implementing these protocols can be very time-consuming and therefore expensive. A promising way to limit the cost is the approach of protocol composition as new services may be created by re-using existing protocol building blocks. In addition, the usage of an optimized design process and design tools, may reduce development cost and increase the quality of the protocols created.

In this paper, we present a life-cycle for protocol composition and a flexible design process to support it. They lay a foundation for examining different aspects of protocol composition. In addition, design tools currently being developed will be shown to present how the concepts can be applied.

This paper is structured as follows: In Section II we present our protocol composition life-cycle. We then discuss the design process (Section III) as a major part of it and show first design tools (Section IV). We end this paper with related work (Section V) and the conclusion (Section VI).

II. A Life-Cycle for Network Services

According to the IEEE Software Engineering Glossary [5], a software life-cycle is defined as "the period of time that begins when a software product is conceived and ends when the software is no longer available for use. The software life-cycle typically includes a concept phase, requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase, and, sometimes, retirement phase. [...]". The software development model, in contrast, does not include operation, maintenance, and retirement phases.

In Figure 1, we show our life-cycle for composed network services. Note that for reducing the system's overall complexity and especially the complexity of service deployment, we discern the (off-line) design and development of components (visible in the upper *design time* part of the image), and the (on-line) selection and operation at *run time* of components in the image's lower part.

The life-cycle starts in the image's upper right with an elaborate *abstract service model*, i. e. a formal description of the service, and its *properties*. Using specific *design tools*, a *design* as well as an *implementation* of *components* reflecting the service can be created. This may be an iterative process as design decisions may not only lead to refinements of the service model but also to changes of the respective properties.

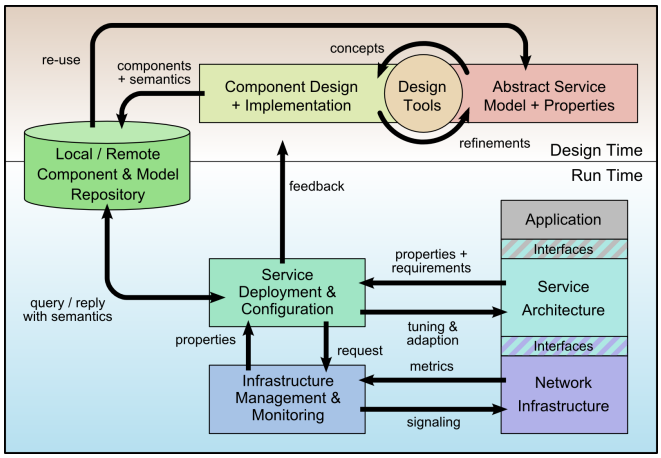

Figure 1. Life-Cycle for Composed Network Services

An implemented component is placed in the *component repository*, including its model and general semantics, such as the component's description and properties. Several types of repositories may exist: local repositories on each network node, remote repositories accessible within an entire network, and design only repositories for *re-using* existing components and models when designing new services. Each type comes with different complexities and properties, and its choice depends on the target network's type and intended use. For instance, dynamic requests for service components from remote locations might be unsuitable in sensor networks.

For a static and local repository the *service deployment and configuration* is done during the deployment of the network nodes, and configuration is limited to adaptions during run time. Network *infrastructure management and monitoring* functions ensure a stable network during run time and change service configurations according to the properties of the un-

derlying infrastructure.

Last, but not least, a run time environment, i. e. a *service architecture*, is needed for running the instantiated service components. In [2], [3] we have proposed a possible service architecture allowing to run several protocol architectures in parallel – a key aspect for concurrent future networks.

III. Design Process

This section focuses on the design phases of network services, and thus mostly relates to the upper part of Figure 1 described in the previous chapter.

The development model in Figure 2 puts this design process in relation to Software Engineering development processes. The phases of the development model are not self-contained but will provide feedback to the other phases. Therefore, as in Software Engineering, the whole design process is an iterative process that will refine all intermediate models until the desired end-product is in a satisfactory state. The main difference to the combined waterfall/incremental development model often used in software development is, that the network development model here allows for feedback between any preceding phases. In addition, the last step of the iterative or incremental process is the component model, not the actual source code. This model includes semantics and a complete documentation of the component, and is stored in a (design) repository.

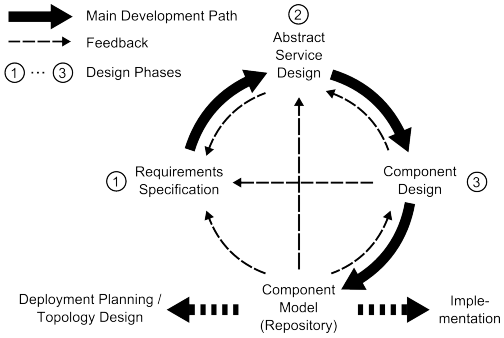

Figure 2. Simplified Design Process

According to this model, actual implementations for various hard- and software platforms can be created – a real advantage over informal protocol specifications. In addition, the model can be used for tool supported deployment planning and network topology design, similar to some vendor specific tools already existing today [6].

