CogPiT — Configuration of Protocols in TIP

Burkhard Stiller *

University of Cambridge, Computer Laboratory

New Museums Site, Pembroke Street

Cambridge, CB2 3QR, England, U.K.

Phone: +44 +1223 334476, FAX: +44 +1223 334678

E-Mail: Burkhard.Stiller@cl.cam.ac.uk

Abstract

The variety of upcoming applications in terms of their performance and Quality-of-Service (QoS) requirements is increasing. Besides almost well-known applications, such as teleconferencing, audio- and video-transmissions, even more contemporary ones, such as medical imaging, Video-on-Demand, and interactive tutoring systems, are introduced and applied to existing networks. On the contrary, traditionally data-oriented applications, such as file transfer and remote login, are considerably different in terms of their QoS requirements. Therefore, the consequences of this evolution effect the architectures of end-systems, e.g., *workstations that have to be capable of maintaining all different kinds of multi-media data, and intermediate-systems as well.*

Therefore, a configuration approach of communication protocols has been developed to support the variety of applications. This approach offers the possibility to configure communication protocols automatically depending on the application requirements expressed in various QoS parameters. The result, an application-tailored communication protocol, matches the requested application requirements as far as possible. Additionally, network and system resources (NSR) are taken into account for a well-suited configuration.

The Configuration of Protocols in TIP is called **CogPiT** *and is part of the Transport and Internetworking Package (TIP). As an example, in the TIP environment the transport protocol TEMPO is used for configuration purposes.*

* The author is currently on leave from Universität Karlsruhe, Institut für Telematik, D-76128 Karlsruhe, Germany, and is sponsored by the Commission of the European Communities as a Research Fellow under the Human Capital and Mobility Scheme (RG 19327). This work has been performed in Karlsruhe within the Cooperation between the Universität Karlsruhe, Institut für Telematik and Siemens AG, ZFE München, except for the final testing and evaluation phase, which has been done remotely distributed between Karlsruhe and Cambridge. The generation of this Final Report took place in Cambridge as well, which includes besides new results and details on the prototypical implementation of CogPiT, updated versions of former reports number 1, 2, and 3, respectively.

Contents

List of Figures

List of Tables

1 Project Targets

Existing communication protocols and processing environments are not well suited for a sufficient support of innovative applications, for their emerging Quality-of-Service (QoS) requirements, and for future networks, since the protocol functionality of considered protocols is limited and pre-defined and the run-time environment is not efficient. However, the Transport and Internetworking Package (TIP) allows for the configuration of communication protocols out of protocol building blocks. While newly defined building blocks may be added in the future to support modern application characteristics the run-time system Channels defines an efficient framework for executing these protocols.

The specific target of the project **CogPiT** (**Co**nfi**g**uration of **P**rotocols **i**n **T**IP) is the definition and implementation of a configuration task for protocols, especially for the TEMPO protocol as one example, which is described throughout this final report. Prerequisites for this task are requested QoS requirements of an application and the prevailing status of the underlying network and the run-time system. The result of this configuration task is an application-tailored communication protocol, which is represented in a list of selected protocol building blocks (module configuration) and their interconnections that form the protocol configuration. The second task of downloading these module configurations to Channels into the TIP environment and running the executable protocol is already part of TIP and has been described elsewhere.

2 Introduction

Emerging applications offer a variety of requirements that can be expressed in different Quality-of-Service (QoS) parameters. These QoS have to be provided by the underlying communication subsystem and the network to allow for the sufficient processing of application-supporting protocols. Furthermore, standard protocols offer a rich but fixed set of protocol functionality that is not always acceptable for certain application requirements. *E.g.,* the standard ISO/OSI Transport Protocols, such as TP4 and its service specification [1], [2], neither provide a jitter control functionality nor a jitter QoS parameter to support isochronous traffic adequately. Additionally, since traditional models of protocol processing do not show a well suited behavior [3], [4], solutions for achieving a better behavior have to be developed.

2.1 State of the Art

The reasons for this behavior are twofold. On one hand, services offered on top of traditional end-systems are quite restricted in the sense that only very simple QoS parameters can be specified. Examples can be found in existing transport protocols, *e.g.*, TP4 [1], [2] and TCP [5], offering a limited number of QoS parameters only, *e.g.,* the mentioned jitter parameter is missing, synchronization aspects can not be specified, or security characterizations are not included. On the other hand, performance of traditional end-systems is not only dependent on transport protocols specifications [6], moreover, multiple different factors are involved, *e.g.*, protocol processing in software, redundant protocol functionality in multiple layers, and data copying between different address spaces.

To overcome these misbehaviors various approaches are possible. A categorization of these alternative approaches can be done by distinguishing their main targets [7], while further refining performance issues into implementation-related and architecture-related ones:

- avoiding the service gap between applications and networks,
- avoiding problems of layer-oriented architectures, and
- avoiding the performance bottleneck.

2.2 Solution Approaches

These three different areas of research have been tackled often in separate approaches. Examples for bridging the service gap are the introduction of new services for request/response type of traffic and transaction-oriented traffic [8] or acknowledged datagram services [9]. The 3-component model [10] avoids problems with layered architectures and introduces much broader components to overcome intra-layer communication and to escape multiple sub-layers. Examples for avoiding the performance bottleneck are manifold as well: efficient implementation of protocols ([11], [12], [13], [14], [15] and [16]) on one hand and the development of light-weight protocols, such as the Versatile Message Transaction Protocol (VMTP) [17], the Xpress Transfer Protocol (XTP) [9], PATROCLOS [13], or TEMPO [18] on the other hand are possible. Furthermore, operating system support for protocol processing has been developed resulting in the design of the *x*-kernel [4] or the Channel Server [19].

