A FRAMEWORK FOR RE-CONFIGURABLE FUNCTIONS OF A MULTI-MODE PROTOCOL LAYER

Marc Schinnenburg, Fabian Debus, Arif Otyakmaz, Lars Berlemann, Ralf Pabst
Communication Networks, RWTH Aachen University
{msg | fds | aoz | ber | pab}@comnets.rwth-aachen.de

ABSTRACT

This paper presents a framework for building re-configurable protocol stacks. A high degree of re-configurability is achieved through composing complex behavior of a communication system using Functional Units. A uniform interface allows these units to be connected to form a Functional Unit Network. The requirements and the resulting interfaces for such units are subject to this work.

1. INTRODUCTION

The increasing complexity in today's software systems has led to a number of new methods in software development in the last years. The goal of modern software design is to create systems consisting of small units. Every unit should have one cohesive responsibility, provided through a preferably slim and simple interface. The avoidance of tight coupling and the focus on testability leads to units that are easier to build, to test and to maintain. Thus the development and maintenance costs are reduced. Further, the quality of software is increased. As a second benefit reusability of units is improved if dependencies between units can be reduced.

Ubiquitous radio access at high data rates and low delays is the customer's expectation at next generation communication systems. To meet this expectation the protocols of future communication systems need to efficiently exploit the available spectrum in a dynamic way. The need to achieve optimal performance in a variety of different environments (e.g., indoor/outdoor) will force devices and their protocols to adapt themselves to the current situation by using different modes, i.e. Radio Access Technologies (RAT). The integrated project Wireless World Initiative New Radio (WINNER) (funded under the 6th Framework research funding Program (FP6) of the Commission of the European Union) focuses on different aspects of such an adaptive and flexible air interface [1].

To achieve the highest degree of adaptivity, the ideal protocol stack should be completely re-configurable. The complexity inherent to such systems raises the same problems as found in software development. This paper therefore tackles the problem of re-configurability using similar methods as described above. A protocol stack consisting of small and independent units, here called *Functional Units* (FUs), with cohesive responsibility is therefore one of the key technologies for next generation radio networks.

This paper will introduce a framework for re-configurable protocol functions of a multi-mode protocol layer. Thereby, our focus is on the Data Link Layer (DLL). The framework can be seen as a complementary technology to Software Defined Radios (SDR) for higher layers [2].

Section 2 discusses the proposed interface of FUs. Especially data handling of and interconnecting between FUs to form a Functional Unit Network (FUN) is described.

Section 3 describes different kinds of dependencies between FUs. Since dependencies between units introduce tighter coupling, the section describes situations where dependencies are necessary and gives guidelines how to cope with them.

In section 4 a technique is presented used to instantiate FUs dynamically for different flows.

2. FUNCTIONAL UNITS

As discussed in [3] DLLs of protocol stacks of wireless communication systems in general comprise among others the following set of functions: Automatic Repeat request (ARQ), Segmentation and Reassembly (SAR), scheduling, multiplexing and buffering. Having identified such a set of FUs a necessity for common interfaces arises. How should FUs be organised to support such a wide range of different tasks? How can these units be connected in a generic way to support the configuration of larger systems based on such units only?

To answer these questions, we start analyzing the most fundamental requirements and describe interfaces that allow these requirements to be met. We will describe applications of the defined interfaces and how the FUs can be used to compose complex systems based on these interfaces. In cases where the interfaces are still too weak,

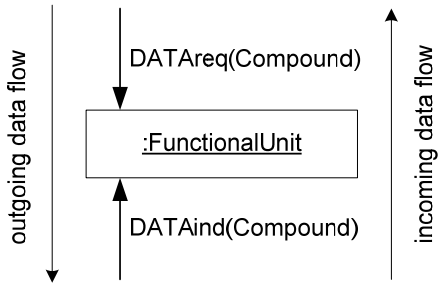

Figure 1. Basic compound handling interface.

