Domain Concepts for Communication Protocols

Juha Pärssinen
juha.parssinen@vtt.fi

In this paper domain concepts for communication protocols are introduced. These concepts were found from several different protocols, and protocol implementation frameworks and environments. In this paper there are introduced first two different protocol implementation environment, then the domain concepts, and finally how those concepts can be mapped to implementation environments. These concepts were used to support protocol design, implementation, documentation, and also as a learning aid for newcomers to this domain. They were also used when a guideline for standardization was developed in ETSI. The main reason for this work was (and still is) the insufficience of the UML for protocol engineering domain. The limitations of the UML in this domain are introduced briefly in this paper.

1 Introduction

Nowadays graphical modeling, e.g. the Unified Modeling Language (UML) [18][20], and code-generation from graphical models are often used in software engineering to master complex systems and to improve productivity. In protocol engineering domain their usage is not so common.

Graphical modeling of protocol systems and code-generation from those models using traditional box-and-line graphical modeling based on Open Systems Interconnection (OSI) reference model, as explained in this paper in chapter 2 and [8][10], or the UML [18][20] are not possible:

- The OSI based traditional modeling is concentrating only on the external behavior of the protocol systems, not implementation issues, and does not include any concepts required for the code-generation.
- The OSI based traditional modeling is bound tightly to a layered structure, but some protocol systems, e.g. the GSM [12], has layers located beside the rest of the stack.
- The OSI based traditional modeling does not model subconnections, which can be found from some modern protocols, e.g. the WAP +[19].
- The UML1[18] or the UML2[20] have no formal semantics, and it is impossible to use any of them to make unambiguous and precise description of all aspects of protocol systems.

These are some of the reasons why domain concepts for communication protocols presented in the chapter 3 were developed. Another import reason is current protocol engineering frameworks, which introduce their own concepts for communication protocols. Domain concepts introduced in this paper were developed as a tool or an implementation language independent teaching aid for newcomers, and communication aid between protocol engineers.

This paper introduces briefly what kind of concepts can be found from communication protocols, what kind of relations there are between those concepts, and how those concepts can be described. More information about them can found from [1][2][3][4][6].

Chapter 2 introduces communication protocols, including specification and implementation issues, and two different protocol engineering frameworks. Chapter 3 introduces domain concepts for communication protocols, and mapping between these concepts and two different protocol engineering framework. Chapter 4 contains conclusions and future work. Chapter 5 list references used in this work.

2 Communication protocols

2.1 Communication Protocol Specification

For two parties to successfully communicate, it is necessary that they speak the same language. What is communicated, how it is communicated, and when it is communicated must conform to some mutually acceptable set of conventions between the parties involved. The set of conventions is referred to as a *communication protocol*, which may be defined as a set of rules governing the exchange of data between parties. This paper uses the term *protocol* as a shorter form of the term communication protocol.

A communication system typically contains more than just one protocol. Protocols of one system are typically specified adjacent layers, e.g. Open Systems Interconnection (OSI) reference model [10] (see Figure 1a.). Layers are built on top of each other, and higher layer uses the interface provided by the lower layer [8]. The use of an interface hides the lower layer from layers above it. This modularity and encapsulation of layers, or *entities,* have been in use years before the advent of the object-oriented programming.

Interface of a protocol layer is traditionally specified using services. A *service* represents a set of functions offered to a user by a provider. The service is made available through *service access points* (SAP) (see Figure 1b). From the user perspective, all of the qualities of the service are completely defined by the interface at the SAP. [8]

The service provider entity itself might be composed of smaller entities, which in turn use the service below as shown in Figure 1b. This figure also explains how two entities combine

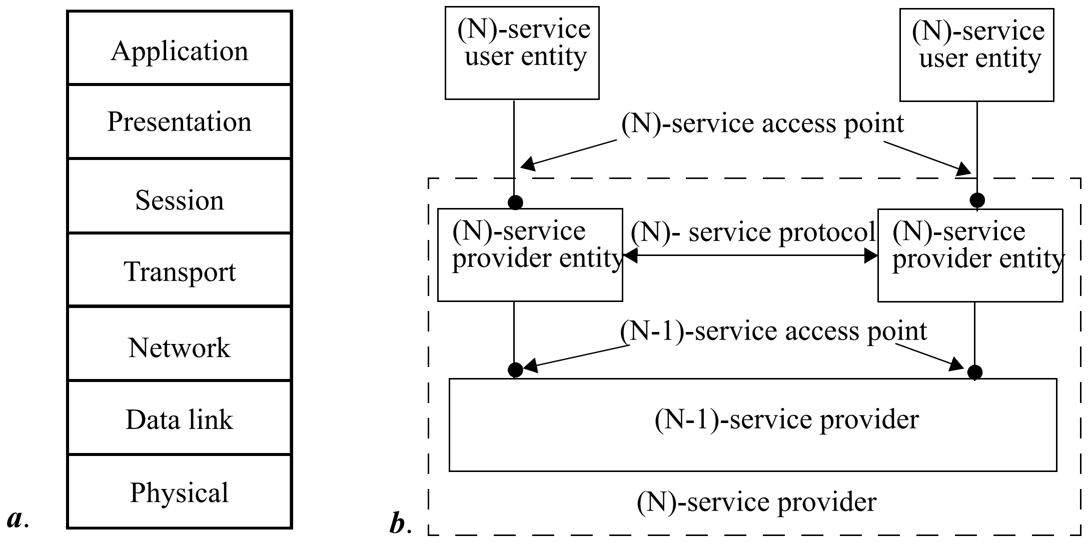


Figure 1. OSI Reference model [10] (a), and service layering [8] (b)