Nevertheless, an important target of a communication subsystem is to efficiently serve various applications with different requirements on top of a network, using appropriate communication protocols. Therefore, a flexible and dynamic approach has been developed. The *Transport and Internetworking Package (TIP)* offers the degree of necessary flexibility and efficiency to overcome the mentioned misbehaviors. Comparable configuration approaches are the Function-based Communication Subsystem (F-CSS) [20], [21], ADAPTIVE [22] or Da CaPo [23].

2.3 Outline

Throughout this final report the results of the cooperation between Universität Karlsruhe, Institut für Telematik and Siemens AG, ZFE München are documented, regarding various separate problems and issues to meet the project targets as defined above in Section 1. Therefore, this report is organized as follows. Section 3 introduces into the main and important aspects of TIP as far as they are relevant for the configuration task. Besides a brief overview on used terminology the applied transport protocols TEMPO is presented. Conceptual issues on the configuration task are afterwards discussed in Section 4. They encompass general aspects, the selected granularity of building-blocks, Quality-of-Service as well as network and system parameter, and the detailed description of the formal Module Description Language for configuration support. Based on these arguments the design of CogPiT is depicted in Section 5. The developed configuration algorithm is discussed in detail, relevant parameter are outlined, and negotiation schemes of parameters are delineated. Furthermore, Section 6 includes all relevant information on the derived implementation. Finally, Section 7 rates the developed approach, discusses the attained project targets, and gives an outlook onto further task to be done.

Finally, the Appendix includes a list of abbreviations in Section A, a list of synonyms used in CogPiT and F-CSS in Section B, the Extended Backus-Naur Form of the Module Description Language in Section C, information on how to extend current symbolic variables (Section D), an informal method on preparing a correct configuration specification (Section E), and two example excerpts for a formal specification of the configuration basis in F.1 and F.2.

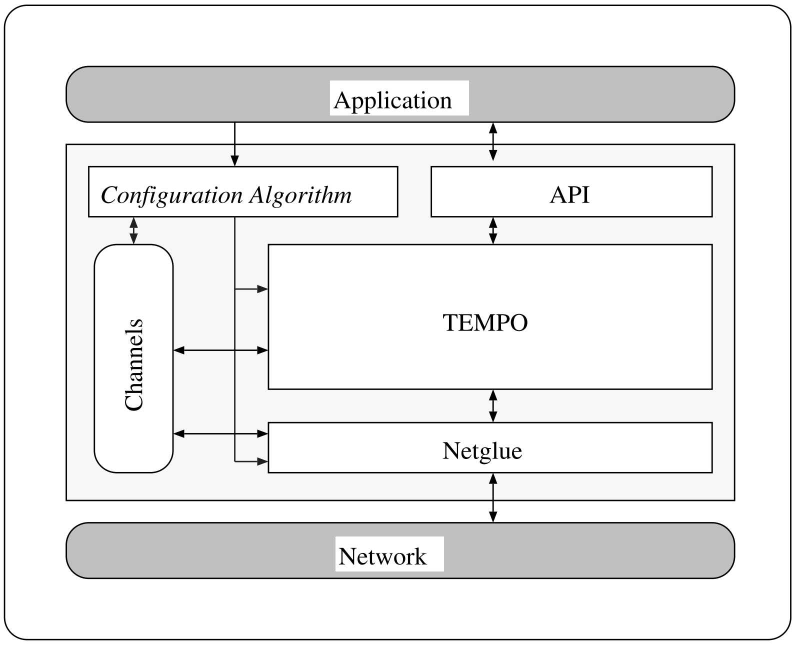

Figure 1: TIP's Architecture

3 TIP — The Transport and Internetworking Environment

As the requested service requirements increase, a new transport protocol and architecture is needed, to satisfy all new services. Therefore, the protocol and architecture called *Transport and Internetworking Package (TIP)* has been developed [24]. As depicted in Figure 1, TIP consists out of the run-time system Channels (cf. Subsection 3.2), the transport protocol TEMPO (cf. Subsection 3.3), Netglue, a subnet convergence sublayer — all besides the developed configuration algorithm COFAL (cf. Section 5) and an Application Programming Interface (API).

Therefore, TIP can be configured using the configuration task developed for this project to provide a specifically requested service according to application demands (such as for video conferencing, traditional FTP, or image browsing) and network prerequisites given by, *e.g.,* ATM [25], FDDI [26], or Ethernet. The scalable functionality of TIP is its main advantage. Besides TEMPO, well-known protocols, such as TCP (Transmission Control Protocol) [27] or IP (Internet Protocol) [28], may be used on top of any of the mentioned networks without any changes to the environment.

A closer look onto TIP unveils a control and user plane, which are responsible for the set-up and release of TIP associations and the user data transfer respectively. Especially for user data transfer the requested communication protocol is configured. Furthermore, associated set-ups and releases may be supported by certain security functionality that can be configured as well.