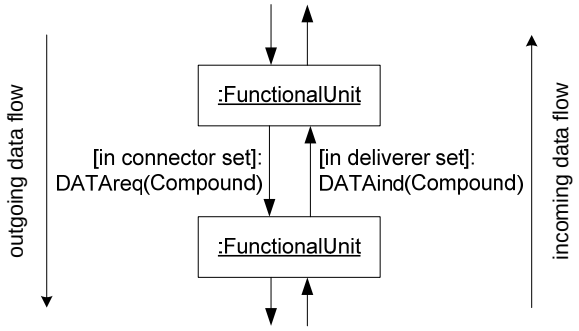

Figure 2. FU connections for data handling.

we will formulate new requirements and present extensions to the interface that meet the newly identified requirements.

2.1 Data Handling

The most fundamental requirement for FUs is the ability to handle data. In the following we will denote a basic data unit that is transmitted between FUs a *compound*. For now a compound can be seen as some chunk of data of variable size.

FUs as part of a protocol stack may receive compounds for processing before and after such a compound has been transmitted over the air-interface. The first case is called outgoing data flow, while the latter case is referred to as incoming data flow. The interface for handling compounds has to provide services for accepting data in both directions, incoming and outgoing. The interface must further enable the FU to distinguish between compounds of both flows. To support that, it is advisable to choose two different methods: `DATAreq(Compound)` for compounds in the outgoing flow and `DATAind(Compound)` for compounds in the incoming flow as depicted in figure 1.

2.2 Functional Unit Networks

The methods `DATAreq` and `DATAind` are called by other FUs to propagate compounds through an FU Network (FUN). Every FU contains two sets of references to other FUs: The connector set and the deliverer set. FUs call the `DATAreq` method of other FUs in their connector set to pass on compounds in the outgoing flow and call `DATAind` of FUs in their deliverer set to pass on compounds in the incoming flow (see figure 2).

The FUs can be connected to multiple units in both directions to support multiplexing and scheduling, realized by choosing different strategies to select a unit for compound delivery.

An FUN can now be constructed by choosing FUs from a toolbox of FUs and connecting them, defining their connector and deliverer sets.

It is possible to further identify a set of units as sink for outgoing flows: Compounds delivered to these units are leaving the FUN for delivery to lower layers. Another set of units may be identified as sink for incoming flows: Compounds delivered to these units are leaving the FUN for delivery to higher layers. Consequently, an FUN can be seen as a bi-directional data processing network. Input to the network is injected using either the `DATAind` or `DATAreq` method of any of the FUs. The output of the network is measurable at the sink units.

2.3 Commands

Whenever a compound arrives in an FU, the FU gains control over the compound and can realize different behaviors by handling the compound accordingly. It may choose to mutate or drop the data unit, buffer it, forward it to other FUs or inject new data units into the FUN.

A large class of FUs is characterized by enriching the compound, adding control information on outgoing compounds and reinterpreting the added information on incoming compounds. Usually these FUs provide a transparent connection to other FUs above. An ARQ protocol for example adds sequence numbers as control information to the compounds of the outgoing flow. It creates and injects compounds as acknowledgements in order to reply to compounds of the incoming flow. The ARQ instance in the peer FUN reinterprets the added control information, delivers valid information frames to some FU in the deliverer set and consumes dedicated compounds containing acknowledgements. The control information added by FUs is called *command*. The command can have different characteristics for different purposes, like an information command or an acknowledgement command for the ARQ.

The ARQ in our example is completely invisible to the FUs above. Even underlying FUs do not need to have

knowledge about the control information added by ARQ implementations. The only FU that is required to be able to handle the ARQ command is the peer unit of the ARQ.

Sometimes, however, FUs add commands that are important to other FUs either in the peer FUN or within the same FUN. Connection identifiers may serve as an example for such information. An FU that decides to which hop to send a compound would require being able to retrieve the final destination which is part of a higher level routing command. This leads to the requirement of having a possibility to access commands added by other FUs.

