A Building Block Interaction Model for Flexible Future Internet Architectures

Dennis Schwerdel, Daniel Günther, M. Rahamatullah Khondoker, Bernd Reuther, Paul Müller
Integrated Communication Systems Lab, University of Kaiserslautern, Germany
{schwerdel,guenther,khondoker,reuther,pmueller}@informatik.uni-kl.de

Abstract

Today’s Internet has a static architecture that makes introducing new functionality a complex and costly task, so the Internet can not keep pace with rising demands and new network capabilities. Therefore, evolvability and flexibility are the keys to a future Internet architecture. In this paper we propose a building block interaction model that can be used to build highly flexible and evolvable network architectures. The paper also explains how the interaction model can be used as a basis for automatic protocol selection and composition.

1. Motivation

Future networks must be more flexible than the Internet today, because the types of applications, access networks, and devices using the network is continuously increasing [2]. In addition to technical changes there are also different usage scenarios and even different political constraints [4]. In order to achieve such flexibility, concepts of software-engineering can be adapted to build networks [13]. A common approach in several recent and current research projects is to use a modular design of network architectures (see also section 5). Modules (building blocks) can be composed as needed to adapt to current requirements (e.g. from applications and users) and constraints (e.g. from access networks and devices). Further building blocks can be added, exchanged or removed to modify functionality provided by networks, i.e. networks will be able to evolve. In order to support composition of building blocks the interfaces of building blocks must fulfill the following requirements:

- Interfaces must be defined precisely and completely, i.e. it must be possible to decide if two building blocks may interact by comparing their interface specifications;
- Interfaces must be generic, i.e. the technique to interact with a building block must not be specific for each functionality otherwise the composition of building blocks will be limited;
- Interfaces must hide implementation details, i.e. building blocks must expose only the service they provide not a specific implementation (e.g. an algorithm or protocol), otherwise it will be hard to modify functionality if there are components that rely on specific implementations of building blocks.

In this paper a building block interaction model for flexible future network architectures is presented. Based on this interaction model it is possible to define generic, precise, and complete interfaces. Hiding implementation details is also supported, but this still requires appropriate design of building block developers.

In general building blocks can be used to provide any functionality, but here it is considered that building blocks implement (micro-)protocols or related network functionality (e.g. logging or mapping functions). Networking functionality often comes as paired functionality where one reverses the other like encryption and decryption, compression and decompression, and checksum calculation and checksum checking. Since data fields are building block specific and building blocks are hidden from each others it is not feasible to split this paired functionality into different building blocks. In fact, most building blocks will work with a simple model having two ports and two operation modes where one reverses the other.

Section 2 describes the concept of functional composition. Section 3 describes the interaction model and all its details and section 4 shows how this can be used for automatic selection and composition of networks. Related work will be discussed in section 5 and section 6 explains the context of this work and planned future work.

2. Function Encapsulation

New network architectures need flexible ways to introduce, exchange, and combine networking functional-
A common approach to achieve such flexibility is to divide functionality in modular and loosely coupled units called building blocks. Building blocks contain all the processing instructions needed to provide a certain functionality. To enforce loose coupling each instance of a building block can have a thread that executes the processing instructions so that the building block does not depend on external triggers. Also, building blocks can be instantiated to separate states when a building block is used in different contexts. This way building blocks can hide away the inner workings and reduce the interface to what is needed for the provided functionality. That makes building blocks ideal for functional composition and a flexible new Internet architecture.

Building blocks can be defined at different granularities ranging from fine-grained building blocks with small functionality, to building blocks containing complex functionality like TCP/IP. Our approach does not define which granularity must be used, so the granularity is only limited by practical factors. Building blocks with a lot of functionality reduce the flexibility and building blocks with little functionality introduce a noticeable overhead.