3.1 Terminology

The following itemization includes relevant terminology used throughout this final report in point-form. It includes brief explanations of each term as well. Associated abbreviations are listed in Appendix A.

- Association — Logical binding between users
- Channel configuration — One of the channel templates, defining the load pattern for the channel server of all configured modules (selected protocol mechanisms), their interconnections in terms of their successor/predecessor relation, and user-provider-child relation
- Channel Server — Run-time system of TIP (in short: Channels)
- Channel template — Either a channel configuration or a module configuration
- COFAL — The configuration algorithm developed for automatically mapping application requested Quality-of-Service onto Network and System Resources to generate appropriate channel templates.
- Configuration task — General term for the task of processing the requested protocol configuration
- Function — cf. protocol function
- MDL specification — Formal specification of the pool of modules called **SPECIFICATION** and the worldgraph called **GRAPHSPECIFICATION** completely
- Mechanism — cf. protocol mechanism
- Module — A protocol mechanism, defining the finest granularity for execution purposes of the protocol.
- Module configuration — One of the channel templates, defining initial parameter values for module internal variables
- Network and System Resources — Definition of special characteristics of a certain network and the end-system that is used for communication purposes
- Parameter — The definition of a specific characteristic of Quality-of-Service or Network and System Resources that can be expressed as a value with a defined unit
- Pool of modules — First part of the MDL specification including all relevant MDL information of a single module
- Protocol configuration — The result of the configuration task in terms of generated channel templates for the use of Channels
- Protocol function — The higher level definition of protocol functionality (in short: function) that may be implemented in several alternative protocol mechanisms
- Protocol mechanism — The lower level definition of protocol functionality (in short: mechanism) that is specified and implemented as a module
- Quality-of-Service — Definition of special characteristics of a certain application that may be translated to transport/network specific parameters
- Sink — Receiver of user data
- Source — Sender of user data
- Stream — cf. transport stream
- Transport Service Data Units — User information to be transferred represented in TIP-specific data formats

- Transport stream — Unidirectional flow of transport service user (in short: stream)
- Worldgraph — Second part of the MDL Specification including all relevant successor/predecessor information between modules to define the correct order of processing modules according to the protocol specification

3.2 The Run-Time System Channels

Channels is the run-time system for protocol processing [19]. It is supposed to be a part of TIP providing an efficient support of multi-media applications. In order to design a new run-time system the following *objectives* have been taken into consideration:

- Providing a basis for the portability of communication protocols.
- Supporting the extensions of existing communication protocols within the same environment.
- Definition of a configurable architecture allowing for the execution of protocol building blocks of different levels of granularity.
- Parallel processing of independent protocol building blocks is required.
- Supporting the real-time capability of modern applications.
- Migration of protocol building blocks into hardware components shall be possible.
- Providing a simple method for developing communication protocols based on libraries and basic protocol operations.
- Allowing for a performance-oriented design including the concepts of threads and light weight context switches.

The architecture of Channels encompasses light weight threads and libraries for communication purposes, such as for timer or messages. The service of Channels is used by clients, such as the user or the network driver. Multiple modules — defining the protocol processing functionality — may be interconnected to form the requested protocol. Standardized messages are exchanged between these modules to communicate between modules. Upcalls and downcalls are processed between a module and its successor and predecessor respectively. Events and control messages are exchanged between child and parent modules, which is used for creating and deleting modules in the run-time system. Finally, an entire collection of specific modules to be extracted is described by three different *channel templates*:

- the "channel configuration",
- the "module configuration", and
- the "run-time configuration".

The mandatory channel configuration includes all relevant information on modules to be downloaded and instantiated as well its interconnection information between other modules. The optional module configuration defines the initial settings of module internal parameters with calculated or default values. Finally, the optional run-time configuration encompasses a list of priorities that are used for defining the run-time ranks of processes within Channels.

3.3 TEMPO — A Transport Protocol

The transport and internetworking protocol TEMPO includes the specification of some protocol mechanisms that are useful to support various application requirements [29], [24]. In summary, these selected mechanisms form a lightweight protocol. Therefore, TEMPO offers a transport service to applications that is configurable as well.

3.3.1 QoS Structure in TEMPO

Quality-of-Service (QoS) is used to specify application requirements that are supported by transport associations within TIP. These transport associations model the logical binding between users. The manner of communication in these types of associations is defined by transport streams that allow for the unidirectional flow of transport service data units (TSDU) from the source (sender) to one or multiple sinks (receiver).

The quality of a single stream may be defined by valid QoS parameters that are defined in the *QoS structure* of TEMPO. Therefore, multiple streams being part of one association may have different QoS values assigned to. General QoS attributes within TEMPO include:

- Reliability,
- Performance,
- Security,
- Costs,
- Synchronization, and
- Priority.

As depicted later in Tables 5, 6, and 7, forty-three different QoS parameters are directly support by TEMPO [29], 8 others have been added for the CogPiT project according to a wider range of application specification possibilities. *E.g.,* the workahead parameter has different meanings for different types of streams. In a continuous stream workahead of the requesting side equals the burstiness, which is the maximum number of TSDUs passed to the provider before their actual time slot. For the indicating side it defines the maximum number of TSDUs accepted from the service provider before their actual indication time. Asynchronous streams regard the workahead parameter as the definition of the window size for the flow control function. Especially, if the formal parameter `DSQ_Workahead` equals zero, than an isochronous type of traffic has been specified, which means that no jitter – or at least a certain maximum may not be exceeded — is allowed.