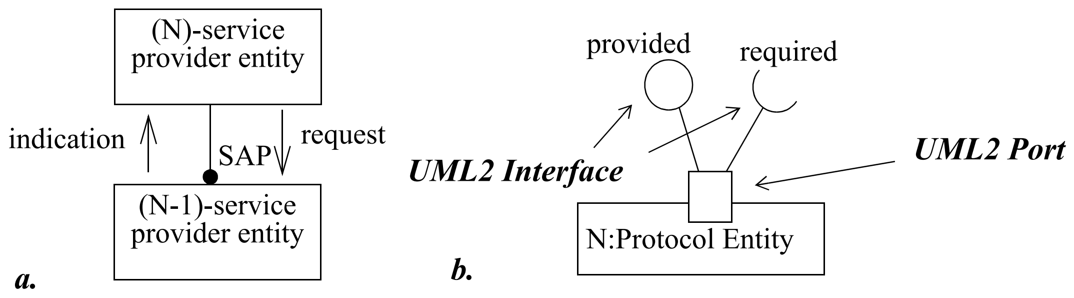


Figure 2. Comparison between the service interface at the Service Access Point (SAP) (a.), and the Interface and the Port of the UML2 (b.).

to offer a new service: they use the service from below and they communicate using a protocol. A relatively simple service may be augmented to offer more powerful services at the layer immediately above. This process may continue until the desired level of abstraction is reached. For example, the logical structure of the OSI Reference Model [10] is made up of seven protocol layers.
It is important to notice that there is a terminology difference between the protocol engineering domain and the object-oriented modeling: the word "interface" is used in both of them, but it has different meanings. In the traditional protocol modeling a protocol's service interface (defined for service primitives at the Service Access Point, SAP) is bidirectional, as shown in Figure 2a. However, in the object-oriented modeling the interface is unidirectional (i.e. incoming only). Figure 2b. shows an example of use of UML2 [20] when a protocol's service interface is modeled. As shown in this figure, UML2 Port is quite similar to the interface in protocol engineering domain, i.e UML2 Port can include both provided (UML) interface for incoming messages and required (UML) interface for out-going messages.

2.2 Communication Protocol Implementation

A communication protocol implementation process can be facilitated if some of existing protocol engineering framework is used [13][14][15][16][17]. In next section 2.2.1 two of such frameworks, Conduits+[13] and SDL[17] are introduced briefly. However, it is no matter if protocol framework is used or not: the architecture of a protocol entity implementation will contain several common parts. These are shown in Figure 3:

- Codecs, coding and decoding functionality, for primitives to communicate to different entities, e.g the peer entity, the upper and the lower layer entities.
- Codecs for the protocol messages

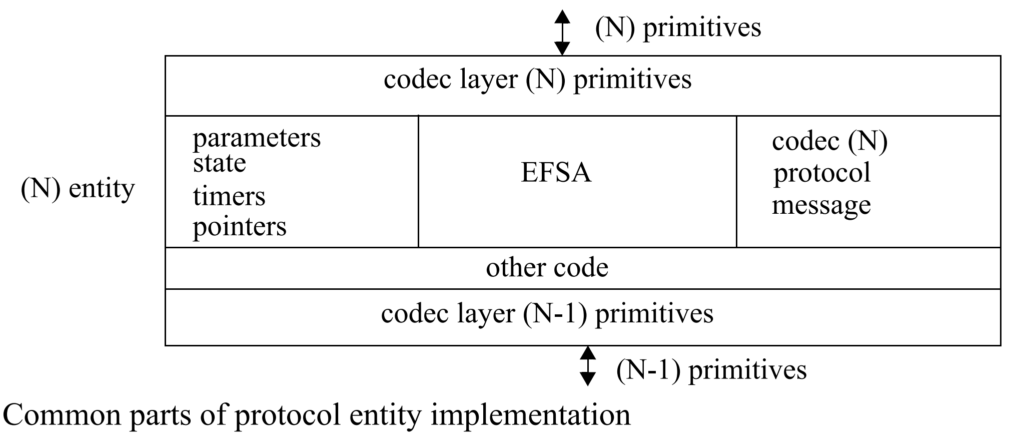


Figure 3. Common parts of protocol entity implementation

- State machine (Extended Finite State Automaton, EFSA). This usually implements those protocol functions that are needed for the protocol entity. Protocol functions can be e.g. segmentation and reassembly, encapsulation, connection control, ordered delivery, flow control, error control, synchronization, addressing, multiplexing, and transmission services [7].
- Storage space for protocol entity's parameters, state, timers, pointers etc.

2.2.1 Protocol Engineering Frameworks

In this section two very different protocol engineering frameworks and their concepts are introduced briefly. Small parts of TCP/IP stack [9] design are shown as examples. How domain concepts for communication protocols are mapped to these framework concepts is shown in the section 3.1. Detailed explanations of this mapping and can found from [4].

Conduits+. The Conduits+ [13] is an object-oriented protocol implementation framework. The framework consists of two basic elements, conduits and information chunks or messages. The conduits can be connected to each other creating a conduit graph that represents the actual protocol stack, as shown in Figure 4. Messages represent the information flowing through the stack. There are four different conduits: Protocol, Mux, ConduitFactory and Adapter.