3. Ports

A crucial point in maintaining flexibility is loose coupling. Building blocks that are tied to specific implementation details of other building blocks reduce the flexibility to freely combine building blocks and to exchange building blocks with others that provide the same functionality. To enforce loose coupling the described interaction model hides all building block from each others.

Since building block instances still need to communicate with each others to provide a certain functionality it is necessary to provide a way of anonymous communication. But some functionality can only be provided when the building block instance is able to distinguish different communication partners. To solve this problem the concept of named communication endpoints called “ports” is introduced. Building blocks can use the ports to distinguish different kinds of communication partners or different operation modes.

Figure 2 shows a building block, its ports, communication partners, connections, and messages. The following subsections describe these elements in detail. Subsection 3.1 describes the message types that can be used for communication and subsection 3.2 explains connections between ports in detail. Message filters are introduced in 3.3. Subsection 3.4 shows how building block patterns can be wrapped to enable easy reusability and subsection 3.5 describes how to add network and application adapters to the building block graph.

To exemplify building blocks and their interaction a primitive loss detection building block will be used as an example throughout this section. Figure 3 shows the Loss detection building block with its three ports. Ports Up and Down are used to distinguish the direction of data (data is forwarded to the other one) and the operation mode. The port Loss rate is used to provide information about the measured loss rate. The building block is a very primitive implementation of a loss detection that completely neglects packet reordering and thus only serves as an example.
3.1. Message Types

Ports can communicate with each other by sending and receiving message objects. These message objects can be arbitrary objects i.e. simple data types or complex data structures. This allows for very efficient communication as no conversion is necessary and objects can be passed as references. Because ports can only interact when their types are compatible standards for encoding certain kinds of information will be needed. Since objects can be of any type and contain local references, the so called object messages, i.e. messages containing arbitrary objects, cannot be easily serialized and thus are not suitable to be sent over a network link. Communication over a network link is important for different instances of the same building block to communicate and to transport payload data.

![Figure 4. Data message structure](image)

A special kind of message object is the data message which consist of data fields as shown in figure 4. Each data field contains a unique numeric type (field_type) and the field data as a byte array. The field with the reserved field type 0 is the main data field (used for e.g. payload). The other field_types are normally associated with a specific building block that uses this field to store additional data with the payload. The structure of the field data is undefined, which means that fields are only meaningful to building blocks that know their structure. Data messages have a defined encoding based on a TLV structure. This encoded form of a data message can be used by building blocks to apply a transformation (e.g. encryption or checksum calculation) to the data message as a whole and also allows to send data messages over a network link. This way different instances of the same building block can communicate over the network and application data can be transferred.

The loss detection building block shown in figure 3 has three ports. It forwards data messages from the port Up to the port Down and vice versa. During forwarding data messages downwards it will add a field containing a sequence number. The remote partner will use this sequence number to detect missing data messages and remove the field from the message when sending it upwards. The detected loss rate will be calculated and sent via the Loss rate port as a float value. An improved version of this building block adds the locally measured loss rate as a field to data messages that are sent over the network so that the other instance gets feedback of the remotely measured loss rate.

3.2. Connections

With ports building blocks can interact in a loosely coupled way. This allows building blocks to be combined in a flexible way by connecting ports. All connections of all building blocks form a graph that is called building block graph. When the focus lies on the steps that are executed to process the data, the graph can also be seen as a workflow. Connections are always symmetric, i.e. if port A, (port a on building block A) is connected to port B, then B, is also connected to A, even if the messages are sent unidirectional. One port can also be connected with more than one port.

In most cases ports will only have one connection that data sent via this port will be delivered to but some use cases like multiplexing and de-multiplexing need the ability to have multiple connections and to distinguish them. To support this, all ports that are connected to a building block are enumerated using a connection id. The environment interface of a building block allows it to obtain a set of all connection ids on each of its ports. Also, all incoming messages contain the connection id of the sender.