Note that FUs cannot simply reinterpret control information added by other units to the compounds' data. FUs have no information about the layout of the FUN and therefore also have no information about the layout of the combined control information within the compound. There might be an arbitrary number of FUs in between the unit that added the control information and the unit that intends to access it. Additionally, the data might have been heavily modified by other FUs in between.

The solution is to attach a set of commands to each compound. Since an FUN has a known number of connected FUs, there is a known set of potential commands. The set containing all the commands of every FU within an FUN is called *command pool*. Now the union of a data unit and a command pool is denoted a compound.

Initially all commands within the command pool of a compound are inactive. The data attached to a compound is set to the data unit delivered by higher layers for transmission. A data unit is initially empty for compounds being created/injected in the FUN (e.g., ARQ acknowledgements). Parts of the command pool get activated during the propagation of a compound through the FUN, where every FU activates its command when in control. At the same time FUs can mutate the data. A set of activated commands ordered by their time of activation is named a *command sequence*. A FUN is required to be free of cycles to assure that commands do not get activated more than once. Hence, an unambiguous command sequence must exist in which single commands may be retrieved.

2.3.1 Relaying of Compounds

Note that the activation of commands from a non-extendable command pool introduces the problem of implementing relaying FUs [4]. Having a single point of activation implies that compounds may not cross the borders between the two networks from incoming to outgoing data flows. No FU may forward a compound using `DATAreq`, when received via `DATAind`. This is a direct consequence from the activation of commands.

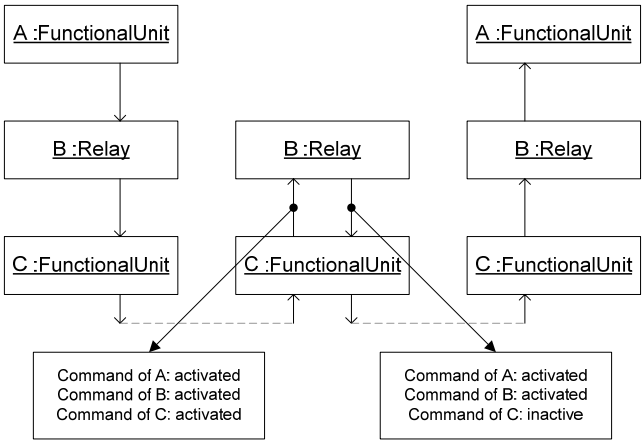

Figure 3. Partial copy of a command pool for relaying.

Otherwise, it would be possible for the compound to be delivered to a FU that already activated its command.

To implement relaying, the relaying FU has to inject a copy of the received compound. It has only those commands activated and copied, that are in the command sequence before the relaying unit. The rest of the commands will stay inactivated. See figure 3 for an overview of the activation status of commands within a compound before and after being processed by a relaying FU.

2.3.2 Coding of Commands

Besides commands being accessible by other FUs, delaying the coding of commands as part of the data has another advantage: Often information in communication protocols is not transmitted explicitly as a stream of bits, but implicitly through the choice of radio resources element like time, frequency, space or code. In a Time Division Multiple Access (TDMA) system for example with fixed slot reservations for connections, it would be useless to explicitly transmit connection identifiers. Nevertheless the information is indirectly transmitted through the choice of a specific slot. Such a slot must be chosen at some point of time based on the connection identity. A command provided by a connection aware FU may contain the connection identifier. But the choice how to transmit the connection identity is delayed, and the outcome may be different depending on the system.

As we have shown, attributes of commands serve different purposes. Some are meant to be transmitted, while other attributes are only meaningful within an FUN and are not meant to be sent to the peer FUN. In order to be explicit about the purpose of a command attribute, we divide the attributes of commands in two distinct sets: The local on the one hand and the peer set on the other hand.