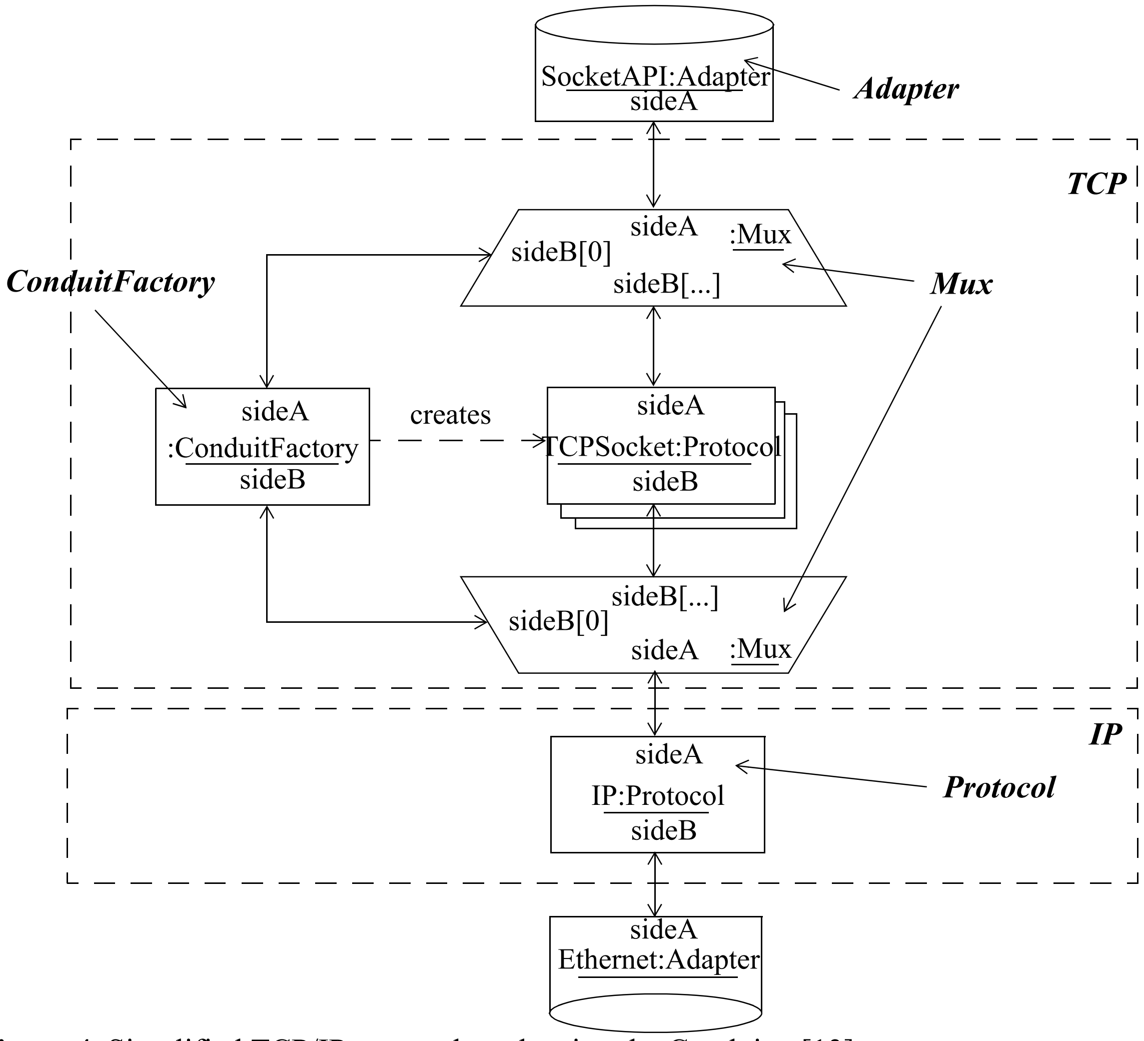
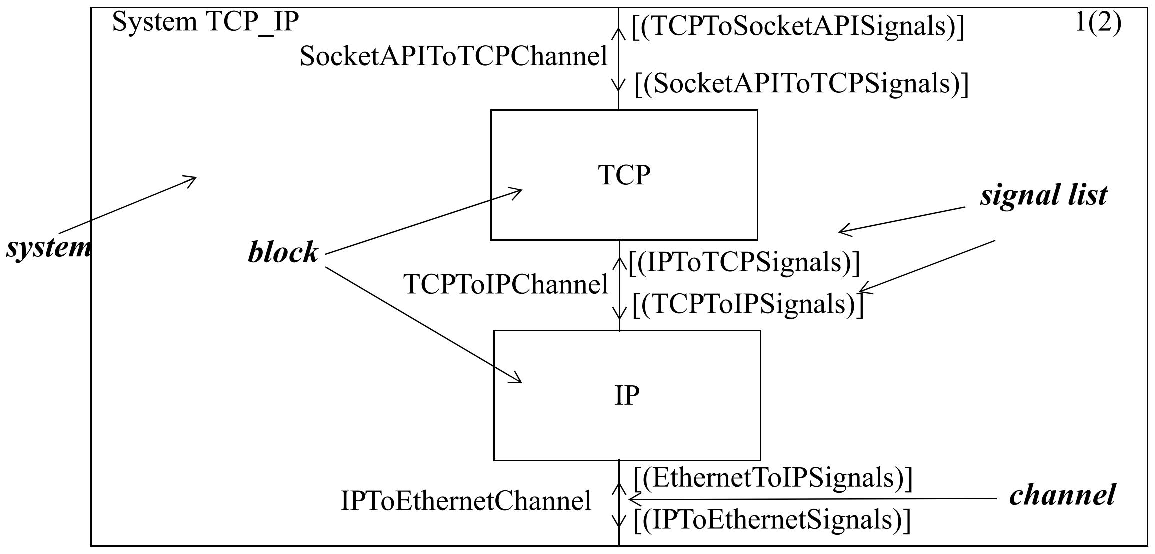