The maximum throughput of an application may be calculated within the TEMPO QoS structure using the interval parameter and the specification of the maximum TSDU length. Therefore, the definition in a formal sense may look like this:

```
Max_Throughput = DSB_Max_TSDU_Length / DSB_Interval;
```

Furthermore, each transport stream can be classified basically into one of the following four categories:

- Asynchronous streams, as a best effort type of service.
- Continuous streams, as a guaranteed service with soft or hard guarantees.
- Unreliable streams without any loss protection.
- Reliable streams with loss detection and correction.

3.3.2 Functions and Mechanisms in TEMPO

Within the TEMPO protocol a certain functionality is defined, which encompasses segmentation, error control, checksumming, acknowledgements, and flow control. In general according to a two-level hierarchy, all protocol functionality can be ordered into functions and mechanisms. Therefore, Tables 1 and 2 include in an overview five different *uses* of functions and mechanisms in CogPiT, which are explained as follows:

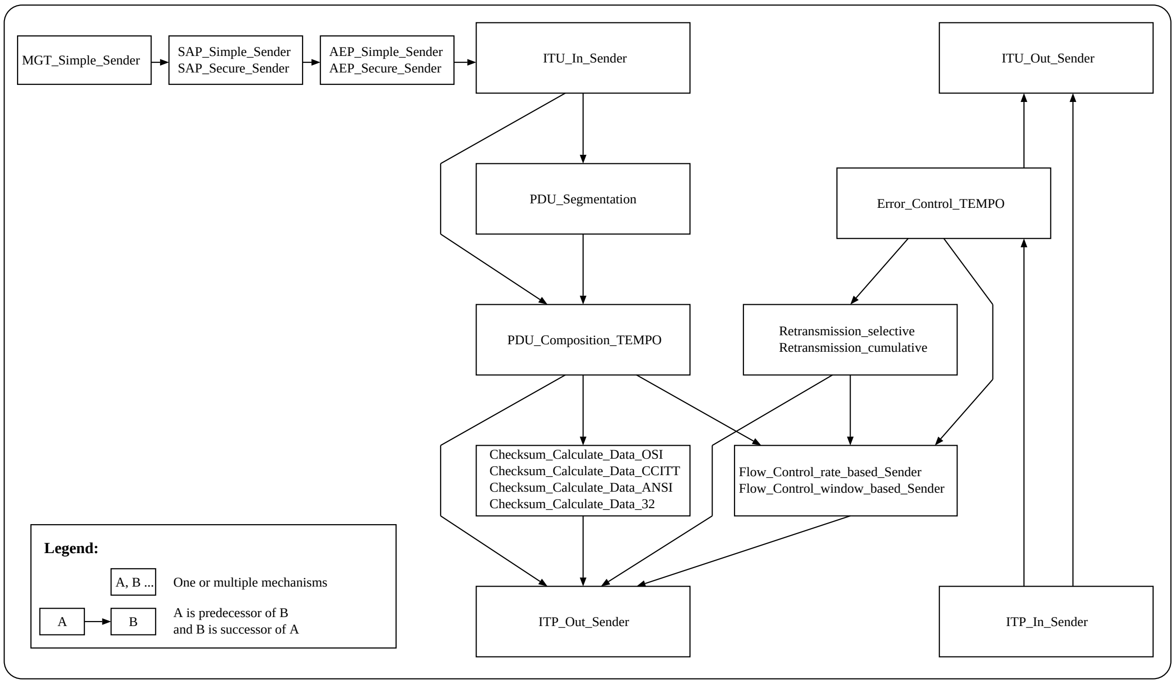

Figure 2: Sender Mechanisms of TEMPO (Source)

- Defined in the TEMPO specification [29]: T
- Considered for configuration purposes based on the functional composition of [30]: C
- In CogPiT included extensions for a wider choice of alternatives for the configuration task (cf. Appendix F.1), which are used for configuration purposes but are not included in the specification, originally: X
- Identification of several other mechanisms for possible future use: F
- Environmental mechanisms used within TIP: E

TEMPO shall be used as the basic for configuration purposes within TIP. Therefore, the mentioned decomposition of the protocol into separate building-blocks is necessary. Relying on the fundamentals of the functional decomposition of [30] the following decomposition of TEMPO, which is useful for configuring different protocols, while distinguishing certain mechanisms (such as various checksumming alternatives), has been developed and specified. Figures 2 and 3 depict the first mechanism/module-grained view in a graphical manner, while the management (MGT), service access point (SAP), and association end point (AEP) functionality of TIP are introduced already. In later Sections of this final report, this view is refined according to configuration and environment needs, such as with the detailed specification and semantics of "interconnections" (arrows), defining the success/predecessor relation of mechanisms, and "mechanisms" as such (nodes), respectively.

