ADAPT: A Semantics-oriented Protocol Architecture

Stefan Götz, Christian Beckel, Tobias Heer, Klaus Wehrle

Distributed Systems Group
RWTH Aachen University, Germany
{goetz, heer, wehrle}@cs.rwth-aachen.de
beckel@informatik.uni-tuebingen.de

Abstract. Although modularized protocol frameworks are flexible and adaptive to the increasing heterogeneity of networking environments, it remains a challenge to automatically compose communication stacks from protocol modules. The typical static classification into network layers or class hierarchies cannot appropriately accommodate cross-cutting changes such as overlay routing or cross-layer signaling.
In this paper, we discuss how protocol composition can be driven by functionality and demand at runtime based on extensible semantic models of protocols and their execution environment. Such an approach allows to reason about the functionality and quality of automatically composed and adapted protocol compounds and it is open to existing and future protocols.

1 Introduction

The static nature of the classic TCP/IP protocol stack and its increasing complexity has prompted research in the area of dynamic and modularized communication subsystems from a number of different angles [1,2]. Ideally, a communication subsystem should resemble a configurable, dynamic framework of individual protocols that, in their entirety, provide a target functionality. To maximize modularity, protocol implementations should adhere to a uniform interface while information about protocol semantics is represented separately [3].

In addition to protocol functionality, a large number of factors play a role in composing a protocol stack that best matches its operational environment: dependencies among components, the network configuration (e.g., requiring authentication or tunneling), capabilities of communication partners (such as their support for specific protocols), user preferences (privacy, cost, etc.), and classic Quality-of-Service (QoS) metrics (e.g., application requirements or device capabilities). The length and incompleteness of this list illustrates the main deficiency of existing approaches: they describe protocols through strict design-time classifications, such as class hierarchies or languages with a fixed vocabulary, and are thus only poorly extensible. Consequently, they cannot incorporate new protocols, functionalities, or requirements.

In this paper, we discuss automated protocol composition in our dynamic protocol framework ADAPT. It leverages a semantic description format based on ontologies that provides an abstract notion of a protocol's functionality and properties to guide protocol composition. Transparently to the user, mobile applications can use dynamically composed protocol stacks that permanently adapt to their needs as well as to changes in the execution and network environment, as caused by, for example, roaming and network hand-offs. ADAPT's generic protocol model is not bound to established layering conventions and consistently integrates tunneling, encapsulation, and transformation (e.g., for VPNs, overlays, encryption), and significantly simplifies protocol deployment. Despite these significant differences from a traditional TCP/IP stack, ADAPT strives for transparency to existing protocols and applications.

2 Related Work

In modularized protocol frameworks, the composition of a protocol stack needs to satisfy criteria like protocol dependencies or functional requirements (for example to include loss handling or to perform compression before encryption). F-CSS [4] and DaCapo [5] explicitly separate this information from the implementation and represent it through custom languages. However, these description formats fail to achieve a clean separation since they expose implementation aspects. Also, they rigidly encode the protocol-related knowledge at design time and do not lend themselves to later extension.

In knowledge representation, ontologies have received wide-spread attention in particular around the semantic web and for web service management [6,7], but also innumerable other fields of information science. Ontologies strike a compromise between formalism and expressiveness that allows for being intuitive, generic, extensible, and powerful for reasoning and querying [8]. Zhou and associates apply this approach to the protocol domain [9] but their framework and modeling centers on a classic stack design of monolithic protocols ignoring decomposition and protocol extensions. Their ontology bases on the Internet protocol layers to reduce redundancy in the description and to reduce the complexity of the orchestration process. Consequently, this approach cannot support non-classic protocol arrangements such as overlay routing. ADAPT lifts this restriction by removing the layer structure from the protocol model. Thus, the complexity of protocol orchestration increases significantly and forms the challenge which we address in this paper.

3 Design

The basis of ADAPT is an end-system communication framework in which protocols operate as software components which share uniform interfaces and which can be instantiated, configured, and replaced at runtime. This framework follows the basic concepts of other componentized OS and network systems such as the x-Kernel [2] so that individual protocols can be composed into *protoocol chains*.

3.1 Semantic Protocol Modeling

The semantic model of protocols and their execution environment formally describes protocols and the orchestration criteria. Adapt uses *OWL DL*, a sublanguage of the Web Ontology Language (OWL) [10], for its high expressiveness and decidability in reasoning. In OWL DL, knowledge is represented by classes with class properties, individuals belonging to classes, and relations between them. Our ontology models not only protocols as individual classes but also the orchestration criteria and the relationships between them. This allows for extensibility with new criteria and forms of relationships as opposed to a model based on class properties.

