Automatic Configuration of Communication Subsystems - A Survey -

B. Geppert and F. Rößler

SFB 501 Report 17/96

Automatic Configuration of Communication Subsystems - A Survey -

B. Geppert and F. Rößler

{geppert,roessler}@informatik.uni-kl.de

Report 17/96

Sonderforschungsbereich 501

Computer Networks Group

Computer Science Department

University of Kaiserslautern

P.O. Box 3049

67653 Kaiserslautern

Germany

Automatic Configuration of Communication Subsystems - A Survey -

B. Geppert and F. Rößler

Computer Science Department, University of Kaiserslautern
P.O. Box 3049, 67653 Kaiserslautern, Germany
{geppert,roessler}@informatik.uni-kl.de

Abstract

Today's communication systems are typically structured into several layers, where each layer realizes a fixed set of protocol functionalities. These functionalities have been carefully chosen such that a wide range of applications can be supported and protocols work in a general environment of networks. However, due to evolving network technologies as well as increased and varying demands of modern applications general-purpose protocol stacks are not always adequate.

To improve this situation new flexible communication architectures have been developed which enable the configuration of customized communication subsystems by composing a proper set of reusable building blocks. In particular, several approaches to automatic configuration of communication subsystems have been reported in the literature. This report gives an overview of theses approaches (F-CCS, Da CaPo, x-Kernel, and ADAPTIVE) and, in particular, defines a framework which identifies common architectural issues and configuration tasks.

1 Introduction

Today's communication systems are typically structured into several layers, where each layer realizes a fixed set of protocol functionalities. These functionalities have been carefully chosen such that a wide range of applications can be supported and protocols work in a general environment of networks. Well-known architectures of this type are the OSI stack [9] or the Internet Protocol Suite [17]. However, due to evolving network technologies as well as increasing demands of modern applications general-purpose protocol stacks are not always adequate [2]. In particular, varying demands on throughput and delay as well as on delay jitter, synchronization and multicasting are not well supported by existing protocol stacks. Also, classical protocols are not designed to exploit the advantages of advanced transmission technologies (e.g., fiber optics) and high-speed networks (e.g., ATM), which combine high bandwidth with low error rates and therefore shifted the performance bottleneck from transmission links to protocol processing nodes. Rather, classical protocols enforce the use of mechanisms that may actually not be needed by a given application, for instance, the use of error control mechanisms, which leads to reduced performance [3], [27], [29].

To improve this situation new flexible communication architectures are currently being developed which enable the configuration of customized communication subsystems (a communication subsystems upgrades a given basic communication service in order to provide a target service requested by the user, see Figure 1) by composing a proper set of reusable building blocks. Basically two kinds of configuration have to be distinguished. In case of *automatic configuration* the actual configuration process is performed by special configuration tools, where the input is a specification of the desired target service. According to the communication requirements suitable components of a pool of predesigned protocol building blocks are automatically selected, and a corresponding sub-

system implementation is generated. Automatic configuration generally supports the dynamic adaptation of an communication subsystem to varying user demands or network services. Another possibility is to view configuring as part of protocol design (*manual configuration*), where customization of special-purpose protocols and the reduction of additional development effort is of major concern [4]. This report exclusively deals with automatic configuration, i.e. when talking about configuration, automatic configuration is meant.

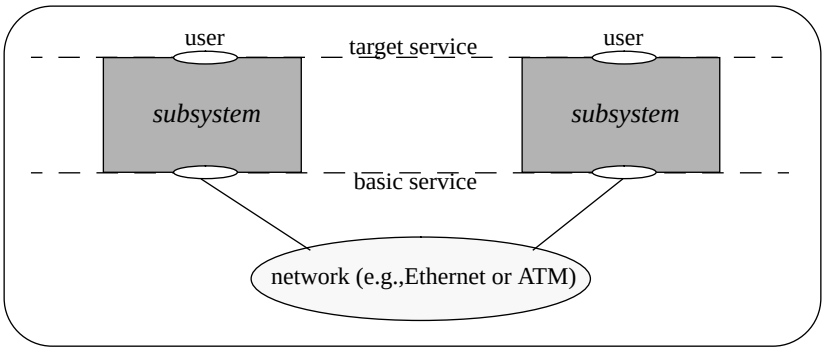


Fig. 1: communication subsystem

Several approaches to the automatic configuration of communication subsystems have been reported in the literature. This report gives an overview of F-CCS, Da CaPo, x-Kernel, and ADAPTIVE. In particular, a framework which identifies common architectural issues and configuration tasks is defined.

The remainder of the report is organized as follows: in Section 2 a framework for automatic configuration is presented, which outlines the general architecture of a configurable communication subsystem and describes necessary configuration tasks. This framework constitutes the common ideas of the approaches considered in this report. In Section 3 the approaches F-CCS, Da CaPo, x-Kernel, and ADAPTIVE are individually discussed and related to the framework of Section 2. We summarize the main results in Section 4.

2 A framework for automatic configuration

This section identifies the common ideas of the configuration approaches considered in this report. Furthermore, a uniform vocabulary is defined in order to simplify comparison. The approaches can be classified with respect to the granularity of the employed building blocks, which results in two basic subsystem architectures (Section 2.1). The necessary participants and tasks of the configuration process are discussed in Section 2.2.

2.1 Subsystem architecture

As already mentioned, conventional subsystems are structured into layers, where the set of layers and functionalities per layer is fixed. As a consequence, some layers may comprise irrelevant mechanisms or lack necessary mechanisms with respect to a desired target service (sometimes necessary functionalities, for instance multicasting, are even hidden from upper layers to build a uniform interface). Furthermore, the same functionality may be found in several layers. For instance, flow and error control is conducted hop-by-hop in OSI layer 2 and end-to-end in OSI layer 4. Also, several layers may segment and reassemble data units. To overcome the disadvantages of irrelevant, missing or redundant functionalities configuration is based on a new subsystem architecture, where two basic derivatives can be distinguished, which differ in granularity of the employed building blocks.

For a *protocol-based architecture* (Figure 2, left side) the building blocks are predesigned protocols (where existing protocols such as TCP or OSI-TP4 and newly developed protocols are equally suited). Contrary to conventional subsystems there is no rigid hierarchy of layers with a fixed set of functionalities per layer. Rather, customized protocol stacks are configured, where the number and functionality of layers may vary. As a prerequisite a *predecessor-successor relation* has to be defined over the set of predesigned protocols. The relation determines whether two protocols may be composed on top of each other (e.g., protocol A on top of protocol B and protocol B on top of protocol C to build the protocol stack of Figure 2). Protocol stacks, which are configured according to the predecessor-successor relation form valid communication subsystems.