In the design of complex systems, abstractions are inevitable. Figure 3 shows the abstraction levels which we consider suitable for network architecture design. At the *Network Level*, the network's topology is regarded, including the necessary node types. This level gives a coarse overview of the network. The *Node Level* describes the services certain node types provide as a set of components which we call *Netlets* [2], [3]. Netlets composed of building blocks are described in more detail at the *Netlet Level*. Here, existing building blocks can be re-used to construct new Netlets. Finally, the *Building Block Level* describes the process model of a building block. This is, for instance, well covered by SDL-MDD [7].

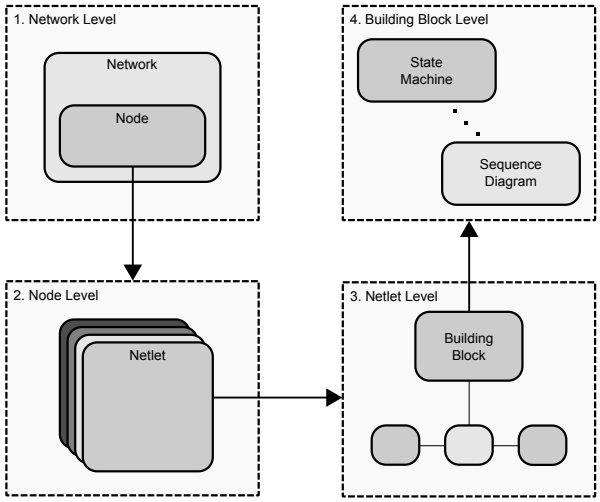

Figure 3. Abstraction levels during the Design Process

IV. Tool Support

For the Netlet Level described in the previous section, an investigation of possible tool support has taken place. A prototype of a composition tool was implemented as an Eclipse plug-in based on the GEF framework. Its main features is the automated aggregation of functional block properties as described in [4]. This paper also describes the needed selection process, which works on the combined properties of the composite (Netlet). Figure 4 shows the resulting latency profile of an example Netlet consisting of a fragmentation block and a block that adds CRC checksums to outgoing packets.

Currently, we are extending this tool to cover the Node Level. By emulating the run time selection algorithm, the tool aids the network architect in finding the most suitable

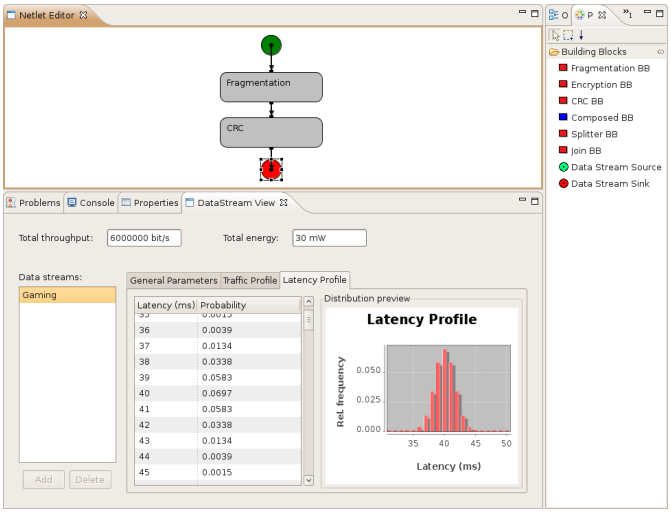

Figure 4. Property Aggregation Tool

set of Netlets, based on expected traffic and application requirements.

V. Related Work

In the area of protocol composition various related work exists. Many systems using service composition have been proposed, for example [8], [9]. Other work focused more on formal aspects, like [10], [11]. While in the past the flexibility of protocol composition was often considered in the context of high-speed transmission protocols, current work has shifted more generically towards the Future Internet [12]–[14].

Well-known formal description languages that already have been applied to networking include the Specification and Description Language (SDL) [15], Extended State Transition Language (Estelle) [16], and Language Of Temporal Ordering Specifications (LOTOS) [17]. Some newer examples include State Chart XML [18], UML [19], or Cosmogol [20]. In addition, OPNET Modeler has some visual modeling capabilities for the implementation of protocols and topology design, which is mainly used for simulation. Description languages however are just one part to the solution, design processes are also important. An interesting approach that combines SDL with model-driven development methods is described in [7].

VI. Conclusion and Future Work

In this paper, we have presented a design process and development tools for the protocol composition life-cycle, which can tremendously reduce the cost of creating new protocols for concurrent protocol architectures within the Future Internet.

In future work, we will further detail the design process itself and continue the development of the tools sustaining the process of components design. Also the development of additional tools and the integration of already existing tools are major topics for future work.

In the broader context of protocol composition and especially protocol deployment within a Future Internet, a large number of open issues arise, requiring dedicated attention. Figure 5 depicts just some of them.