2.4 Flow Control

In practice every FU has only a limited capacity to store compounds and often FUs do not need to store compounds at all to accomplish their task (e.g. forward error correction units). The physical layer on the other hand introduces a bottleneck, limiting the amount of information transmitted and thus the rate at which compounds must be handled.

Without any flow control mechanism within an FUN, compounds could leave the FUN with much higher rates than the physical layer could possibly handle. This would result into a dropping of compounds in the physical layer. Buffering between the layers is not an adequate choice either, since the delay between processing the compound in the FUN and data transmission would increase. The increase of delay has several drawbacks. First, timeout mechanisms would not work as expected. Retransmission timers could lead to retransmission of compounds although the last transmission of these compounds has not even been started. Such compounds would be added to the buffer several times, leading again to increasing delays.

Another drawback is that decisions of FUs based on feedback of the physical layer would loose accuracy; and gathered information would be outdated, when the consequences of the decisions would finally manifest.

Thus the need for an intra layer flow control arises. FUs must have the ability to prevent other units from delivering compounds to them, when they decide not to accept additional compounds.

There are different reasons for an FU to decide not to accept compounds. All these reasons are direct consequences of the limited resources of the physical layer and thus only apply for outgoing flows. Resources in higher layers are usually not a bottleneck for incoming flows.

2.4.1 The Intra Node Flow Control Protocol

To implement flow control between FUs, two methods are necessary:

- `isAccepting(Compound)` → `Boolean`
- `wakeup()`

Before an FU is allowed to deliver a compound to another FU using `DATAreq`, it has to ask for permission using the `isAccepting` method. If the response is negative, it may not send a compound to the questioned unit.

It is essential that the FU asks for permission for a concrete compound, since the answer may depend on the content of the compound. An FU may be willing to accept compounds of some type, refusing to accept others. E.g. a concatenation unit could still be able to use a small compound for concatenation, not having capacities left for concatenating a larger one.

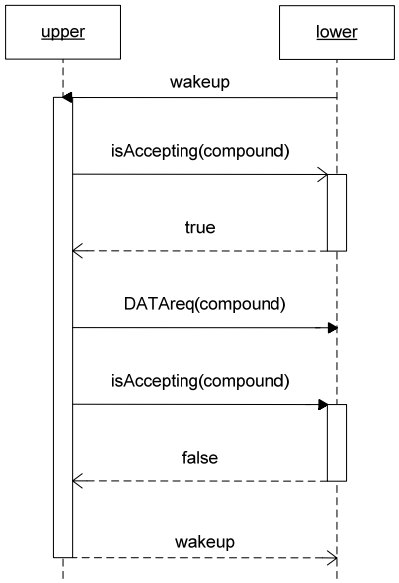

Figure 4. Intra node flow cotntrol.

When an FU can not deliver further compounds, it cannot proceed and thus ceases operation until it is triggered again. Such triggers can come from new compounds being delivered, timers expiring, but it may as well happen that an FU in its connector set changes its state to accept compounds again.

The method used for informing other FUs that they might succeed in sending a compound is `wakeup`. A set of FUs that have to be notified when an FU is willing to accept new compounds is called receptor set. The receptor set of an FU "A" contains exactly those FUs that have FU "A" in their connector set.

Figure 4 shows an example of two FUs transmitting compounds with respect to intra node flow control.

Besides the rules above, there are some rules which must be followed by every FU to conform to the flow control protocol: Two consecutive calls to `isAccepting` with the same compound and no `DATAreq` calls in between have to yield the same result. The following rules provide a way how to accomplish this stability.

1. An FU may only base its decision whether to accept a compound or not on its internal state, on the content of the compound and on the outcome of `isAccepting` calls to FUs in its connector set.
2. An FU may not mutate the compound during a call to its `isAccepting` method.
3. An FU may not change state during a call to its `isAccepting` method.
4. An FU may not mutate the compound between the `isAccepting` call to an FU of its connector set and

the delivery of that unit to the questioned unit.
Since an FU may base its decision whether to accept a compound on the content of the compound, it is illegal for the questioner to mutate the compound, potentially invalidating the promise of the questioned unit to accept the compound.