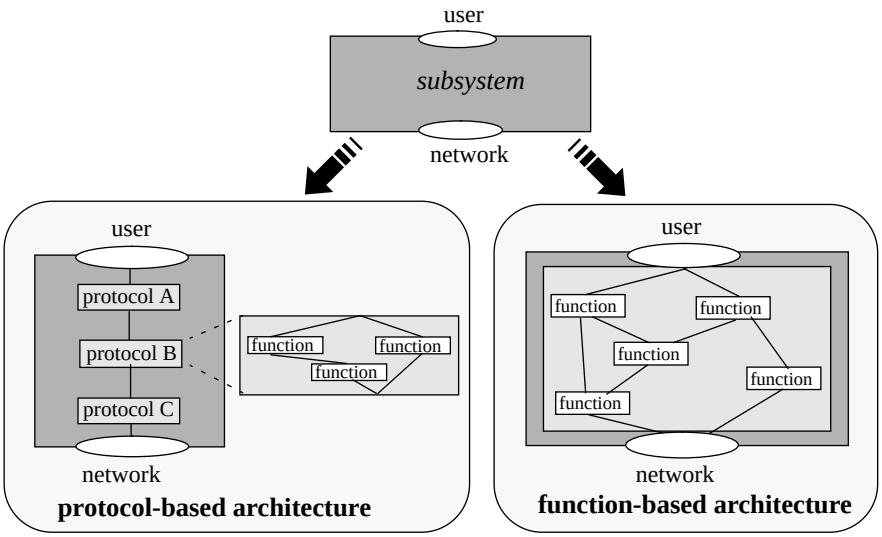


Fig. 2: subsystem architectures

Each protocol comprises multiple *protocol functions* (e.g., connection establishment, sequencing, checksum calculation, or routing) which may also serve as basic architectural components. The removal of protocol boundaries results in an architecture of separate protocol functions instead of a protocol stack (Figure 2). This *function-based architecture* is more flexible as finer-grained building blocks are employed, and it additionally supports parallel implementation. Instead of composing complete protocols, smaller units are configured to build an entire subsystem. Similar to the protocol-based architecture a *predecessor-successor relation* over the set of predesigned protocol functions contains the information about possible compositions. Only configurations according to the predecessor-successor relation are allowed.

A protocol function may be provided by different algorithms (called *mechanisms)*. Flow control, for instance, could be realized by a window-based or rate-based mechanism. Furthermore, a mechanism may comprise a sending and a receiving part. For instance, in case of a window-based flow control the sender has to administer a sending window and the receiver a corresponding receiving window. In order to distinguish between a mechanism and its implementation the latter is called a *module* (Figure 3). As a consequence a *predesigned* protocol function must comprise a set of possible mechanisms and corresponding modules. Examples for protocol functions and corresponding mechanisms are listed in Table 1.

In addition to the protocol-based and function-based architecture *hybrid architectures* are also possible, which combine predesigned protocols with protocols configured from a set of protocol functions.

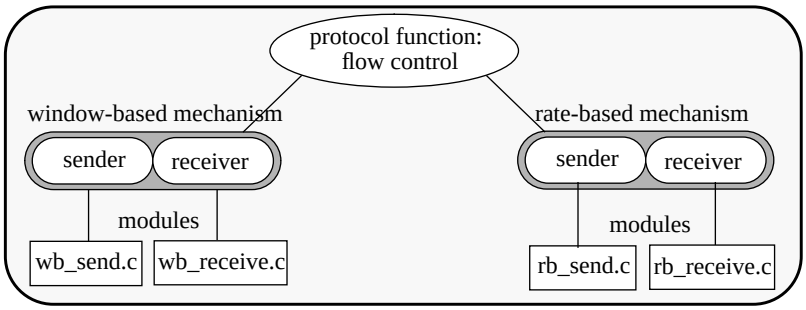


Fig. 3: relation between protocol functions, mechanisms, and modules

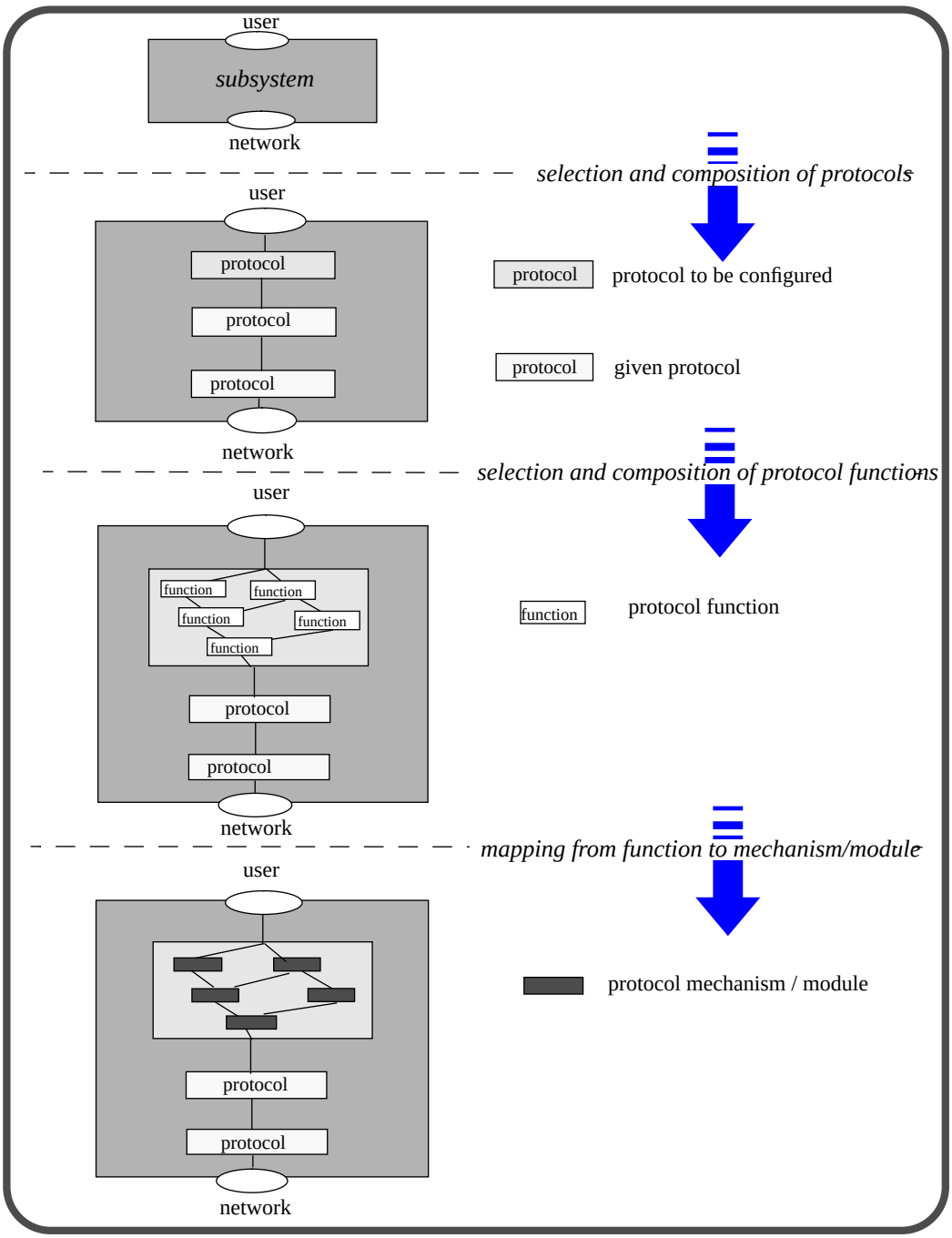


Fig. 4: configuring a subsystem with mixed architecture

protocol function	mechanism
flow control	Stop-and-Wait window-based rate-based
corruption control	checksum parity
connection management	implicit 2-way-handshake 3-way-handshake no connection (datagram)
acknowledgement	cumulative-positive cumulative-negative selective-positive selective-negative

Table 1: example protocol functions and mechanisms

2.2 Subsystem configuration

The architectural models of the previous section enable the configuration of customized communication subsystems with respect to specific communication requirements and offered network services. Given a pool of predesigned building blocks a subsystem is generally configured by selecting and composing a suitable subset. Two kinds of automatic configuration have to be distinguished. If several users share the same service requirements the corresponding subsystem can be configured in advance (i.e. before connection establishment) and invoked by the service user whenever needed. This is called *static configuration*. In case of *dynamic configuration* the subsystem is configured on demand (i.e. during connection establishment). Naturally, this increases connection set-up times, but it supports a flexible handling of changing conditions. For instance, an overloaded communication network can result in a *reconfiguration* during data transfer. In order to configure a communication subsystem the following tasks have to be performed:

- Specification of the functional and non-functional communication requirements.
- Negotiation of open configuration parameters between the communicating peers. For instance, communicating peers need to agree on a common basic service and compatible sender and receiver protocol machines if more than one subsystem configuration is possible. Negotiation is handled by a special meta-protocol.
- Actual configuring of the communication subsystem by selecting and composing suitable building blocks.
- Instantiating the configured subsystem on the underlying platform.