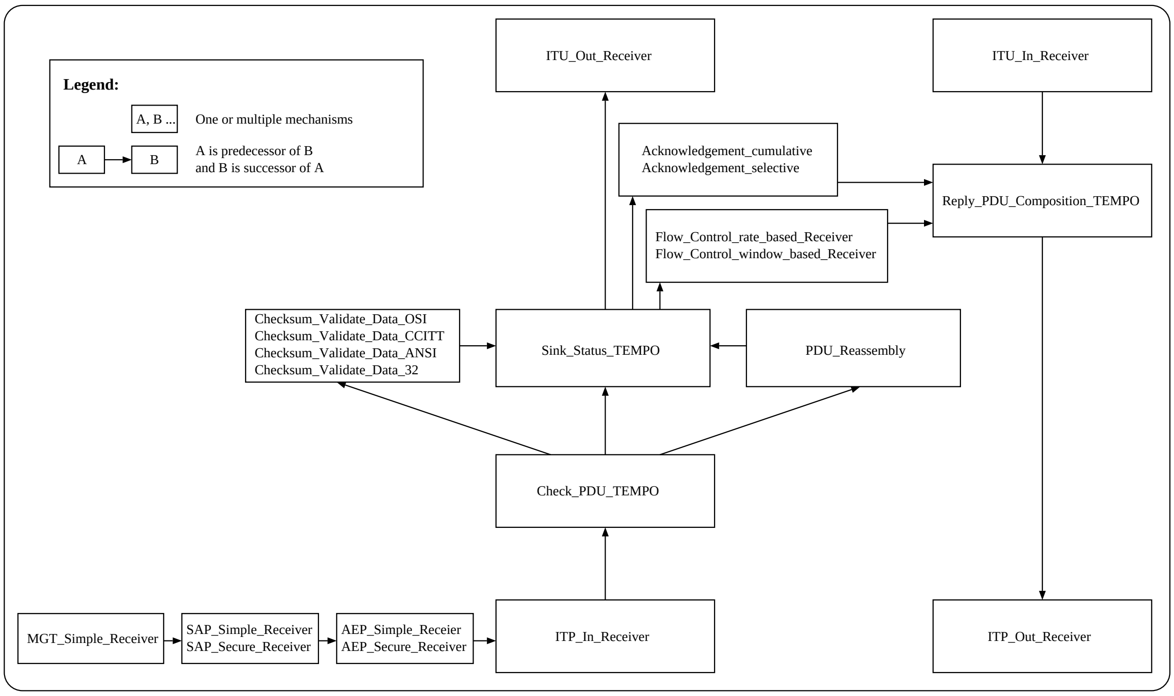

Figure 3: Receiver Mechanisms of TEMPO (Sink)

3.3.3 A Special Functional Decomposition of TIP

As described above, the TIP environment encompasses various features and tasks. The current available implementation of communication protocols at SIEMENS, ZFE did not yet include the fully decomposed TEMPO protocol. Therefore, the entire TIP system as it stands at the moment has been used for demonstration and verifying purposes.

As the Figures 4 and 5 depict, the mechanism/module-grained view has been adopted again. The boldface printed name defines the function's name and additional mechanism names are abbreviated, since they always start with the ASCII sequence of **M_146_254**.

Two parts can be distinguished quite easily. Sender as well as receiver side modules include equally named modules for the Netglue-relevant processing, the specially used Application Programming Interface for SUN machines, and the Service Access Points (SAP) as well as the Association End Points (AEP). These TIP-specific modules are introduced as dummys, since the implementation of the code is not yet complete. They allow for, *e.g.*, for the introduction of various security functions for access control, which are under development.

Finally, Table 3 includes the used — marked in the first column with an *S* for the sender's side and with an *R* for receiver's side — functions and mechanisms for this example in summary.