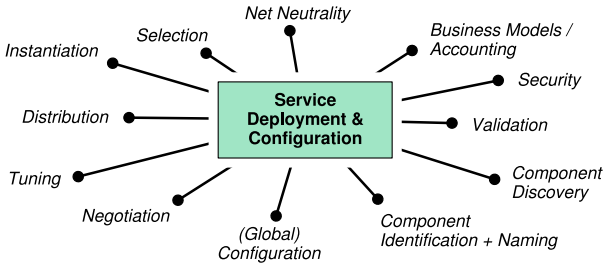

Figure 5. Service Deployment Issues

Acknowledgment

This work was carried out in parts within the research projects 4WARD, G-Lab as well as the Graduate School IME which are funded by the European Commission (within 7th framework programme), the German Ministry of Education and Research (BMBF) and the German Research Foundation (DFG) respectively.

The authors would like to thank Joe Touch (ISI/USC) and Reinhard Gotzhein (TU Kaiserslautern) for valuable discussions about service composition.

References

[1] N. M. K. Chowdhury and R. Boutaba, "A Survey of Network Virtualization", Technical Report, David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, Tech. Rep. CS-2008-25, Oct 2008.

[2] L. Völker, D. Martin, I. El Khayat, C. Werle, and M. Zitterbart, "An Architecture for Concurrent Future Networks", in *2nd GI/ITG KuVS Workshop on The Future Internet*. Karlsruhe, Deutschland: GI/ITG Kommunikation und Verteilte Systeme, Nov. 2008.

[3] L. Völker, D. Martin, I. El Khayat, C. Werle, and M. Zitterbart, "A Node Architecture for 1000 Future Networks", in *Proceedings of the International Workshop on the Network of the Future 2009*. Dresden, Germany: IEEE, Jun. 2009, (to appear).

[4] L. Völker, D. Martin, C. Werle, M. Zitterbart, and I. El Khayat, "Selecting Concurrent Network Architectures at Runtime", in *Proceedings of the IEEE International Conference on Communications (ICC)*. Dresden, Deutschland: IEEE Computer Society, Jun. 2009, (to appear).

[5] *IEEE Standard Glossary of Software Engineering Terminology*, Standards Coordination Committee, IEEE Computer Society Std. 610.12-1990, Dec 1990.

[6] Miercom, "Miercom Test – Cisco Business Communications Deployment and Management Apps", Whitepaper, October 2005.

[7] T. Kuhn, R. Gotzhein, and C. Webel, "Model-Driven Development with SDL - Process, Tools, and Experiences", *Model Driven Engineering Languages and Systems*, vol. 4199/2006, pp. 83–97, 2006.

[8] N. C. Hutchinson and L. L. Peterson, "The x-kernel: An architecture for implementing network protocols", *IEEE Trans. Softw. Eng.*, vol. 17, no. 1, pp. 64–76, 1991.

[9] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek, "The click modular router", *SIGOPS Oper. Syst. Rev.*, vol. 33, no. 5, pp. 217–231, 1999.

[10] D. D. Clark and D. L. Tennenhouse, "Architectural considerations for a new generation of protocols", *SIGCOMM Comput. Commun. Rev.*, vol. 20, no. 4, pp. 200–208, 1990.

[11] M. Zitterbart, B. Stiller, and A. Tantawy, "A model for flexible high-performance communication subsystems", *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 4, pp. 507–518, May 1993.

[12] J. D. Touch, Y.-S. Wang, and V. Pingali, "A Recursive Network Architecture", ISI, Tech. Rep., Oct 2006, iSI-TR-2006-626.

[13] 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 *Proc. IEEE International Conference on Communications ICC '07*, G. N. Rouskas, Ed., Glasgow, Scotland, 2007, pp. 1899–1904.

[14] M. Sifalakis, A. Louca, A. Mauthe, L. Peluso, and T. Zseby, "A functional composition framework for autonomic network architectures", in *Proc. IEEE Network Operations and Management Symposium Workshops NOMS Workshops 2008*, 7–11 April 2008, pp. 328–334.

[15] *Specification and Description Language (SDL)*, International Telecommunication Union (ITU) ITU-T Recommendation Z.100, Aug 2002.

[16] S. Budkowski and P. Dembinski, "An introduction to Estelle: a specification language for distributed systems", *Computer Networks and ISDN Systems*, vol. 14, no. 1, pp. 3–23, 1987.

[17] M. D. P.H.J. van Eijk, C.A. Vissers, *Formal Description Technique Lotos: Results of the Esprit Sedos Project*, P. Van Eijk and Michel Diaz, Ed. New York, NY, USA: Elsevier Science Inc., 1989.

[18] *State Chart XML (SCXML): State Machine Notation for Control Abstraction*, World Wide Web Consortium (W3C) Std., May 2008.

[19] *Unified Modeling Language*, Object Management Group (OMG) Std., Rev. 2.1.2, Nov 2007.

[20] S. Bortzmeyer, "Cosmogol: a language to describe finite state machines", Nov 2006, expired IETF Draft.