Figure 5 illustrates the participating agents of the configuration process which will be discussed in the following.

- **user:** The user specifies the required target service by the means of a special template (*service requirements model)*. It contains different parameters to describe the requested functional and non-functional requirements. Parameters may be mandatory or optional. The user may also determine the basic service to use and prescribe specific building blocks to be integrated in the subsystem.
- **configuration manager:** The configuration manager is responsible for controlling the order of events for configuring and especially negotiates open configuration parameters by means of a spe-

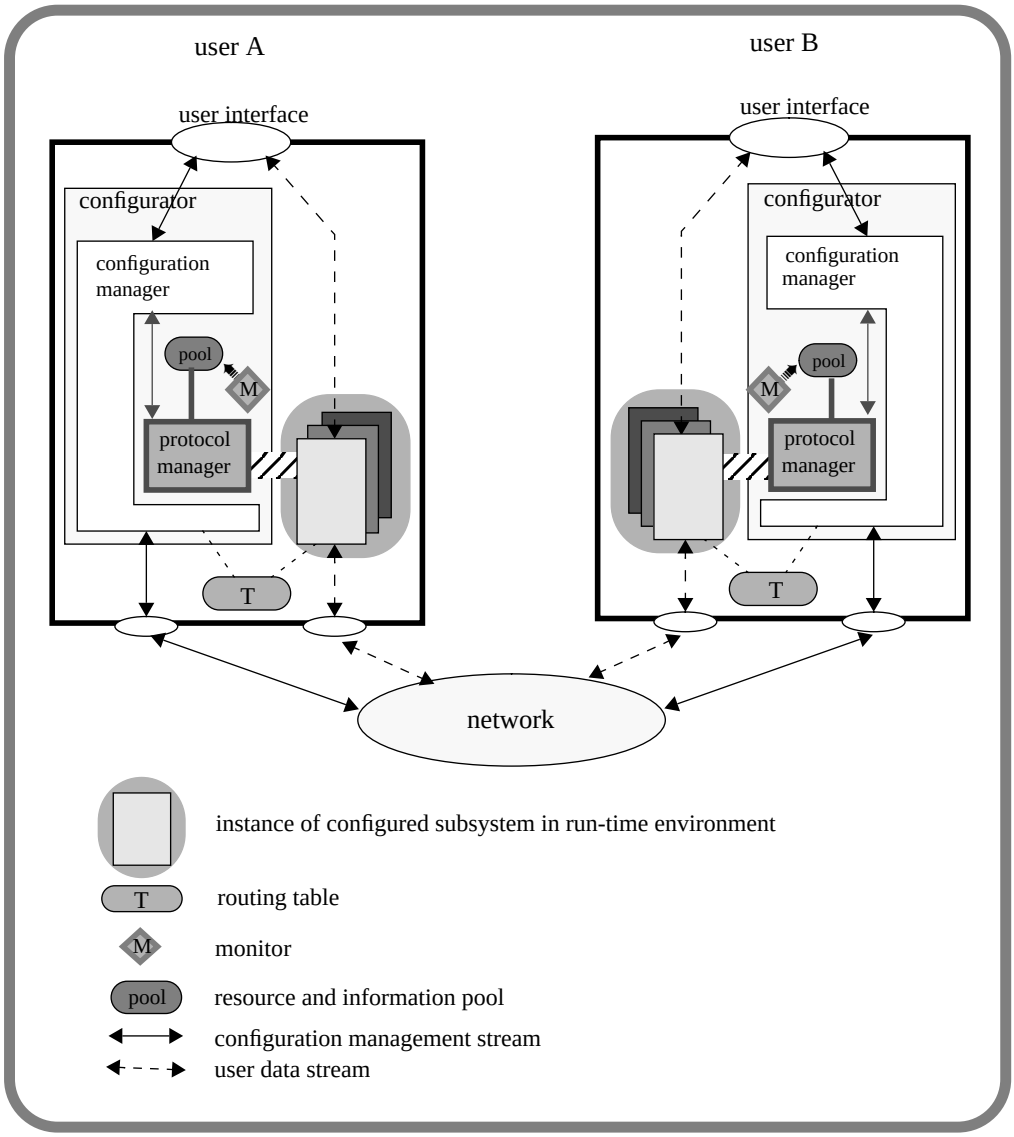
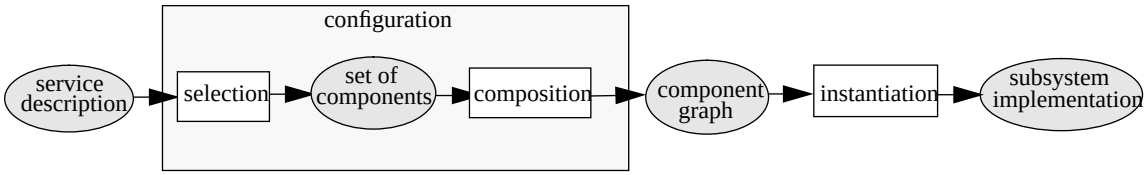


Fig. 5: configuration framework

cial meta-protocol. Negotiation is initiated after receiving a request from the service user to establish a connection. The result of this negotiation and the specified communication requirements are passed to the protocol manager for actual configuration and instantiation of the communication subsystem. Finally, the user is informed about the configuration results.

- **protocol manager:** The protocol manager performs the actual configuration based on the information given by the configuration manager. In a first step the service requirements have to be mapped on a proper set of building blocks. According to the employed subsystem architecture these building blocks are protocols or protocol functions which are predesigned and collected in a pool. In order to enable the **selection** of a proper subset the components must be described suitably. This description should not only define the predecessor-successor relation, but also additional properties such as the impact on the target service. The selection can then be performed by comparing the service requirements with the components‘ descriptions. The selected components are then **composed** according to their specified predecessor-successor relations which yields a *component graph*. Depending on the subsystem architecture this graph is called *protocol graph* or *protocol function graph*. In case of a function-based architecture each protocol function must additionally be mapped on a suitable mechanism and corresponding module which yields a *mechanism/module graph* (see Figure 4) for each communicating peer. After configuration the compo-

nent graph is **instantiated** on the target platform. The protocol manager therefore contains a platform-independent part for configuration and a platform-dependent part for instantiation. The following figure illustrates the sequence of activities and resulting descriptions.