![Figure 5. Congestion control](image)

Figure 5 shows how the loss detection from figure 3 and a new building block for flow control can be combined to provide congestion control functionality. The Loss rate port of the loss detection is connected to the Loss rate port of the flow control. This way the measured loss rate from the loss detection building block is used as an input of the flow control building block. Also, the Down port of the flow control and the Up port of the loss detection building block are connected to
allow bidirectional data message communication. Since the data messages contain the payload data from the application this essentially means a chaining of the two building blocks from an application perspective.

3.3. Filters

When a building block receives a message on one of its ports, the building block decides how to handle the message. It could directly process the message, put it in a message queue, check whether the message must be processed at all or directly forward it to an outgoing port. Since the processing of messages in some building blocks can take a long time, a single-threaded model would reduce performance significantly by waiting most of the time for these building blocks. On the other hand, changing threads at the building block border introduces a synchronization overhead that is unreasonable for most building blocks. Since the sender of a message can not know how long the receiver will process the message the decision must be taken by the receiver.

To solve this problem and still enable maximal flexibility in message processing the concept of message filters is introduced. Each port has a filter that processes incoming messages. The filter is part of the building block that receives the message but is called by the thread that sends the message. This way the receiving building block can decide how to handle the incoming message and whether and how to call the processing thread of the building block.

The filter should process the message quickly and return whether the message was successfully delivered. The filter can reject a message when it cannot be processed at the moment. The message sending interface enables the sender of a message to wait for the receiver to become ready.

Using a filter allows building blocks to implement message buffers (with or without prioritization) on their own. Building blocks can also decide to store a received value in a variable and read it on demand.

In the continued example from figure 5 the flow control building block uses the filter on the port Up to implement a blocking queue to do the actual flow control and the filter on the Loss rate port to store the latest loss rate value in a variable to be read on demand. Since the loss detection building block and the upwards direction of the flow control are able to process messages very quickly they do not use a separate thread for this functionality, thus the filters directly execute the respective code to improve performance.

3.4. Wrapped Building Block Graphs

Subsection 3.2 explained how building blocks can be connected to form a graph. Some combinations of building blocks will be used frequently, forming a so called pattern. To encourage pattern development and usage, building block graphs can be represented (wrapped) as building blocks. To allow communication, the wrapped graph needs ports. This can be easily accomplished by mapping ports of internal building blocks to external ports of the wrapped building block. Unmapped internal ports stay hidden from the outside which allows to simplify the outer interface and also be used for access control.

3.5. Usage for Networking

To use this interaction model for networking, external interfaces to the application and the network must be added. To allow easier communication with the building blocks those external components are also represented as building blocks.

The application interface is simply a building block adapter that allows the application to define its ports and their types. This will enable novel applications to utilize the full potential of this interaction model. Functionality that currently resides in the application scope like multimedia encoding and decoding, security layers, and peer-to-peer networks can be moved to the network scope allowing a higher degree of flexibility and software reusability. New data processing paradigms where the data source, all processing steps and the data sink are realized as building blocks and the application controls...
the data flow can be developed. For legacy applications, adapters can be implemented providing interfaces compatible to the socket interface used for UDP/IP and TCP/IP protocols today.

For the network interface multiple realizations are possible depending on the abstraction level the framework uses as a networking basis. On a very low level of abstraction, each network interface could be represented as one building block. This leaves it to the building block graph how to choose and aggregate the network interfaces. Also, routing algorithms would have to be implemented as building blocks. On the other side, the framework could also be built upon a network abstraction that already provides routing and network interface management. This way the framework would be used only in an end-to-end communication model.

4. Automatic Selection & Composition

Selection & composition is the task of finding a building block graph to fulfill a certain need. Automatically creating a workflow on demand i.e. connection setup, offers the highest degree of flexibility and workflow optimization. Doing the composition automatically, requires that the building blocks must have descriptions of what they do and how they can be combined. These descriptions are used to select building blocks and construct a building block graph. The graph that is built must fulfill certain requirements that are usually given by the application, the system administrator or the network operator. If more than one graph matches these requirements the best graph should be selected where the term “best” is normally described in terms of performance, reliability, security, and costs as an optimization function.