Figure 4. Simplified TCP/IP protocol stack using the Conduits+[13]

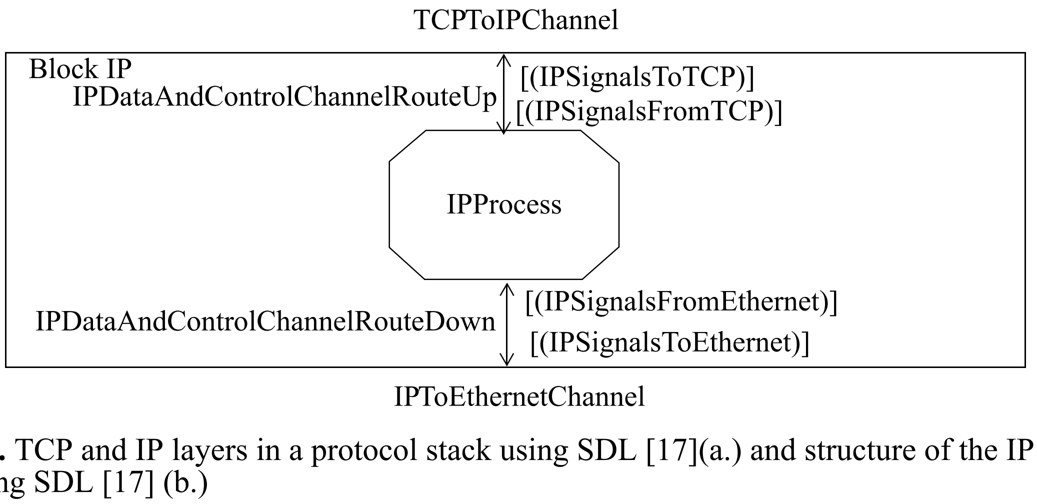
One example of a simple TCP/IP [9] protocol implementation using the Conduits+[13] is shown in Figure 4. In this figure there are two Adapters to connect conduits graph to Ethernet and to SocketAPI (TCP/IP socket application programming interface), two Muxes to handle connection-oriented socket communication of the TCP, and single Protocol conduit to handle all connectionless communication of the IP protocol. In Conduits+ one can not separate protocol layers, or entities, by encapsulating selected conduits within other structures. In Figure 4 additional notation is used to guide a reader and make easier for him or her to see the protocol entity boundaries

SDL. The Specification and Description Language (SDL)[17] is a standard language for specification and description of communicating systems. It is currently developed by ITU-T and is defined in the Z.100 recommendation [17]. There are several SDL versions. The current version is SDL2000 which contains many object-oriented features, and similar kind of notation as UML2[20]. However, there are no tool support existing for SDL2000, and for this reason older version of SDL [17] is used here as an example.

An SDL system consists of blocks which can communicate with each other and with the environment surrounding the system by sending signals. In Figure 5a the system consist of two blocks TCP and IP. They correspond to protocol layers. Inner structure of the IP block is shown in Figure 5b. Communication between the blocks is done via channels. Signal lists associated with the channels list the signals that are allowed to be sent to a given direction.



a.



b.

Figure 5. TCP and IP layers in a protocol stack using SDL [17](a.) and structure of the IP layer using SDL [17] (b.)

3 Domain Concepts for Communication Protocols

In this chapter domain concepts for communication protocols are introduced. These concepts and relations between them are independent from particular specification, implementation details or implementation language. A more detailed descriptions of these concepts can be found can be found from [1][2][3][4][6]. These concepts have been also used in ETSI Guideline document [5].

Relations between these concepts are presented in this paper using simple UML1[18] class diagrams. However, this does not limit their use in object-oriented design and implementation nor in UML.

The TCP/IP protocol suite [9] is used in this paper as an example. However, in real-life TCP/IP protocol suite is not coded as a layered structure.

In this paper following domain concepts (see Figure 6 for concepts and their relations) are introduced: *Protocol System*, *Protocol Entity*, *Environment Interface*, *Entity Interface*, *Peer Interface*, *Storage*, *Protocol Behavior*, *Router*, *Communication Manager*, and *Communication Session*.