5. If an FU delegates the `isAccepting` call to an FU in its connector set, it has to deliver the compound to exactly this FU.
This leads to arbitrary long chains of promises to accept a compound.

Note that rule 4 has a very strong impact on the implementation of FUs that have no internal capacity since they may not mutate the compound. A weaker version of rule 4 would allow the modification of the compound given the knowledge that no FU in the chain of promises bases its decision on the changes made to the compound. But this condition is very difficult to guarantee.

It is important to note that the order in which an FU awakens units in its receptor set significantly changes behaviour of the propagation of compounds. Units being called first, have a higher chance of being able to deliver compounds. A fair strategy would wakeup units using a round robin algorithm, starting with another unit every time. For three units A, B, C in the receptor set, the wakeup sequences would be: ABC, BCA, CAB, ABC, ... If the units in the receptor set have clear priorities, a single wakeup sequence with the units ordered by descending priorities would suffice. The wakeup strategy is part of the receptor aspect of each FU.

2.4.2 Inter Node Flow Control

In the case of a bottleneck in higher layers (e.g. streaming applications that accept data with a lower bit rate than the physical layer provides), protocols usually provide inter node flow control mechanisms between the communicating nodes. In fact, this again is based on the flow control of the outgoing flows, but this time between FUs of the peer node. Protocol functions must exist that inform the peer node producing the data to slow down, which results in intra node flow control of the producing node to limit the amount of data generated.

2.5 Five Aspects of a Functional Unit

To summarize the discussion above, we distinguish five aspects of an FU:

1. **Compound Handler**
Implement the handling of compounds of an FU including intra FUN flow control. The methods provided are
   - `DATAind(Compound)`,
   - `DATAreq(Compound)`,
   - `wakeup()` and
   - `isAccepting(Compound)` → `Boolean`.

   Handling of compounds includes mutation, dropping, injection and forwarding. Activation and initialization of commands is considered as mutation.
2. **Command Type Specifier**
Define the type of command provided by the FU. This type will be used to create an initial command pool and to verify unit dependencies as will be discussed in section 3.
3. **Connector**
Hold the set of FUs that compounds will be delivered to in the outgoing direction.
Define a strategy to select the appropriate FU for a given compound.
4. **Receptor**
Hold the set of FUs in which the FU itself is in the connector set.
Define a strategy to wake up FUs.
5. **Deliverer**
Hold the set of FUs that compounds will be delivered to in the incoming direction.
Define a strategy to select the appropriate FU for a given compound.

3. UNIT DEPENDENCIES

Ideally an FUN would consist of FUs without any inter unit dependencies. But that is not an option for building real world protocol stacks. Knowing what kinds of unit dependencies exist, what they imply and when to accept them is essential for the design of FUs and FUN.

We distinguish between two different kinds of unit dependencies: Direct and deferred coupling. Direct coupling is a dependency on the interface of an FU; deferred coupling is the dependency on the command of another FU. When FU "A" depends on the interface or the command of FU "B" we say that "B" is a *friend* of "A".
Direct dependencies arise for example for

- multiplexing FUs that need assistance of their friends below them to decide where to deliver compounds.
- horizontal collaboration; FUs responsible for realising control plane functionality, receiving supervisory frames from a peer node and configuring their friends to modify the user data plane accordingly.
- vertical collaboration; layered protocol functions that must work close together but change behaviour in different places of a protocol stack.

To avoid tight coupling, those dependencies should rely on the most general interface possible [6]. The goal should be to make FUs depend on families of units sharing a common interface, than to depend on a single

type of FU. This allows friends to be exchanged without modifying the dependent FU.

Since the exact layout of an FUN is unknown to the FUs, the FUN provides services for the FUs to find their friends by name and desired interface. Making the names of the friends a configuration option to dependent FUs results in a high degree of flexibility. Friends can be retrieved once after re-configuration of the FUN.