Solving the problem of selection & composition in general is beyond the scope of this paper, but some prerequisites needed to implement a composition algorithm will be described. Any algorithm that tries to solve the composition problem needs information about which building block connections and graphs are valid and which attributes that combination has. This information must be provided in a way that can be processed automatically and it must not restrict building block interaction in any way. The easiest approach would be to list all building blocks that are compatible together with the attributes of the combination. For some usage domains this is feasible but not for a future network architecture. A crucial point of such a future network architecture is flexibility, not only the flexibility in how to combine building blocks but also the flexibility to add, exchange, and remove building blocks. This flexibility would be destroyed by a global database where all dependencies and combination possibilities of building blocks are to be stored. This means that dependencies and combination possibilities must be expressed in an implicit way that does not require any changes to other building blocks when one building block is added, changed or removed.

4.1. Simple Port Matching

The most obvious information on how to combine and how to not combine building blocks is syntactical compatibility. This means that ports that are connected must use compatible message types. To check this, all ports must provide a list of message types they can receive and/or send. Using this list a simple compatibility check can be done: Ports must be able to receive all message types that can be sent by connected ports. Using this check a lot of connections can be ruled out and for some exotic message type the list of compatible ports can be a valuable hint for composition.

The number and kind of connections can also be used for checks. Ports can define if connections on a port are optional or mandatory and may set limits on the number of connections. Ports can also require that a message type they can receive must be send by at least one or by all connected ports.

Finally the graph theory yields some very basic tests to check the validity of the graph as a whole. Of course no building block must be completely unconnected even if all of its ports allow to be unconnected. Also, the building block graph must contain at least one data source and one data sink (e.g. network and application adapters) to be of any use. Finally the graph must be strongly connected or at least must contain a path from all data sources to at least one data sink.

In the continued example the loss detection and the flow control building blocks require that both the Up and Down ports are connected at least once and the flow control building block requires that the Loss rate port is connected to exactly one other port that is actually able to send the loss rate object that the flow control needs.

4.2. Advanced Port Matching

The simple syntactic tests allow to check whether port connections are possible and graphs are syntactically valid but it can not determine whether combinations make sense and what the attributes of combinations are. To test for attributes a more expressive language must be used that rely on syntactical correctness.

To allow the calculation of attributes provided by building block or combinations of building blocks it is necessary that building blocks define their attributes and...
how they influence attributes of other building blocks. In the described model all ports have attributes that have numeric values. The attributes are not restricted to a certain predefined set to allow maximal flexibility. Attributes can be used by giving their name and comparing the value. There need to be standards on the naming of attributes and their meaning. This can be accomplished by having a standards authority defining it or by having an ontology that allows to convert between different attributes with similar meaning.

There are many ways in which attributes might be expressed ranging from simple descriptions to complex description languages. Maximal flexibility can be achieved when ports provide their attributes to themselves. Thus it is required that ports provide methods to query their attributes. These methods can either reply with the attribute value if it is a fixed value or they can calculate the value dynamically. These dynamic calculations can also query attributes from other connected ports of the same building blocks. If a value cannot be calculated the method can also return an undefined value. In the most cases this means that the connection structure is not valid (e.g. the port is unconnected but must have at least one connection to calculate the value).

All building blocks can set limits on the attributes (required attributes) of connected ports. When the limits are violated a connection cannot be established. The limits can be expressed with a simple first order logic using attributes of all connected ports of a building block.

In the continued example the loss detection building block adds a data field containing a sequence number. Assuming the field adds 8 bytes to the data this obviously reduces the MTU by 8. The loss detection building block expresses this by setting a limit on the MTU for its down port: \( \forall c \in \text{con}(\text{down}) : c.MTU \geq 8 \) and by relaying the request for the MTU attribute on the up port as follows: \( \text{up}.MTU := \min(c.MTU | x \in \text{con}(\text{down})) - 8 \). Since the data stream between up and down is not modified in any other way all other attributes can be inherited from the connection on the port down. This will be the most common case for building blocks.