- The *Protocol System* (see Figure 6 for concept, and Figure 8 and Figure 9 for TCP/IP example) encapsulates the whole protocol system, and forms the basis of other concepts. It describes a protocol system structure by specifying what are the components that a system is composed of, and how they are interconnected to each other.
- The *Protocol Entity* (see Figure 6 for concept, and Figure 7 for TCP example) represents an active entity in a protocol layer or sublayer. It contains Entity Interfaces. A Protocol Entity communicates with other Protocol Entities in the same system by exchanging messages through Entity Interfaces. A Protocol Entity needs to manage possible multiple concurrent communication sessions, store internal states and other information, communicate with other entities in the same system, and communicate with entities in peer systems.
- The *Entity Interface* (see Figure 6 for concept, and Figure 7 for TCP example) defines the allowed set of incoming and outgoing messages. It is these bindings between Entity Interfaces that specify how the components of the system are interconnected. The Entity Interface concept is considered to be bidirectional, as they are in the traditional protocol modeling. It interprets Entity Messages which are received from another Protocol Entity. It also produces Entity Message which are sent to another Protocol Entity in the same system.

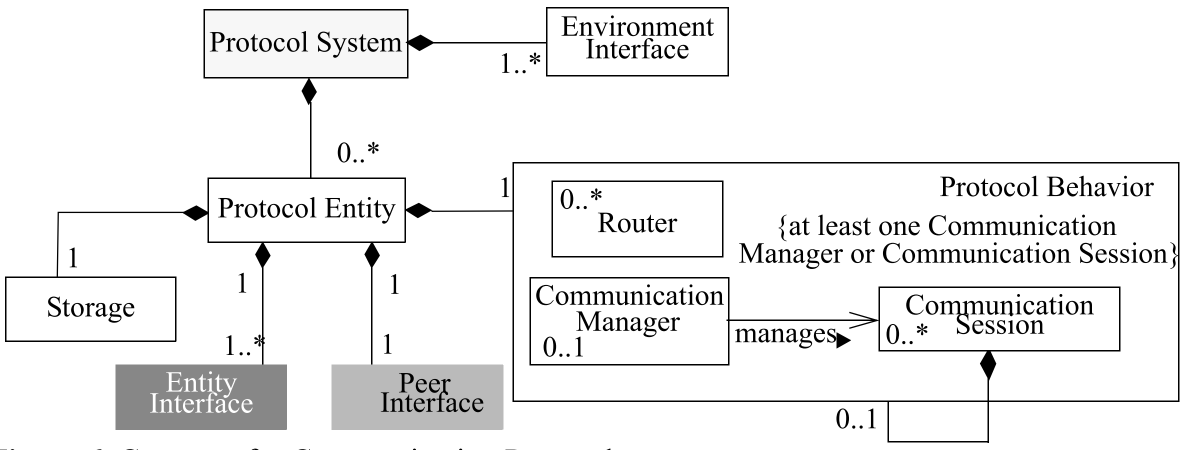


Figure 6. Concepts for Communication Protocols

- The *Environment Interface* (see Figure 6 for concept, and Figure 8 and Figure 9 for TCP/IP example) models interfaces to system's environment. From a protocol system's point of view an Environment Interface acts as a message source for incoming external messages and as a message sink for outgoing messages.
- The *Storage* (see Figure 6 for concept) contains all volatile and non-volatile information of a Protocol Entity. Information collected to the Storage can be visible for the whole Protocol Entity or it can be split to dedicated parts. An example of this is communication session specific information.
- The *Peer Interface* (see Figure 6 for concept, and Figure 7 for TCP example) handles communication between entities located in the peer protocol systems. It interprets Peer Messages which are received from a peer entity. It also produces Peer Messages which are sent to another entity in a peer system.
- The *Protocol Behavior* (see Figure 6 for concept, and Figure 7 for TCP example) encapsulates the intelligence of the protocol and contains roles which can be used to compose any kind of behavior of the protocol in concern. The Protocol Behavior contains *Routers*, a *Communication Manager*, and *Communication Sessions*. Detailed behavior of Protocol Behavior can be described using e.g. statecharts [18].
- The *Router* (see Figure 6 for concept, and Figure 7 for TCP example) is needed if there can be multiple receiving Communication Sessions for messages coming from a single entity interface. A Router routes incoming messages to the correct receiver i.e. a Communication Manager or one of Communication Sessions.
- The *Communication Manager* (see Figure 6 for concept, and Figure 7 for TCP example) creates, controls, and closes sessions as needed.
- The *Communication Session* (see Figure 6 for concept, and Figure 7 for TCP example) handles communication between two communicating peers. It uses a Peer Interface to send and receive messages as shown in Figure 7.

One example of the use of the Protocol Behavior pattern is shown in Figure 7 on page 7 as an object diagram. It presents a snap-shot of a simplified TCP protocol. In this diagram there are two concurrent communication sessions, TCPSockets.