To support the orchestration process and its individual stages, we distinguish three categories of models: functionalities, dependencies, and qualitative information. The functionality model derives directly from individual protocol functionalities, such as session support or loss handling. By sub-classing, it expresses a refinement of a more abstract functionality (e.g., packet retransmission is a sub-class of the loss handling class). The dependency model establishes associations between protocols, functionalities, and user-defined criteria. It also establishes *and* and *or* relationships between multiple dependencies (e.g., to enforce the inclusion of one of two specific encryption protocols). Furthermore, the ontology provides *qualitative information* about such aspects as protocol resource demands. Here, we distinguish between *requirements* a protocol imposes on its protocol chain or the environment (e.g., the existence of a DNS server) and information that solely affects the *ranking* of different protocol chains.

Based on the explicitly defined *asserted model*, the reasoning process derives an *inferred model* that represents additional knowledge. Adapt employs the following reasoning capabilities, primarily as fundamental means to integrate future protocols, orchestration criteria, and metrics.

Type inheritance: by inference, individuals of class X inherit the types of X's super classes. Thus, protocols describe their functionality precisely (e.g., RSA encryption) and can later be classified more generically (e.g., as providing confidentiality) through newly introduced knowledge.

Inverse properties: a symmetric relation between X and Y specified only for X is inferred to apply to Y. Thus, the inferred knowledge base remains consistent despite the incorporation of new knowledge.

Instance classification: *defined classes* classify individuals through a set of logic expressions. Consequently, the knowledge about the criteria of class membership receives an explicit representation.

Rule support: user-defined rules allow to assert new facts about individuals. Protocols can be asserted to support reliability, for example, if they provide ordered data delivery, retransmission, and a checksum algorithm.

OWL DL represents information in an abstract syntax or *RDF/XML* format. It allows to describe protocol semantics as a single unit which can be easily exchanged among endsystems and merged with other pre-existing ontologies at runtime.

3.2 Orchestration Criteria

The process of protocol orchestration in Adapt is driven, on the one hand, by the functionality and inter-dependencies of the protocols and, on the other hand, by the properties of the execution and network environment. Although many of these factors relate to typical QoS parameters, our research focus lies not on a QoS framework but on supporting arbitrary protocols at all layers. We distinguish four broad categories in the ontology:

Network capabilities primarily influence the orchestration of the network-related lower-level elements of chains. They range from local properties, such as a mandatory link protocol, across remote factors (e.g., the necessity of authentication) to QoS aspects, such as link loss rates or latencies.

Device capabilities are of a similar qualitative nature and primarily reflect information about the available CPU, memory, and energy resources.

Application requirements stem directly from application requests, e.g. to establish a new connection. They comprise functional requirements, such as session semantics, and qualitative aspects (a preference for low latency, for example).

The user influences via their *user preferences* how the above qualitative factors are traded against each other in the orchestration process. Typical trade-offs concern security, cost, and performance aspects. User preferences also provide immediate configuration information, for example fixed IP addresses, authentication information, and wireless network priorities. Finally, they allow users to influence the orchestration process such that, e.g., VPN tunneling is enforced for specific networks.

3.3 Orchestration Process

The fundamental goal of protocol orchestration is to determine the protocol chain best suited to the given application and environment requirements. Finding such a chain in the full set of all possible protocol combinations (a selective approach) suffers from limited scalability with a growing number of protocols. Adapt thus follows a constructive approach in two phases. First, it composes only the protocol chains that are functionally viable and fulfill all requirements imposed by the application request and the current execution environment. In the second phase, an expert systems ranks each resulting chain to determine the one which matches the environment best.

Composition The semantic composer consecutively relies on three types of composition information contained in the semantic protocol descriptions. First, it evaluates the functional requirements of the application and the execution environment to obtain all protocols that are necessary to satisfy these demands. Next, it recursively resolves the dependencies among protocols and constructs partial protocol chains (*stubs*) from this information. Finally, it merges the partial chains into complete functional compounds that can later be instantiated for packet processing.

For the dependency resolution of the second step, the semantic composer distinguishes direct dependencies, which need to be satisfied by the next protocol in the chain (e.g., a specific address format), and indirect dependencies, which can be fulfilled further down the chain (e.g., encryption). For each protocol, resolving dependencies is a recursive process that creates individual stubs for each alternative. To avoid a state explosion during stub merging, the composer obeys the ordering imposed by the dependencies, immediately discarding chains contradicting that order. Since mechanisms like tunnels or overlays change and potentially contradict regular ordering constraints, so-called *connector modules* allow overriding them. Finally, the composer validates the chains so they satisfy their internal dependencies and the external requirements and discards non-matching chains.