- **resource and information pool:** The resource and information pool contains a collection of predesigned building blocks (protocols or protocol functions) specified in a proper notation. Furthermore, it contains a local database where additional information necessary for automatic configuration and instantiation is stored. This information comprises, for instance, the local memory capacity or current network load. It must be guaranteed that a proper set of building blocks can be selected by comparing the user provided service requirements description with the descriptions of the collected building blocks and the information currently stored in the local database. Also, the information given with the building blocks must suffice to correctly compose the selected set of components. In case of static configuration the preconfigured subsystems are also deposited in the pool.
- **monitor:** The monitor is responsible for updating the local database. Therefore it observes all values relevant for configuration and instantiation, e.g. the current load of local resources. This is especially important for dynamic reconfiguration.

3 Approaches for automatic configuration

In this section major approaches to automatic configuration are described within the framework of Section 2. Note, that the terms of Section 2 may differ from the terms originally used in the literature.

3.1 F-CCS

F-CCS (Function-based Communication Subsystem) has been developed at the University of Karlsruhe, Germany [22], [23], [28]-[31]. It supports dynamic configuration with a function-based subsystem architecture. The service user requests *sessions,* that comprise one or more unidirectional connections (for instance, a combination of video- and audio-streams or two unidirectional connections to yield a bidirectional data stream). For each connection a separate protocol is configured. In order to manage the interoperation between the different data streams of a session such as synchronization of audio- and video-streams, a special entity called *session manager* is introduced. The resulting subsystem architecture is shown in Figure 6.

Protocol functions are described in F-PCL (Function-Based Protocol Configuration Language) specially developed for this purpose [23][1]. A generic description template for a protocol function including the description of its mechanisms is shown in Figure 7. For each protocol function its name and assumptions for application (*„Required“, „Forbidden“*) are specified. Two examples are listed below (see [23]):

1. In [28] and [31] an earlier version of F-PCL called F-PDL (Function-based Protocol Description Language) is used.

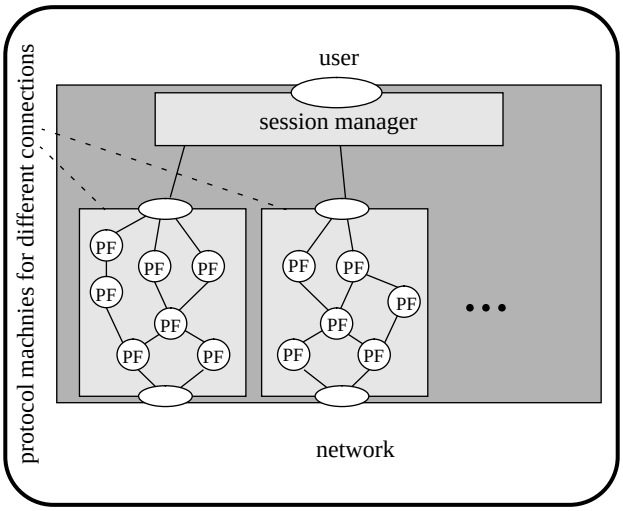


Fig. 6: subsystem architecture of F-CCS (only one session illustrated)

```
DEFINE-PF PF1
{
REQUIRED   } „assumptions for application
FORBIDDEN  }  of function PF1“

DEFINE-PFM mechanism-x
{
SENDER     } „is mechanism x part of
RECEIVER   } sender and/or receiver?“
REQUIRED   } „assumptions for application
FORBIDDEN  }  of mechanism x“
INPUT „needed input parameters“
OUTPUT „produced output parameters“
SUCC „allowed successor functions“
PRED „allowed predecessor functions“
TIME „expected processing time“
MEMORY „expected memory“
CODE „reference to C-module“
PFIB „reference to additional data“
}

PFM mechanism-y
{...}
}
```

Fig. 7: specification of a protocol function (PF) in F-PCL

```
DEFINE-PF Reassembly
{
REQUIRED  (Max_User_Data_Unit_Length + Max_Header) >
           Max_Network_User_Data_Length;
FORBIDDEN (Segmented_Delivery_Allowed = True);

DEFINE-PFM Stream_Reassembly_N ...
DEFINE-PFM Stream_Reassembly_P ...
}

DEFINE-PF Retransmission
{
REQUIRED  (! Data_Loss_Tolerated) |
          (Bit_Data_Loss_Network > Bit_Data_Loss_ThresholdValue) |
          (Packet_Data_Loss_ThresholdValue = 0);
FORBIDDEN (Jitter_ThresholdValue <=
          ((2 * Delay_Max) + Expected_Processing_Time_Receiver));

DEFINE-PFM Selective_Retransmission ...
DEFINE-PFM Cumulative_Retransmission ...
}
```

Furthermore, possible mechanisms are specified. It must be stated, if the mechanism is instantiated as part of the sender and/or receiver protocol machine (*„Sender“, „Receiver“*). Similar to protocol functions, assumptions for the application of mechanisms have to be specified (*„Required“, „Forbidden“*). The input and output parameters (*„Input“, „Output“*) define data dependencies, while the control flow depends on possible predecessor and successor protocol functions (*„Pred“, „Succ“*). Pointers to the corresponding module (*„Code“*) and to nec-

essary databases (*„PFIB“*) such as routing tables are given. The expected memory and processing time of the module is also specified (*„Time“, „Memory“*). Below, an example mechanism for the `Retransmission` protocol function is specified [23]:

```
DEFINE-PFM Selective_Retransmission
{
SENDER   True;
REQUIRED Bit_Data_Loss_Network > Bit_Error_Rate_Limit;
INPUT    Session_ID, Data_Stream_ID, sequence_number;
OUTPUT   Session_ID, Data_Stream_ID, bufferinfo;
PRED     Negative_Selective_Acknowledgement,
         Positive_Cumulative_Acknowledgement;
SUCC     Window_Flow_Control, Rate_Flow_Control,
         NetworkServiceInterface_Out_to_below;
TIME     100 * ySec;
MEMORY   300 * Byte;
CODE     ret_sel.lku;
}
```

Examples for protocol functions and corresponding mechanisms in F-CCS are listed in Table 2.

protocol function	**mechanism**
retransmission	go-back-N (cumulative), positive-selective, negative-selective, automatic repeat request, timer controlled
acknowledgement	cumulative-positive, cumulative-negative selective-positive, selective-negative
checksum	header, user data, packet, combination of different checksums
connection management	0-way-handshake (implicit, reliable), 1-way-handshake (implicit, unreliable), 2-way-handshake (explicit, unreliable), 3-way-handshake (explicit, reliable), no connection (datagram)
flow control	Stop-and-Wait window-based rate-based
routing	source-routing, record-routing
protocol error handling	report-error, discard-PDU
sequencing	sequence number, time stamp

Table 2: example protocol functions and mechanisms in F-CCS [23]

In order to initiate configuration the user specifies the requirements for each data stream of a session (step ① in Figure 8). The service requirements model is shown in Table 3. For each quantitative parameter a *threshold* value (mandatory), a preferred *useful* value (optional), and an average value (optional) are requested. The user can also prescribe specific functions or mechanisms to be applied for protocol configuration.

quantitative requirements **(threshold, average, useful)**	**qualitative requirements**	
throughput delay delay jitter response time rate data corruption data loss	session management:	inter-stream synchronization session update
	stream management:	ordered data delivery multicasting error tolerance (data loss, replication, corruption) expedited data transfer intra-stream synchronization syntax selection security (data security, authorization)
	data unit manipulation:	limited size of data units segmented delivery

Table 3: service parameters in F-CCS [23], [31]

The local *configuration manager* informs the remote configuration manager about the service requirements, to make sure that the communicating peer is willing to accept the session (step ② in Figure 8). For each data stream, the configuration manager instructs the protocol manager to configure a suitable protocol (step ③ in Figure 8). After configuration the protocol manager informs the configuration manager about the resulting protocol mechanism graph (step ⑦ in Figure 8). This description is passed to the remote configuration manager, which itself asks its protocol manager for instantiation (step ⑧ in Figure 8). If the remote configuration manager signals that the requested session is established, and the local protocol manager has also finished instantiation (step ⑨ in Figure 8), the configuration managers inform the users to start data transfer (step ⑩ in Figure 8).