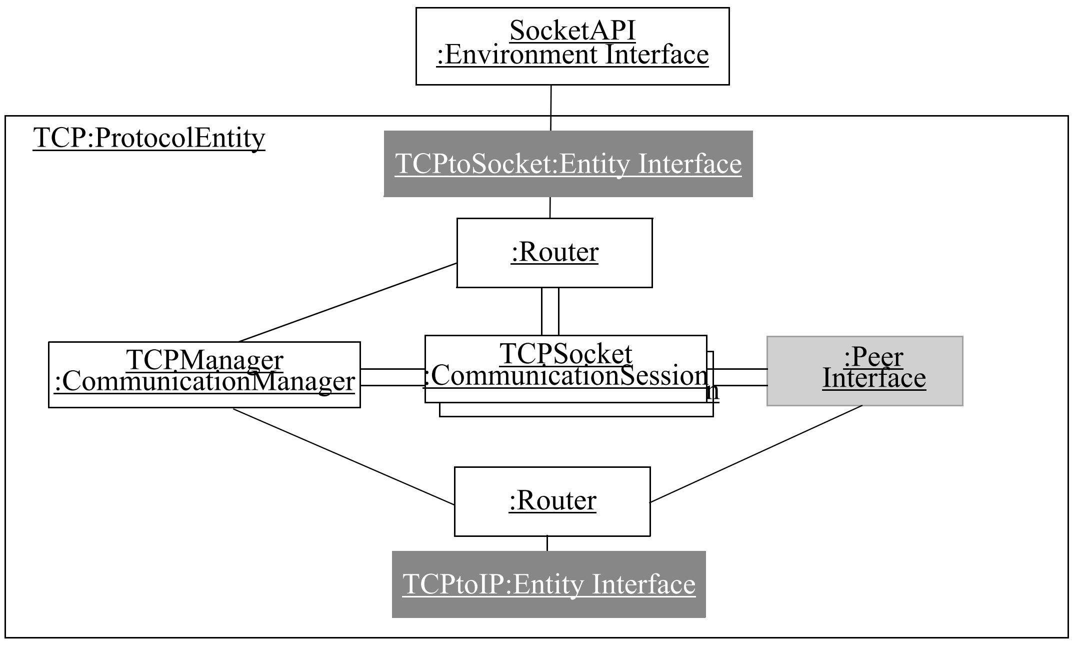


Figure 7. An example of concepts for protocols: the object diagram of the TCP protocol.

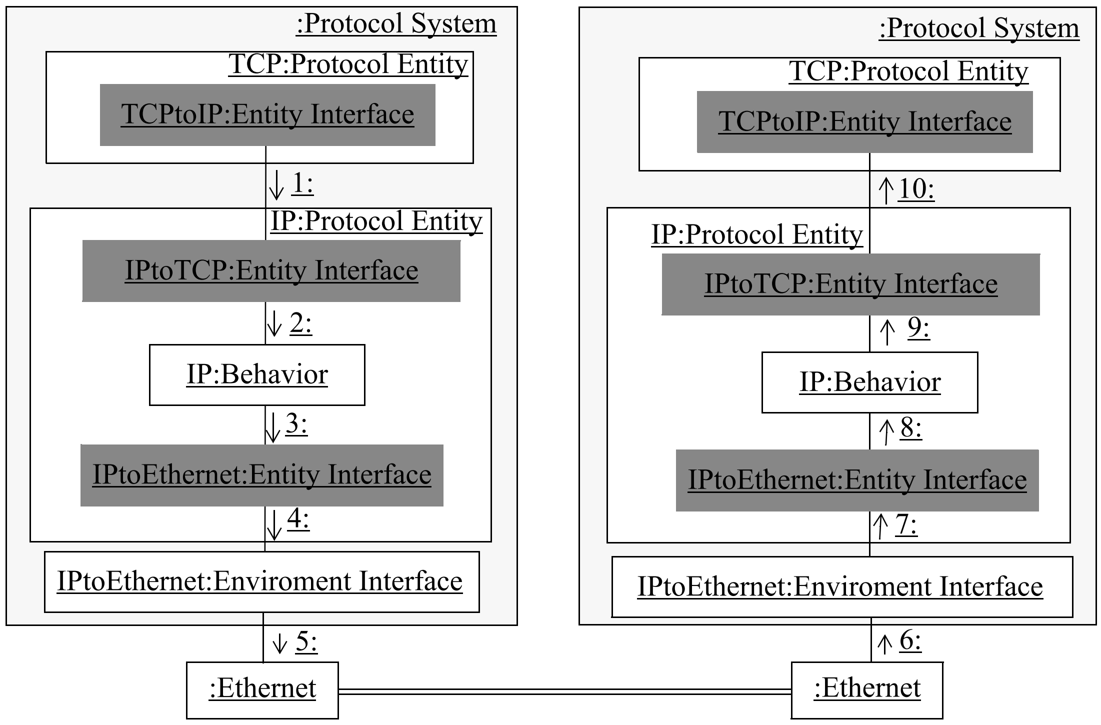


Figure 8. The Entity Interface centric view of the IP.

There are two different views how these concepts can be used when a whole protocol systems is described: the *Entity Interface centric view* and the *Peer Interface Centric view*. Following examples show two different views of it: the Entity Interface centric view in Figure 8; the Peer Interface centric view in Figure 9.

Figure 9 presents the Peer Interface centric view of the IP. The figure shows only those messages that are sent between the TCP and the IP protocol, and those between two peer IP protocols via a virtual connection. The layers below the IP are omitted. This view makes it possible to specify behavior of a protocol only in terms of the messages of the IP.

Figure 8 presents the Entity Interface centric view of the IP. Only physical interfaces and message paths are shown. The virtual peer connection of the IP is hidden inside the behavior of the IP.