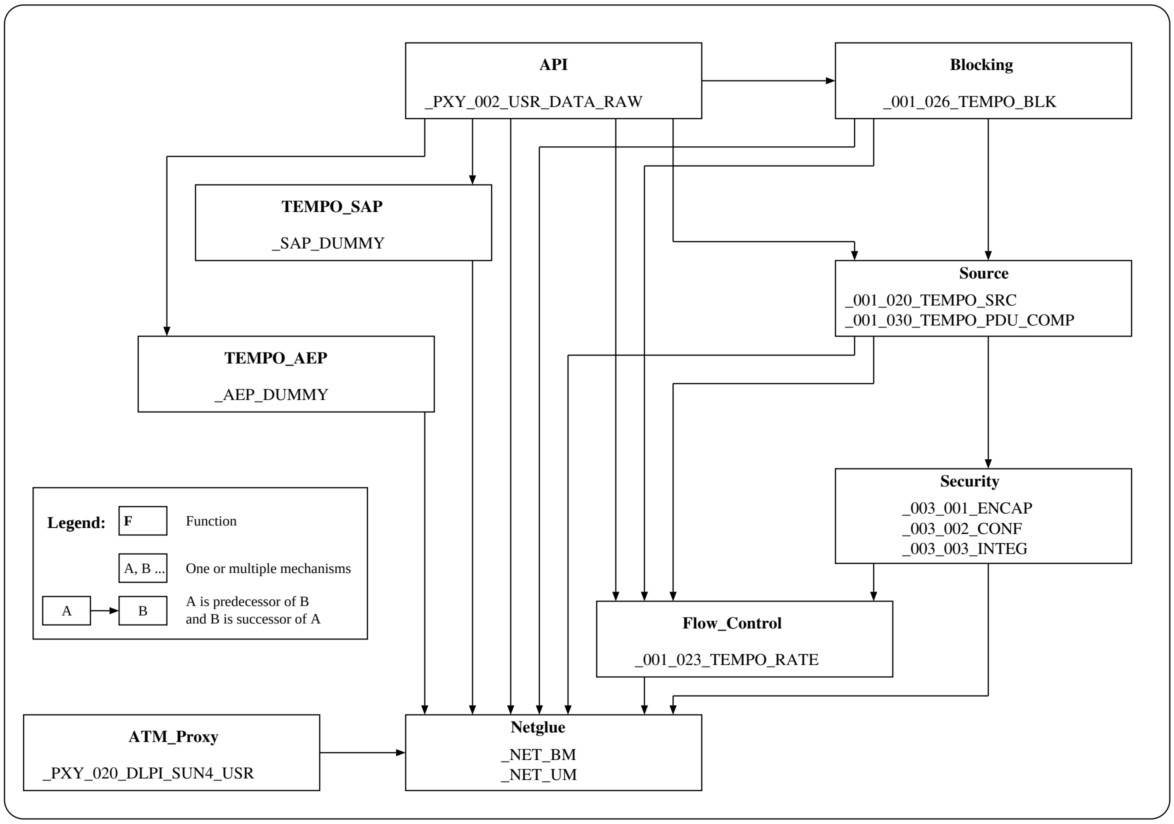

Figure 4: Sender Mechanisms of TIP (Source)

4 The Task of Configuration — Conceptual Issues

Each configurable basic building block — called module, independent of its granularity — has to be specified in terms of relevant information that are important for (1) the configuration task and (2) the instantiation task. The former task of configuration includes the selection of necessary and required modules out of a pool of modules. The second task of instantiation maps the executable modules onto the underlying hardware platform. CogPiT adapts the *Protocol Configuration Manager (PROCOM)* [21], [7], [31] approach of the Function-based Communication Subsystem F-CSS [20] to the special TIP environment [19] (cf. Section 5). In either case, the task of configuring communication protocols demands important prerequisites. There are first of all the granularity questions on the building blocks that have to be configured (cf. Subsection 4.2). Additionally, application requirements have to be expressed within QoS parameters (cf. Subsection 4.3). Furthermore, besides Network and System Resources (NSR) (cf. Subsection 4.4), the specification of every building block according to QoS and NSR consumptions is necessary (cf. Subsection 4.5).

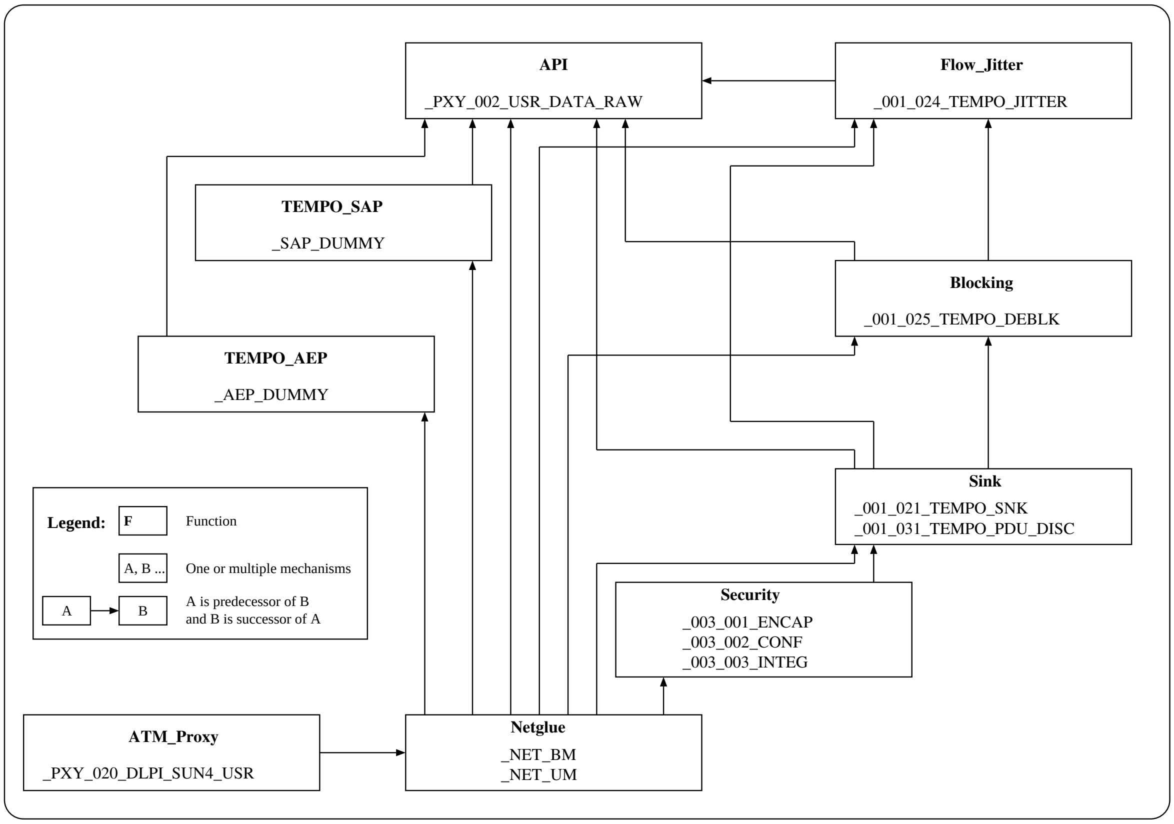

Figure 5: Receiver Mechanisms of TIP (Sink)

4.1 General Aspects of the Configuration Task

General aspects of the configuration task include a brief introduction into the "ancestor root" of CogPiT, a definition of rating criteria for the final evaluation of the developed configuration approach, and its TIP-specific tasks and prerequisites.