Ranking The ranking phase aggregates the quality metrics of individual protocols to determine a single chain to instantiate. While such qualitative information is available for protocols from their semantic model, the Adapt runtime provides the equivalent information for the network, the endsystem, and the user preferences. The latter also specify optional weight factors for each of these properties to bias their influence on the result. Based on these inputs, the expert system first matches the corresponding protocol and environment properties to derive per-protocol metrics. Then, it feeds them to a multi-criterion decision making system, which aggregates the metrics for each chain to arrive at a ranking order of all chains. Thus, the highest-ranking protocol chain emerges as the best match for the current communication requirements.

4 Results

As an initial evaluation, we tested the protocol orchestration with three typical functionality requests: support for a reliable connection, for multicast, and for name resolution. The ontological model contained 14 protocols and protocol extensions (e.g., an extension to TCP which makes it perform better in scenarios with high bit error rate), tunnels (e.g., in case IP Multicast is not supported by the network the host resides in), and network requirements (existence of a DNS server). The matchmaker, the composition engine, and the stub expert system are implemented in Java based on the Jena semantic web framework which manages the ontology and provides a reasoner, a query engine, and basic rule support. The composition engine uses SPARQL [11] to perform queries on the description repository. All measurements were performed on a Linux system with a 1.80 GHz Intel Pentium M processor and 1GB RAM, employing Sun's Java 1.5.0 runtime environment and version 2.5.5 of the Jena library.

Table 1 lists the duration of the matchmaking process, the protocol composition process, and the number of resulting protocol chains for queries which specify the desired functionality concisely. More loosely specified queries are satisfied by more chains but they also increase the composition time significantly, as Table 2 illustrates. We see considerable room for improvement in our current

Query	Matching	Compos.	Sum	# chains
Reliability	43 ms	117 ms	166 ms	6
Name resolution	30 ms	136 ms	166 ms	4
Multicast query	53 ms	62 ms	115 ms	18

Table 1. Protocol composition based on restrictive queries

Query	Matching	Compos.	Sum	# chains
Reliability	33 ms	222 ms	255 ms	226
Name resolution	29 ms	375 ms	404 ms	655
Multicast query	54 ms	112 ms	166 ms	88

Table 2. Protocol composition based on non-restrictive queries

implementation, e.g., via additional, though less generic, rules in the composition process, pre-computation, and caching of chain stubs. Although an evaluation of the expert system is still outstanding, these initial results strongly suggest the practical viability of a functional composition of protocol modules based on ontological models.

References

1. Muhugusa, M., Di Marzo, G., Tschudin, C.F., Harms, J.: ComScript: An Environment for the Implementation of Protocol Stacks and their Dynamic Reconfiguration. In: International Symposium on Applied Corporate Computing. (1994)
2. Hutchinson, N.C., Peterson, L.L.: The x-Kernel: An Architecture for Implementing Network Protocols. IEEE Transactions on Software Engineering **17**(1) (1991)
3. O'Malley, S.W., Peterson, L.L.: A Dynamic Network Architecture. ACM Trans. Comput. Syst. **10**(2) (May 1992) 110–143
4. Zitterbart, M., Stiller, B., Tantawy, A.N.: A Model for Flexible High-performance Communication Subsystems. Selected Areas in Communications, IEEE Journal on **11**(4) (1993) 507–518
5. Plagemann, T., Vogt, M., Plattner, B., Walter, T.: Modules as Building Blocks for Protocol Configuration. In: Network Protocols, 1993. Proceedings., 1993 International Conference on. (1993) 106–113
6. Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.P.: Semantic Matching of Web Services Capabilities. In: ISWC '02: Proceedings of the First International Semantic Web Conference on The Semantic Web, London, UK (2002) 333–347
7. Li, L., Horrocks, I.: A Software Framework for Matchmaking Based on Semantic Web Technology. In: Proceedings of the 12th International Conference on World Wide Web, New York, NY, USA (2003) 331–339
8. Wang, X.H., Zhang, D.Q., Gu, T., Pung, H.K.: Ontology Based Context Modeling and Reasoning Using OWL. (2004) 18–22
9. Zhou, L., Pung, H.K., Ngoh, L.H., Gu, T.: Ontology Modeling of a Dynamic Protocol Stack. In: 31st IEEE Conference on Local Computer Networks, Los Alamitos, CA, USA, IEEE Computer Society (November 2006) 353–360
10. McGuinness, D.L., van Harmelen, F.: OWL Web Ontology Language Overview. W3C Recommendation (February 2004)
11. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. W3C Recommendation (January 2008)