Both views are very useful because they answer to different questions. The first one answers to the question "what is the correspondence between internal messages towards entity interface users (the TCP in case of the IP) and external messages towards peer protocol enti-

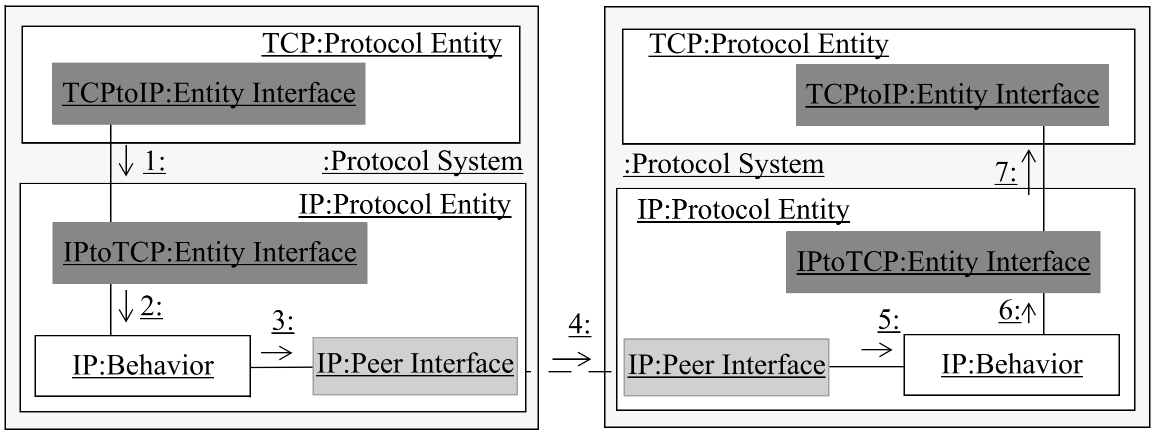


Figure 9. The Peer Interface centric view of the IP

ties”. The second one answers to the question “how do protocol entities within a system communicate with each other”.

3.1 Mapping Domain Concepts for Communication Protocols to Conduits+ and SDL

This section shows a part of mapping between two different protocol implementation frameworks and domain concepts for communication protocols introduced in this paper. More complete mapping can be found from [4].

Conduits+. A Protocol Entity itself is purely conceptual in Conduits+[13]. Its structure can be modeled by one or more Protocol conduits and any number of Muxes and ConduitFactories. These conduits define the Protocol Behavior of a Entity. Every conduit can contain ordinary variables which can be used to store protocol specific information.

The sides of conduits are used to connect conduits to each other, and are one part the Entity Interfaces. Other part of Entity Interface is set of Messengers which are used to define all possible incoming and outgoing events of a Protocol Entity.

A concrete example of these and use of Conduits+[13] to present a protocol entity is shown in Figure 4. The conduits inside the TCP box belong to one Protocol Entity. Muxes are used to route messages. ConduitsFactory is used to create new TCPSockets which are Protocol conduits. These have functionality to handle peer communication and they also contain all data of a single connection.

In Figure 10 it is shown how domain concepts for communication protocols are mapped in Conduits+[13]. This figure uses UML notation for templates from [18].

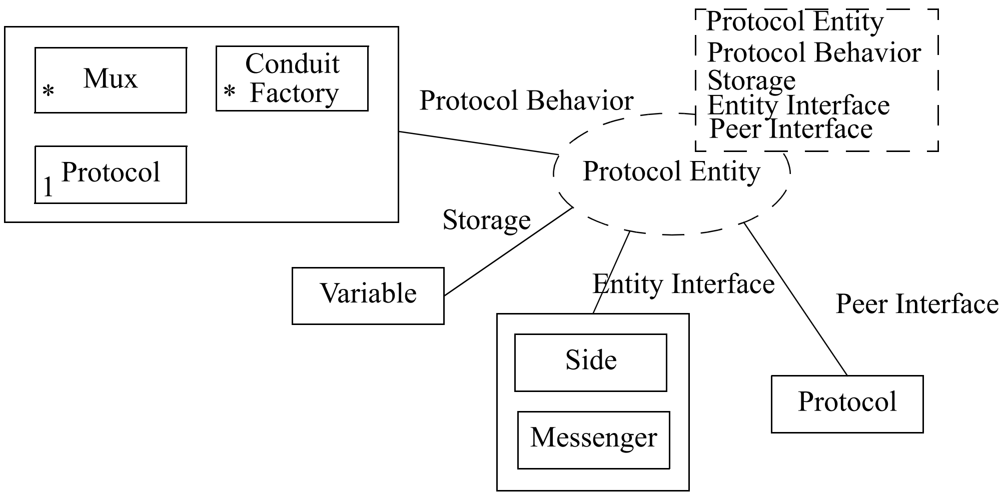


Figure 10. Mapping between Conduits+[13] and domain concepts for communication protocols

SDL. In SDL[17] a block can contain several processes or process sets. A block represents the Protocol Entity role as shown in Figure 11. The behavior of a block is not explicitly defined but it can be derived from the behavior of its processes. The processes model the Protocol Behavior.

Processes inside a block communicate with the block environment, i.e. the system outside them, also using signal routes. The routes linking processes to the block environment are attached to channels which are on the upper abstraction level connected to other blocks or the system environment.