CogPiT's "Ancestor Root" PROCOM — Initially, the Protocol Configuration Manager PROCOM approach is explained shortly to introduce the main concept of configuration. Afterwards, enhancements concerning the different environment of CogPiT are defined and substantiated. PROCOM needs a pool of protocol resources. This pool consists at least out of formal descriptions of protocol functions and mechanisms maintained by a Protocol Resource manager. All functions and mechanisms are arranged in a partial order that defines their correct processing order in a protocol machine. This protocol machine represents a configured communication protocol consisting out of separate protocol mechanisms. Furthermore, protocol functions and mechanisms form a two-level hierarchy that is expressed in the formal description, since multiple alternative mechanisms, such as rate-based or window-based flow-control, may implement the same protocol function flow-control. The configuration processed by PROCOM operates additionally on the formal description of existing protocol functions and mechanisms. This is in detail the evaluation of configuration conditions under which a function or mechanism can be configured in the protocol machine to reach the requested Quality-of-Service. Furthermore, certain information of mechanisms, such as required time and memory, are used to calculate values of resource parameters. These parameters are maintained and updated for a complete end-system within a System Resource Manager. Based on the current situation of resource parameters and the evaluated conditions, protocol functions and mechanisms are selected and added to the sender as well as to the receiver protocol machine respectively [21].

TIP-specific Configuration Tasks and Prerequisites — In the TIP environment the configuration task remains principally the same as within PROCOM except for the following enhancements due to TIP:

- Introduction of additional TIP-specific configuration information.
- Separation of successor/predecessor information for mechanisms.

The dedicated TIP run-time system Channels has to be taken care of with certain network and system resource parameters that are specifiable in a formal language for each module. This is necessary for providing information for the second step of the protocol configuration instantiation as well. Therefore, parameters such as the type of a module, parent and childs of modules have been introduced (cf. Subsection 4.5).

The two-level hierarchy between functions and mechanisms (cf. Paragraph 3.3.2) is kept valid, since the configuration task runs faster and, more important, some of the configuration relevant information in TIP is subdivided and simpler specifiable for every mechanism of one protocol functions, such as the type. Additionally, for mechanisms only, such information as user or provider are valid. Concerning the maintainability of a pool of modules, relevant successor/predecessor information defining the partial order of protocol mechanisms per function have been extracted into a separate definition, the worldgraph specification (cf. Paragraph 4.5.2).

An advantage of this refined TIP-approach concerns any additions of new protocol functions and mechanisms, since only each mechanism in the worldgraph definition of successor and predecessor informations have to be revisited and enhanced by the protocol developer. Every successor and predecessor keywords for the regarded new mechanism have to be changed by the protocol developer if an additional protocol mechanism has to be added. Furthermore, the potential predecessors and successors have to be enhanced as well, defining new successors and predecessors respectively. Additions in the pool of modules in the TIP-approach require the addendum of the formal module description for the protocol mechanism. Nevertheless, no changes of existing module specifications within the pool of modules are necessary. Only if the newly introduced modules restrict the use of others, already existing modules, the configuration conditions have to be refined and adapted. The regarded order of module specifications (cf. Paragraph 5.1.1) has to be considered as well.

Rating Criteria for Configuration Tasks — A number of rating criteria have to be defined to allow for the evaluation of advantages and drawbacks of the developed solution. These criteria encompass the following ones:

- Complete definition of protocol functionality,
- Extensive classification of protocol functions,
- Adequate specification of environmental information (QoS and NSR),

- Configuration information completeness,
- Sufficient specification of configuration conditions,
- Efficient configuration process, and mechanisms, and
- Independence of the configuration algorithm from the specified functionality,

They are used within the final process of rating (cf. Subsection 7.2) of the developed solution to identify that the project targets have been met successfully.

4.2 Granularity of Building-Blocks

The granularity of configurable building-blocks has to be defined to allow for a sufficient configuration approach. Coarse-grained building-blocks are complete protocol stacks consisting of protocols that are harmonized one against the other. Examples are the Manufacturing Automation Protocol (MAP), the Technical and Office Protocol (TOP), or the U.S. Government Open System Interconnection Protocol (GOSIP). Obviously, the configuration process of these stacks has taken place off-line, while regarding the execution of protocols as an on-line activity. In these cases, requirements of relevant applications have been identified initially and the existing standard protocols have been plugged together. Any of these protocol stacks do not offer any internal flexibility, since their operational area is very limited.

Regarding complete protocols for a certain layer of the ISO/OSI Basic Reference Model [32], its protocol functionality is fixed, too. *E.g.,* the ISO/OSI Transport Protocol Classes [1] offer functionality depending on the characteristics of the underlying network. Furthermore, the Transmission Control Protocol (TCP) [28], or the VMTP [17] offer a well-defined set of protocol functionality. Again, the inherent flexibility of these protocols is limited, since they do offer a specific service only. An exception in this sense is the XTP [9] protocol, where an application might request some choices for certain protocol functions, such as the flow control for rate-based or window-based mechanisms.