The *protocol manager* transforms a service specification into a protocol function graph and finally generates the implementation. If enough local resources such as memory or processor capacity are available on both sides (step ④ and ⑤ in Figure 8), the protocol can be configured in the following way (step ⑥ in Figure 8):

1. Selection of protocol functions:
   the set of service parameters given by the user is mapped on a set of protocol functions. Some of the protocol functions and service parameters are directly related. For instance, if *ordered delivery* is requested, this demands the incorporation of the protocol function *sequencing* into the subsystem. If no *data loss* is tolerated, the protocol function *flow control* must be selected thus increasing the *delay* for data transmission. Other protocol functions depend on network characteristics. For instance, if the underlying network does not support sufficiently large frames the functions *segmentation* and *reassembly* are selected. Furthermore, some protocol functions may be prescribed by the user, and others are mandatory for each protocol, like sending (*NetworkServiceInterface_Out_to_below*) and receiving (*NetworkServiceInterface_In_from_below*). According to these dependencies and the *required*- and *forbidden*-conditions of the protocol functions (see Figure 7) suitable protocol functions are selected.
2. Selection of mechanisms/modules:
   mechanisms and modules are selected as prescribed by *required*- and *forbidden*-conditions. In case of multiple candidates the first mechanism listed is chosen.
3. Composing the mechanism graph:
   depending on the predecessor-successor relation the set of mechanisms is transformed into a mechanism graph.
4. Consistency check:
   finally, data dependencies as specified by input/output parameters are checked.

The configuration manager is informed about the resulting mechanism graph (step ⑦ in Figure 8). In a last step the sender and receiver modules are instantiated. The configuration algorithm is implemented on a Transputer platform as described in [23].

As delay in session establishment is an important criterion for configuration at run-time, four *service classes* are predefined with preconfigured protocol function graphs (static configuration) [31]. One service class supports unreliable real-time data transfer, where data corruption, loss, and duplication is tolerated up to certain threshold values (user specified), and ordered delivery as well as guarantees on delay, rate, and jitter are always met. Possible applications are, for instance, voice and video transmission. The second service class supports reliable real-time data transfer though rate and jitter is not guaranteed. Additionally expedited data transfer is supported. Possible applications are process-control applications. The third and fourth service classes support unreliable non-real time data transfer and reliable non-real time data transfer, respectively. The preconfigured protocol function graphs are deposited in the pool and can be retrieved by the protocol manager during session establishment.

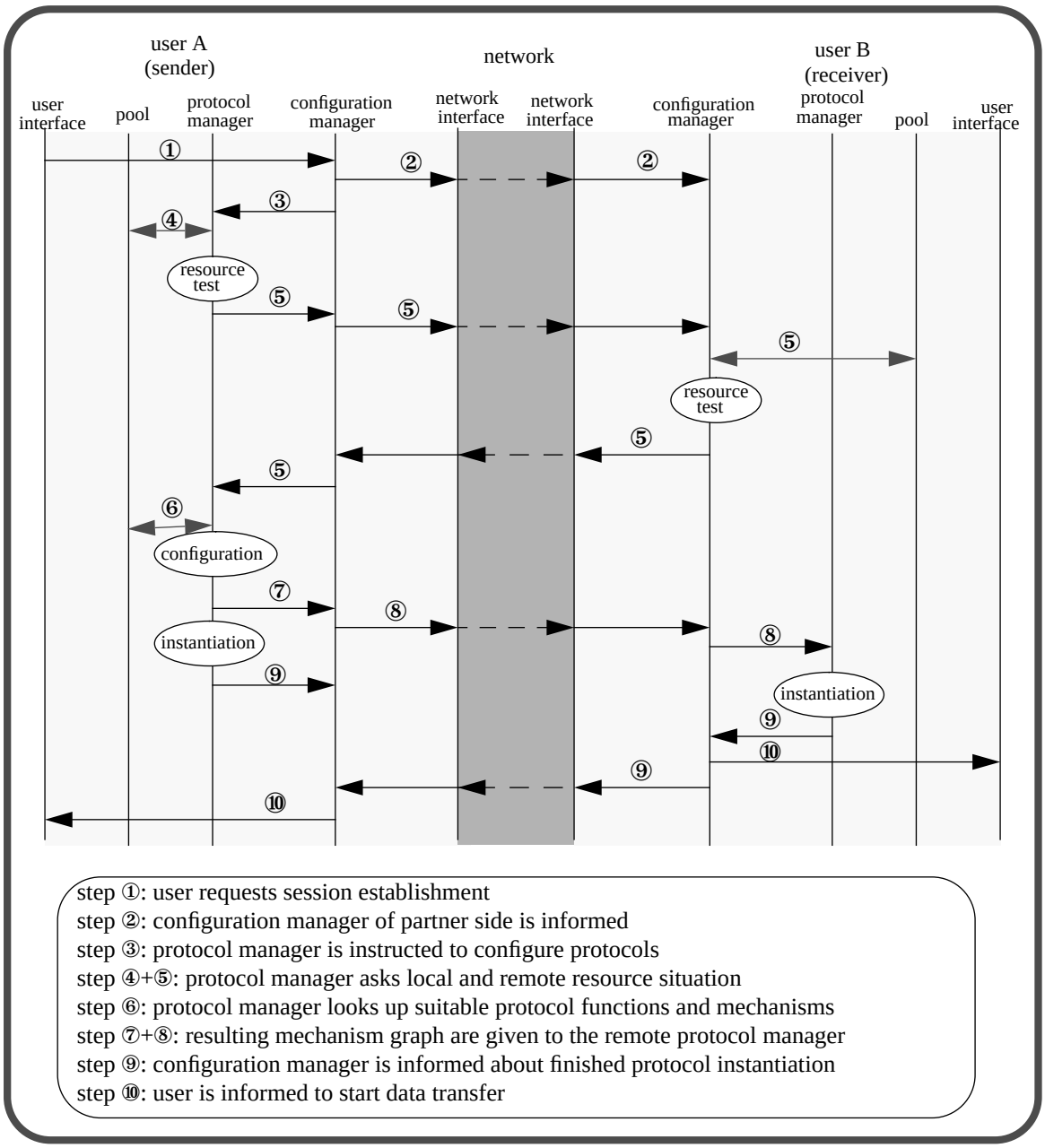


Fig. 8: order of events during configuration in F-CCS

3.2 Da CaPo

Da CaPo (Dynamic Configuration of Protocols) is another approach to dynamic configuration of function-based subsystems ([12]-[16], [24]-[26]). It was developed at the Computer Engineering and Networks Laboratory, ETH Zürich. Contrary to F-CCS, Da CaPo does not distinguish between connections and sessions. Thus multiple data streams can be handled by the same protocol.