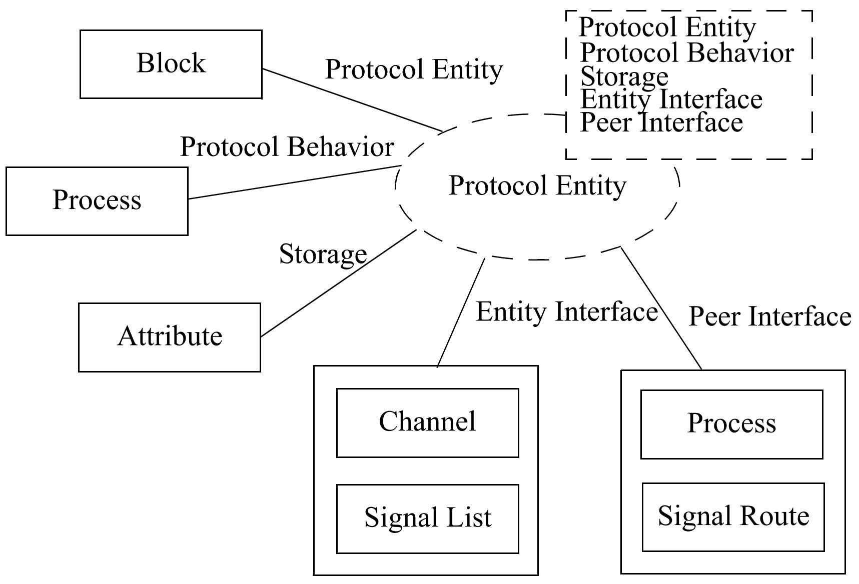


Figure 11. Mapping between SDL[17] and domain concepts for communication protocols

Signal lists can be used to define the signals that can be transmitted through a channel, these two component together define the Entity Interface between two entities. A channel name is shown in the border of a block. A concrete example of these and use of SDL[17] to present a protocol entity is shown in Figure 5.

In Figure 11 it is shown how domain concepts for communication protocols are mapped in SDL. This figure uses UML notation for templates from [18].

4 Conclusions and Future Work

The concepts presented in this paper were developed because there was a need to have a domain specific and implementation language independent language, which can overcome limitations of OSI based graphical notation[8] and UML[18][20]. These concepts have been used in different phases of protocol engineering, including specification, design, implementation, and documentation in author's organization.They are also used in ETSI Guideline document [5].

Due to limitations of available tools in the past these concepts are not used fully in code generation. They are used as stereotypes in different UML tools in a protocol design phase, but due to the limitations of automatic code generation in those tools, it is not possible to use the whole potential of the concepts. Nowadays there exist versatile domain modeling tools, which will be used to develop a graphical modeling language and code generation based on concepts introduced in this paper.

The author has also used these concepts in his lectures on protocol engineering at Helsinki University of Technology to teach newcomers. They are also used internally in the author's organization as a learning aid for new employees, or when there was a need to change from one protocol implementation environment to another.

There are two important tasks for future work: improve further domain concepts of communication protocols; develop a prototype for protocol development environment using these concepts and domain modeling tool.

5 References

[1] J. Pärssinen, *Conceptual Modeling of Protocol Systems*, Licentiate Thesis, 2003, Helsinki University of Technology Department of Computer Science and Engineering.

[2] J. Pärssinen, *Classes of Communication Systems*, 8th Summer School of Telecommunications, Lappeenranta, 1999.

[3] J. Pärssinen, N. von Knorring, J. Heinonen, M. Turunen, *UML for Protocol Engineering - Extensions and Experience*, Tools Europe 2000, 2000.

[4] J. Pärssinen, M. Turunen, *Patterns for Protocol System Architecture*, PLoP2000, August 13-16, 2000, Allerton Park, Monticello, Illinois, USA.

[5] S. Randall, Juha. Pärssinen, B. Koch, J-L. Roux, *Methodological approach to the use of object-orientation in the standards making process*, EG 201 872 V1.2.1, ETSI/MTS 2001.

[6] J. Pärssinen, M. Turunen, *Pattern Language for Architecture of Protocol Systems,* EuroPLoP2001, 4 - 8 July 2001, Irsee, Germany.

[7] W. Stallings, *Data and Computer Communications, 4th edition,* MacMillan, 1994.

[8] M. T. Rose, *The Open Book, A Practical Perspective on OSI,* Prentice-Hall, 1990.

[9] A. S. Tanenbaum, *Computer Networks, 2nd edition*, Prentice-Hall International, 1989.

[10] ITU-T, *Information Technology - Open Systems Interconnection - Basic Reference Model: The Basic Model, Recommendation X.200,* ITU, 1994.

[11] D. E. Comer, *Internetworking with TCP/IP Volume I: Principles Protocols, and Architecture, 3rd edition,* Prentice-Hall International, 1995.

[12] M. Mouly, M. Pautet, *The GSM System for Mobile Communications*, Telecom Pub, 1992.

[13] H. Hüni, R. Johnson, R. Engel, *A Framework for Network Protocol Software,* ACM, 1995.

[14] A. Karila, *Portable Protocol Development and Run-Time Environment*, Licentiate's Thesis, Helsinki University of Technology, 1986.

[15] J. Malka, E. Ojanperä, *CVOPS User's Guide*, Technical Research Center of Finland, 1998.

[16] Lappeenranta University of Technology, *OVOPS Home Page*, `http://ovops.lut.fi/`, 2003.

[17] ITU-T, *Recommendation Z.100 "Specification and description language (SDL)"*, 1993

[18] J. Rumbaught, I. Jacobson, G. Booch, *The Unified Modeling language Reference Manual*, Addison-Wesley, 1999.

[19] WAP Forum, *WAP Architecture Version 30-Apr-1998*, Wireless Application Protocol Forum Ltd., 1998.

[20] Object Management Group, *UML 2.0 Superstructure*, `http://www.omg.org/uml`, OMG, 2003.