To retrieve a command from a command pool, the retrieving FU does not need to rely directly on an interface of the command's provider. It relies on the command's provider to be present in the FUN and on the type of command the provider specified.

4. FLOW SEPARATION

FUs as described comprise state and behavior. An SAR unit for example needs to store segments of compounds to be able to apply segmentation and reassembly. An FUN therefore needs different instances of a SAR unit for different peer FUNs as depicted in figure 5.

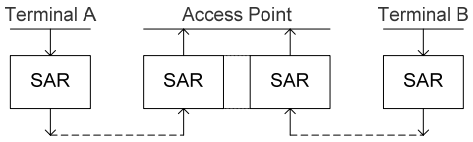

Figure 5. FUs of different flows.

The way of creating FUs within an FUN dynamically depending on the flow is by using a *flow separator* (see figure 6). The flow separator itself is an FU, configured by a key to distinguish flows and a strategy to create FU instances [7]. Compound handling including flow control is delegated to the according instance by the flow separator.

5. CONCLUSIONS

A framework for building re-configurable protocol stacks out of Functional Units is presented. Functional Units are connected using a uniform interface to form arbitrary Functional Unit Networks in order to build complex protocols. An interface, divided into the following five aspects, has been defined: Compound Handler, Command Type Specifier, Connector, Receptor and Deliverer. It is stated that Functional Units should ideally be independent from other Functional Units. However, this is not always possible. A solution for keeping dependencies at a minimum level is presented. Finally, the problem of flow separation is discussed briefly.

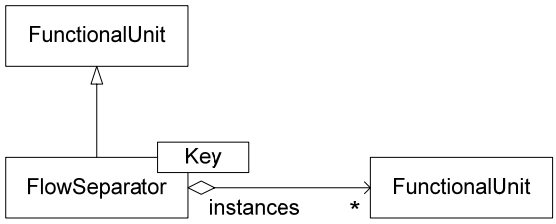

Figure 6. FlowSeparator in a FUN.

Furthermore, the presented framework opens up the potential for accelerated protocol stack development and performance evaluation.

Future work includes further investigations on the identification and implementation of reusable Functional Units conforming to the presented interface.

6. ACKNOWLEDGMENTS

The authors would like to thank Prof. Dr.-Ing. B. Walke for the fruitful discussions and comments to the content of this paper. The presented work has partly been funded by the European Commission in the FP6 IST-Project WINNER (IST-2003-507581).

7. REFERENCES

[1] Berlemann, L., Pabst, R., Schinnenburg, M. and Walke, B. H., "A Flexible Protocol Stack for Multi-Mode Convergence in a Relay-base Wireless Network Architecture", in 16th IEEE Conference on Personal, Indoor and Mobile Radio Communications, PIMRC 2005, Berlin Germany, September 2005

[2] J. Mitola, "The Software Radio Architecture," in IEEE Communications Magazine, vol. 33, no. 5, May 1995, pp. 26-38

[3] Berlemann, L., Pabst, R., Walke, B. H., "Multimode Communications Protocols Enabling Reconfigurable Radios", in EURASIP Journal on Wireless Communications and Networking 2005:3, pp. 390-400

[4] R. Pabst et al., "Relay-Based Deployment Concepts for Wireless and Mobile Broadband Radio," in IEEE Communications Magazine, vol. 42, no. 9, Sept. 2004, pp. 80-89.

[5] Berlemann, L. and Cassaigne, A. and Pabst, R. and Walke, B. "Modular Link Layer Functions of a Generic Protocol Stack for Future Wireless Networks," SDRforum'04, Phoenix USA, Nov. 2004

[6] Sutter, H. and Alexandrescu, A., "C++ Coding Standards", Addison-Wesley, 2005

[7] Gamma, E. et al, "Design Patterns", Addison-Wesley, 1995