Protocol functions of Da CaPo include, for instance, flow control, error control, sequencing, segmentation or reassembly. Instead of listing the possible predecessors and successors for each protocol function and composing a corresponding protocol function graph during configuration (such as F-CCS works) Da CaPo deposits a set of precomposed protocol function graphs in a pool. During configuration a graph is selected, that meets the communication requirements of the user and the characteristics of the chosen basic service (it is assumed, that for each pair of possible target and basic services a corresponding protocol function graph exists [14]). In order to complete configuration a proper mechanism is selected for each protocol function of the selected graph.

User requirements as well as the characteristics of mechanisms and network services are specified by the means of a special notation, which is defined in [14] and [15]. In particular, a set of quality of service (QoS) attributes such as throughput or delay jitter is defined, where each possible network service is characterized by specifying its attribute values. By the influence of the communication subsystem QoS characteristics of the basic service can degrade or improve when observed at the target service interface. For each mechanism certain functions are assumed that calculate the mechanisms individual influence on QoS degradation or improvement, respectively (e.g., a mechanism may guarantee an error rate of 10^{-9} and reduce the throughput to 90%). The communication requirements are specified by thresholds or don‘t care values ‘*‘ for each QoS attribute. The importance of an attribute with respect to the other attributes is defined by a *weight function*.

The user initiates configuration by specifying the required target service (step ① in Figure 9). Additionally the user instructs the configuration manager to either perform a local or a global configuration. In case of a *local configuration* the local protocol manager is asked to configure a suitable mechanism graph for the requested service (step ② in Figure 9). The resulting graph (step ⑤ in Figure 9) is forwarded to the configuration manager of the communicating peer (step ⑥ in Figure 9). Finally, both protocol managers start instantiation (step ⑦ in Figure 9).

In the case of *global configuration* the configuration managers have to negotiate a common configuration. Therefore the requirements specification is additionally passed to the remote manager (step ❶ in Figure 9) to also initiate configuration at the remote side (step ❷-❹ in Figure 9). Both protocol managers return a set of possible configurations (step ⑤ and ❺ in Figure 9). Local results are passed to the remote configuration manager (step ⑥ in Figure 9), which instructs its protocol manager to select one configuration (step ❻ in Figure 9). The resulting mechanism graph is returned (step ❼ in Figure 9) and both protocol managers start instantiation (step ⑦ in Figure 9). Actually, a second version of global configuration is provided, because data transfer can start immediately with a default protocol until global configuration is finished.

Before configuration the protocol manager checks if enough local resources are available (step ③ and ❸ in Figure 9). Afterwards it performs the *configuration* by selecting a protocol function graph and suitable mechanisms for each protocol function contained (step ④ and ❹ in Figure 9). In detail the following steps are conducted [12]:

1. reduction of the set of mechanisms:
   First the set of possible mechanisms is reduced. Therefore the mechanisms are ordered depending on the end-system load introduced. Those mechanisms which introduce an end-system load too high, which are not available (e.g. mechanisms with missing hardware modules), or which violate a threshold value, are eliminated.
2. selection of protocol function graphs:
   The requested service is mapped onto a set of possible protocol function graphs, which contains

one graph for each possible basic service. Only protocol function graphs are selected, with a network service performance, that at least meets the user requirements.

3. selection of a suitable mechanism graph:
   The resulting protocol function graphs are transformed into mechanism graphs and analysed one after the other. During construction of a mechanism graph the influence of the selected mechanisms on the QoS characteristics are incrementally calculated. If a threshold value is violated the next protocol function graph is considered. If it turns out, that several mechanism graphs meet the user requirements a *compliance degree* is calculated (by the means of weight functions that determine the importance of each QoS attribute) to select the best suited graph.

The resulting mechanism graph is returned to the configuration manager (step ⑤ and ❺ in Figure 9). Finally, the protocol manager is instructed to instantiate the mechanism graph (step ⑦ in Figure 9). The configuration tool is implemented in ANSI C on a Sparc 10/20 with a UNIX environment (SunOS 4.1.3).

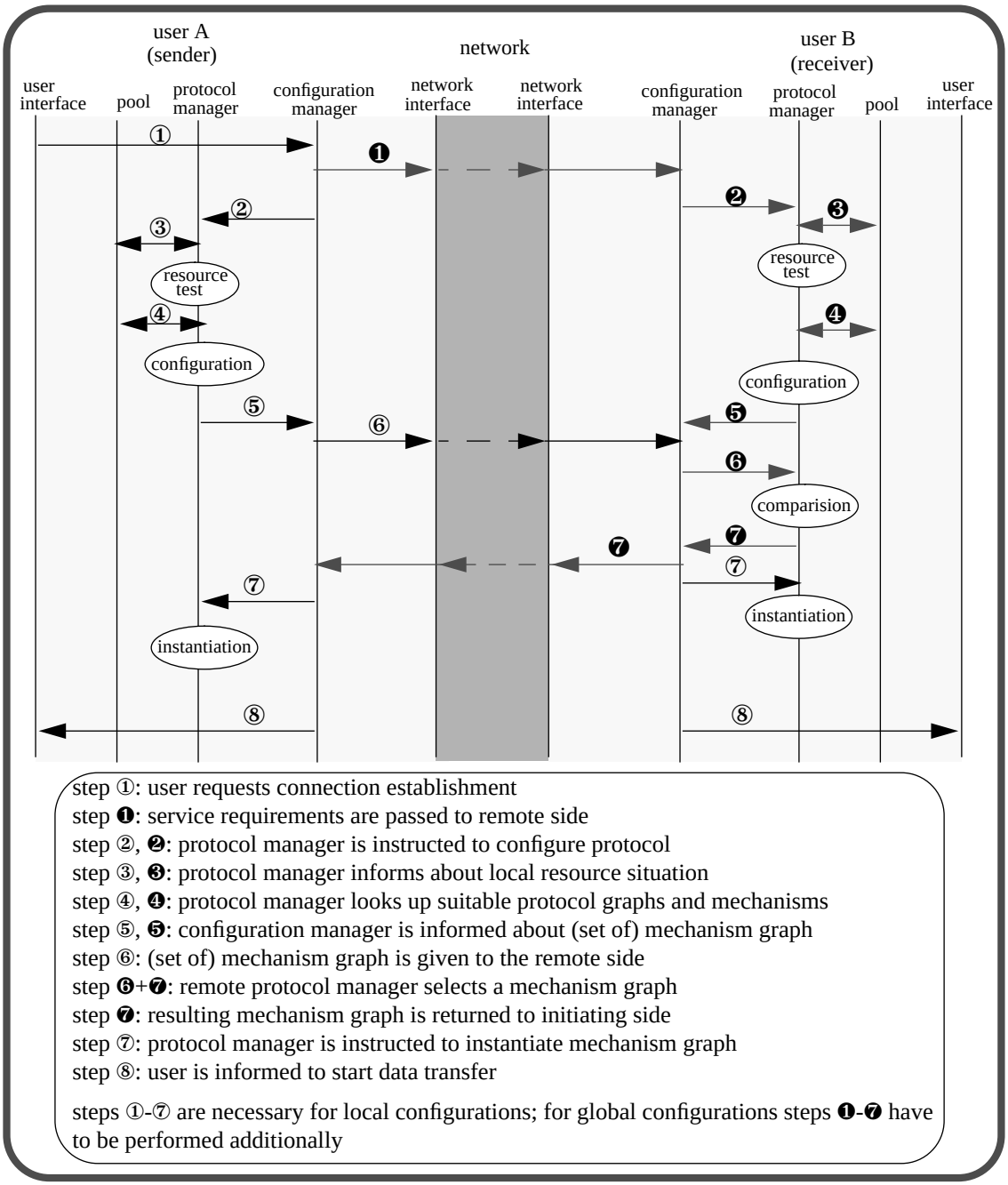


Fig. 9: order of events during configuration in Da CaPo

In order to reduce the time for configuration Da CaPo also supports the use of predefined services (static configuration). The corresponding mechanism graphs are stored in the resource pool and instantiated when requested by the user.