When used for networking most of the fixed attribute values will be provided by network or application adapters. Requirements that come from the user, the administrator, the network operator or the underlying networking technology can be expressed as requirements of the application and network adapters. A lot of attributes, especially network attributes like latency and error rates are unknown in advance and thus cannot be used directly. In these cases other values can be used depending on the attribute. For some attributes that cannot be predicted, past data series allow to limit the value to a range that can either be expressed as minimum and maximum or as average and standard deviation. For other attributes like security, trust and anonymity numerical values cannot easily be obtained. In these cases a mean opinion score based on subjective rating might be used.

4.3. Graph Generation

Attributes can be used for the composition to check whether connections are correct using semantic properties. A composition algorithm can compare the provided attributes with the required attributes on all connections to check graphs or even single connections for validity or to find candidate building blocks for composition.

The whole task of composition is a very complex optimization problem that is in general NP-complete. It should be possible to reduce the Knapsack or the 3-SAT problem onto the selection and composition problem using the message data types and connection count restrictions to limit possible combinations of building blocks. A complete proof of the NP-completeness is part of the planned future work.

To help the composition algorithm to predict the influence of a building block on a specific attribute the building block can express this influence in a way that the composition algorithm can process. For most algorithms it would be a great improvement to simply know which attributes a building block influences at all and (if possible) whether it decreases or increases the value.

5. Related Work

The goal of automatic protocol composition is not new and has often been described e.g. by [12], [11], [17], and [1]. Most of these approaches either limit the communication to linear data processing or limit the functionality of building blocks by defining a fixed set of building block categories. Other approaches like RNA [16], Silo [5] and especially ANA [9] offer a more flexible building block interaction model but they can not provide a fully automatic composition. This is an important difference since automatic composition is not an additional feature of an architecture that can easily be added later, instead it heavily influences and in parts defines the architecture. Another crucial feature of a future network architecture is evolvability. Automatic composition solves this problem by allowing newly introduced building blocks to be selected automatically.
The Specification and Description Language (SDL) [3] gives a good framework for module interaction that hugely influenced the described interaction model. [6] describes how a framework based on SDL can be used for modular design and composition of micro-protocols. SDL however does not address the problem of automatic protocol composition which has high requirements on the interaction model and the building block descriptions.

Approaches like the click modular router[10] and the x-Kernel [8] show that modular network stack implementations can be as efficient as less modular implementations found in most operating systems.

6. Future Work

The described model has been developed in the context of German-Lab [7] as part of a project for automatic and dynamic protocol composition for a future Internet [11], [12], [15]. A part of the project is to develop a prototype framework that allows to match application requirements with network attributes and constraints by building up a graph of interconnected protocol building blocks. This framework is currently being developed and already supports most parts of the described building block interaction model. Some important networking functionality like routing and retransmission were successfully implemented using this interaction model. More functionality will be implemented as building blocks to be able to test more complex workflows. Once the framework is feature-complete and it has been shown that this interaction model supports all requirements for implementing networking functionality, the current prototype implementation will be optimized for performance, which allows to test the overhead introduced by the interaction model and its feasibility for networking.

As another major part of the project, algorithms for automatic composition will be designed. Currently an approach using evolutionary algorithms is being investigated. Other approaches include directory based, pattern based and template based selection and composition algorithms.

Different approaches on combining those algorithms will be examined. Time efficient algorithms could be queried first and if the result is not good enough the more time-consuming algorithms could be queried. Another approach is that all algorithms are queried in parallel with a certain time constraint. More complex approaches using caching, pre-calculation and a feedback loop combining the evolutionary algorithm with all other algorithms will also be explored.

References