3.3 *x*-Kernel

The *x*-kernel is an experimental operating system kernel developed at the University of Arizona [8], [11]. The architecture of the *x*-kernel enables the configuring of protocol-based subsystems embedded in the kernel. Thus all subsystems to be included in the kernel have to be defined during kernel configuration, i.e. dynamic configuration is not possible.

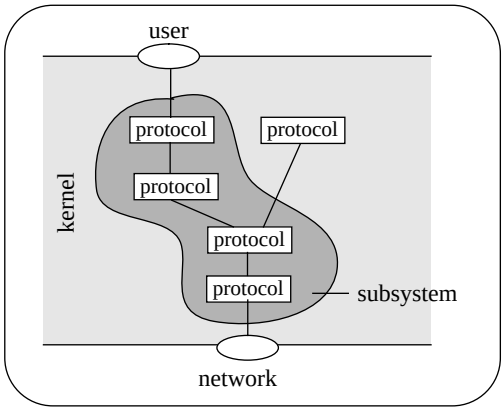


Fig. 10: protocol-based architecture of the *x*-kernel

Contrary to F-CCS and Da CaPo, the *x*-kernel relies on a protocol-based architecture. Example building blocks are IP, TCP, UDP, TFTP, RPC or NFS. All building blocks offer the same set of operations. Figure 11 lists some examples. This provides a common interface, which simplifies the task of composing the different protocols. Starting with the application, each protocol invokes the uniform operations exported by the other protocols on which it depends. The predecessor-successor relation is defined by a protocol dependency graph, where each path represents a possible protocol stack (see Figure 10).

Uniform protocol interface:

```
session = open(high_level_protocol, low_level_protocol, participants)
A high-level protocol calls a low-level protocol to open a session with the
given participant set.

push(session, msg)
Pass a message down to a session.

pop(session, msg)
Pass a message up to a session.

close(session)
Close a session.
```

Fig. 11: example operations of the x-kernel components [11]

The protocol dependency graph is defined during configuration of the operating system kernel. The graph, which is specified in a special graph description language, is read by a composition tool generating C code and instantiated at kernel boot time. Each resulting path in the graph can be interpreted as a preconfigured subsystem stored in a resource pool.

3.4 Other approaches

In the following, four variants of the previously discussed approaches are sketched.

AVOCA [10] is a further stage of the *x*-kernel. The communication subsystem is still embedded in the kernel, but AVOCA differs from the original *x*-kernel in the granularity of the building blocks: instead of composing complete protocols, finer-grained „micro protocols“ are employed, representing protocol mechanisms such as demultiplexing, segmentation, reassembly, routing or data encoding. These micro protocols provide the same uniform protocol interface as the original *x*-kernel protocols (see Figure 11). A protocol graph in AVOCA lists all possible configurations of mechanisms. In order to ensure, that each message is proccessed by the proper mechanisms, so called virtual protocols are introduced, which guide the messages through the protocol graph depending on their header information (Figure 12).

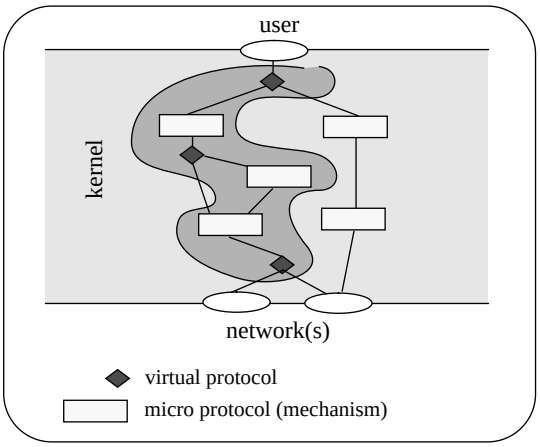


Fig. 12: function-based protocol architecture of AVOCA

The **System for Constructing Configurable High-Level Protocols** is another stage of the *x*-kernel [1]. It realizes a hybrid architecture, where configured protocols are composed with standard *x*-kernel protocols. Therefore the configured protocols have to offer the same uniform protocol interface as the *x*-kernel protocols. Mechanisms are, for instance, reliability, failure detection and acknowledgement. Selected and/or user-written mechanisms are linked with library routines to form a configured protocol, which is then integrated in the *x*-kernel protocol graph.

ADAPTIVE (A Dynamically Assembled Protocol Transformation, Integration, and eValuation Environment) has been developed by D.C. Schmidt and a team of researches at the University of Irvine, California [18], [19]. The approach is very similar to F-CCS. It is based on the same protocol architecture (see [20], [21]), but differs in the implementation of the mechanisms (modules), which are written in C++ using object-oriented design and implementation techniques to reduce the effort for developing flexible and extensible communication system software. Furthermore, the major goal is not to support configuration at run-time, but to offer an environment for developing and experimenting with different protocol designs. ADAPTIVE offers a framework for experimenting with different mechanisms and process architectures of the protocol implementation. The platform-dependent part of its protocol manager creates implementations for different platforms (not only for Transputers as F-CCS does). The instantiated protocols are optimized for the different target platforms depending on available interprocess communication mechanisms (shared memory, message passing), memory and bus architectures, and network adapters.

DyCAT (Dynamically Configurable & Adaptive Transport System) was developed at the University of Aachen (Germany) and is another approach mainly influenced by F-CCS. The long goal is to realize a function-based architecture with dynamic configuration, where the protocol functions and con-

figured protocols are modelled and simulated using Product Nets (see [5]). A first realization of DyCAT, presented in [6] and [7], deals with protocol-based architectures, where preconfigured protocol stacks with specific QoS characteristics are provided (static configuration). Example protocol stacks are TCP/ IP or XTP on top of Ethernet, ISDN, B-ISDN or FDDI.

4 Summary

Due to evolving network technologies as well as increasing demands of modern applications general-purpose protocol stacks are not always adequate. To improve this situation flexible communication architectures have been developed with protocols and finer-grained protocol functions as constituent building blocks. These architectures enable the automatic configuration of customized communication subsystems with respect to specific communication requirements and offered network services. Given a pool of predesigned building blocks a subsystem is generally configured by selecting, composing, and instantiating a suitable subset. We discussed the major approaches to automatic configuration of communication subsystems (the main characteristics are summarized in Table 4). As could be seen the protocol functions identified by the function-based approaches are basically the same, which indicates that the proposed pool of protocol-functions is well-founded. Because of finer-grained building blocks, function-based architectures are more flexible and better suited for dynamic configuration than protocol-based architectures. The approaches considered in this report have in common that protocol *implementations* are configured. As a major drawback, the use of implementation languages prevents the resulting communication subsystem from being verified, which is further complicated by the configuration of protocols during connection establishment. Also, the extension of the component pool appears to be difficult in these approaches, because the knowledge about composition principles is not explicitly described. Here, the use of formal description techniques allowing an abstract, unique specification of protocol components and component interactions seems to be mandatory.

	Architectural Model	**Service Requirements Model**	**static/dynamic Configuration**	**Target Platform**
F-CCS	function-based	yes	both	Transputer
Da CaPo	function-based	yes	both	Sparc/SUN
***x*-kernel**	protocol-based	no	static	-
AVOCA	function-based	no	static	-
SCCP	hybrid	no	static	Mach
ADAPTIVE	function-based	yes	both	several
DyCAT	protocol-based (function-based)	yes	(both)	-

Table 4: comparision of configuration approaches

Acknowledgements. Special thanks go to Prof. Dr. R. Gotzhein for his valuable comments and discussions on an early version of this report.

References

[1] N. Bhatti and R. Schlichting, *A System for Constructing Configurable High-Level Protocols*, ACM SIGCOMM, 1995, p. 138-150

[2] T. Braun, *Ein paralleles Transportsubsystem für zellenbasierte Hochgeschwindigkeitsnetze*, PhD thesis, VDI-Verlag, Reihe 10, Nr. 264, 1993

[3] T. Braun and M. Zitterbart, *Entwurf eines parallelen Transportprotokolls*, GI/ITG Fachtagung Kommunikation in verteilten Systemen, München, 3-5 März 1993

[4] B. Geppert and F. Rößler, *Pattern-based Configuring of a Customized Resource Reservation Protocol with SDL*, SFB 501 Report 19/96, University of Kaiserslautern, Germany

[5] B. Heinrichs, *DyCAT - A Flexible Transport System Architecture*, IEEE International Conference on Communications, ICC‘93, Genf, Schweiz, 23.-26. Mai 1993, S. 1331-1335

[6] B. Heinrichs and W. Reinhardt, *The DyCE Concept - Architecture and Implementation Strategies*, in Proceedings of IFIP/ICCC conference on Integretated Broadband Communication Networks and Services, Copenhagen, April 1993

[7] B. Heinrichs and W. Reinhardt, *Adaptive QOS driven Communication Architecture*, Proc. EuroArch‘93, Springer Publishers, 1993

[8] N. Hutchinson and L. Peterson, *The x-Kernel: An Architecture for Implementing Network Protocols*, IEEE Transactions on Software Engineering, Vol. 17, No. 1, January 1991, p. 64-76

[9] ISO 7498, Information technology -- Open Systems Interconnection -- Basic Reference Model

[10] S. O‘Malley and L. Peterson, *A Dynamic Network Architecture*, ACM Transactions on Computer Systems, Vol. 10, No. 2, May 1992, p. 110-143

[11] L. Peterson, N. Hutchinson, S. O‘Malley, and H. Rao, *The x-Kernel: A Platform for Accessing Internet Resources*, IEEE Computer, May 1990, p. 23-33

[12] Th. Plagemann, A. Gotti, and B. Plattner, *CoRA - A Heuristic for Protocol Configuration and Resource Allocation*, In Proceedings of IFIP Fourth International Workshop on Protocols for High-Speed Networks, Vancouver, Canada, August 1994

[13] Th. Plagemann and B. Plattner, *Evaluating Crucial Performance Issues of Protocol Configuration in Da CaPo*, In Proceedings of HIPPARCH‘94, First International Workshop on High Performance Protocol Architectures, Sophia Antipolis, France, December 1994

[14] Th. Plagemann, B. Plattner, M. Vogt, and Th. Walter, *A Model for Dynamic Configuration of Light-Weight Protocols*, In Proceedings of IEEE Third Workshop on Future Trends in Distributed Systems, Taipei, Taiwan, April 1992

[15] Th. Plagemann, B. Plattner, M. Vogt, and Th. Walter, *Modules as Building Blocks for Protocol Configuration*, In Proceedings of ICNP‘93, International Conference on Network Protocols, San Francisco, October 1993

[16] Th. Plagemann, J. Waclawczyk, and B. Plattner, *Management of Configurable Protocols for Multimedia Applications*, In Proceedings of ISMM International Conference on Distributed Multimedia Systems and Applications, Honolulu, USA, August 1994

[17] J. Postel (Ed.), *Internet Official Protocol Standards*, RFC 1720, Network Working Group, March 1996

[18] D.C. Schmidt, *An Object-Oriented Framework for Dynamically Configuring Extensible Distributed Systems*, IEE Distributed Systems Engineering Journal (Special Issue on Configurable Distributed Systems), Volume 2, No 4, December 1994

[19] D.C. Schmidt, D.F. Box, and T. Suda, *ADAPTIVE: A Dynamically Assembled Protocol Transformation, Integration, and eValuation Environment*, Concurrency Practice and Experience, Vol. 5, No. 4, June 1993

[20] D.C. Schmidt, B. Stiller, T. Suda, A. Tantawy, and M. Zitterbart, *Configuration Support for Flexible, Function-Based Communication Systems*, In Proceedings of 18th Conference on Local Computer Networks, Minneapolis, Minnesota, September 1993

[21] D.C. Schmidt, B. Stiller, T. Suda, and M. Zitterbart, *Configuring Function-based Communication Protocols for Multimedia Applications*, Proceedings of the 8th International Working Conference on Upper Layer Protocols, Architectures, and Applications, Barcelona, Spain, June, 1994

[22] B. Stiller, *FuKSS: Ein funktionsbasiertes Kommunikationssubsystem zur flexiblen Konfiguration von Kommunikationsprotokollen*, 1. Arbeitstreffen zur Architektur und Implementierung von Hochleistungs-Kommunikationssystemen der GI/ITG-Fachgruppe Kommunikation und Verteilte Systeme, Karlsruhe, 17/18. Januar 1994

[23] B. Stiller, *Flexible Protokollkonfiguration zur Unterstützung eines diensteintegrierenden Kommunikationssubsystems*, PhD thesis, VDI-Verlag, Reihe 10, Nr. 306, 1994

[24] M. Vogt, Th. Plagemann, B. Plattner, and Th. Walter, *Eine Laufzeitumgebung fuer Da CaPo*, GI/ITG-Arbeitstreffen über Verteilte Multimedia-Systeme, Stuttgart, February 1993

[25] M. Vogt, Th. Plagemann, B. Plattner, and Th. Walter, *A Run-time Environment for Da CaPo*, In Proceedings of INET‘93, International Networking Conference of the Internet Society, San Francisco, August 1993

[26] M. Vogt, Th. Plagemann, B. Plattner, and Th. Walter, *Parallelitätsaspekte in Da CaPo*, 1. Arbeitstreffen zur Architektur und Implementierung von Hochleistungs-Kommunikationssystemen der GI/ITG-Fachgruppe Kommunikation und Verteilte Systeme, Karlsruhe, 17/18. Januar 1994

[27] M. Zitterbart, *High-Speed Transport Components*, IEEE Network Magazine, Vol.5, No.1, p. 54-63, January 1991

[28] M. Zitterbart, F*unktionsbezogene Parallelität in transportorientierten Kommunikationsprotokollen*, PhD thesis, VDI-Verlag, Reihe 10, Nr. 183, 1991

[29] M. Zitterbart, *Flexible und effiziente Kommunikationssysteme für Hochleistungsnetze*, International Thomson Publishing, 1995

[30] M. Zitterbart, B.Stiller, and A. Tantawy, *Application-Driven Flexible Protocol Configuration*, GI/ITG Fachtagung Kommunikation in verteilten Systemen, München, 3-5 März 1993

[31] M. Zitterbart, B. Stiller, and A. Tantaway, *A Model for Flexible High-Performance Communication Subsystems*, IEEE Journal on Selected Areas in Communications, Vol. 11, No. 4, p. 507-518, May 1993