The consequence is that the fine-grained granularity of configurable building-blocks can be reached by protocol functions and mechanisms only. These basic building-blocks represent a certain protocol functionality that can be configured to a complete protocol requested by an application. Examples include protocol functions, *e.g.,* acknowledgement with various mechanisms such as positive, negative, selective, or cumulative acknowledgement, flow control with window-based or rate-based mechanisms, or retransmission with cumulative or selective mechanisms. Each protocol function with a specific mechanism provides a well-defined mini-service that can be combined into a protocol to offer a special application-tailored service. A finally selected alternative of a protocol function is a mechanisms that is called module that can be configured into an executable communication protocol, which is downloaded into the underlying hardware.

4.3 Quality-of-Service Parameters

Secondly, the application requirements have to be expressed within particular parameters. These QoS parameters are used to stimulate the configuration task, while according to particular requirements certain protocol mechanisms may be selected or not. Therefore, QoS parameters guide the configuration task. In case of the CogPiT environment, the QoS structure of TEMPO as introduced in Paragraph 3.3.1 is applied to the configuration, while the formalized parameters are listed in Tables 5, 6, and 7. Considering the task of configuration as an algorithm, QoS parameter and their current valid values act as an input to it.

4.4 Network and System Resource Parameters

Thirdly, the environmental issues are described using Network and System Resource (NSR) parameters. These parameters include information on the current state of the network and the end-system. In the CogPiT case they are maintained by Channels, the run-time system, since it stores all relevant information already. To make them accessible for the configuration task, the formalized NSR have been defined as enlisted in Table 8. Additionally, each of these parameters have to be updated if the situation has changed [1]. These parameters restrict the finally configured protocol according to available resources and act as an input to the task of configuration as well.

4.5 Module Description Language

For configuration purposes protocol functions and mechanisms have to be described as separate modules in a formal language. The formal language used in this case is called *Module Description Language (MDL)*. This description involves conditions that define exactly under which circumstances a module can be configured as part of a communication protocol. Furthermore, information containing interdependencies between modules and environment-specific data as well have been added. These include, *e.g.,* input and output parameters, additional information on user, child, and provider modules, and the type of modules. In MDL these information are included in the first part called **SPECIFICATION** defining the "pool of modules".

Finally, the order of modules to be processed correctly have to be defined. Interconnections between modules can be represented graphically in a *worldgraph*. For maintenance purposes this predecessor/successor-relation is kept separately from all the other information. In MDL these relations are included in the separately defined second part of the MDL specification called **GRAPHSPECIFICATION**.

Within this Subsection all relevant issues of MDL and its important prerequisites are generally discussed and, furthermore, explained with several examples.

4.5.1 Targets of MDL

The Module Description Language (MDL) has been derived and especially tailored for the CogPiT-project with regard to the TIP-environment from the language F-PCL (Function-based Protocol Configuration Language) [21]. MDL in the entire Extended Backus-Naur Form (EBNF) has been added in Appendix C. A formal language is necessary to define entirely configuration information for considered modules uniquely. Main targets of MDL are:

- Simplicity of its syntax,
- Efficiency of its evaluations for configuration purposes,
- Completeness of information needed for the configuration of protocol modules and the run-time environment,
- Block-oriented structure,
- Maintenance friendliness, and

[1] Within CogPiT the NSR parameters are used, but no maintenance is provided, since the Channel Server is responsible for all future resource administration policies. The details are for further study.

```
GRAPHSPECIFICATION  TEMPO_Worldgraph
...
DEFFUNC Segmentation
        {
        REQUIRED  (DSQ_Max_TSDU_Length + PCQ_Max_Header) > NSQ_Max_Data_Unit_Size;
        DEFMECH   PDU_Segmentation
                {
                SUCC PDU_Composition_TEMPO;
                PRED ITU_In_Sender;
                }
        }

DEFFUNC Reassembly
        {
        REQUIRED  (DSQ_Max_TSDU_Length + PCQ_Max_Header) > NSQ_Max_Data_Unit_Size;
        FORBIDDEN DSB_Segmented_Delivery = TRUE;
        DEFMECH   PDU_Reassembly
                {
                SUCC Sink_Status_TEMPO;
                PRED Check_PDU_TEMPO;
                }
        }
...
/* additional function body descriptions */
...
```

Figure 6: An Extract of a **GRAPHSPECIFICATION**

- Optional possibility to derive the formal specification of modules for the **SPECIFICATION** part of a MDL specification from a module's internal description automatically.

4.5.2 "GRAPHSPECIFICATION" of MDL

As described above, the main body of a MDL specification consists out of two distinct parts. Firstly, the **GRAPHSPECIFICATION** includes the specification of the *worldgraph*, which includes the indispensable successor/predecessor information for modules. This relation allows for the correct specification of processing the configured protocol modules in the right order.

The keyword **GRAPHSPECIFICATION** and a name is followed by a list of keywords called **DEFFUNC**, defining function bodies. Inside a function body keywords, such as **REQUIRED**, **FORBIDDEN**, and **DEFMECH** are included. They define the use or non-use of a function and all of its alternative choices in terms of mechanisms. They are handled in the **SPECIFICATION** part below. Further attributes define for each included mechanism — identified via a unique identifier defining the class name — possible successors (**SUCC**) and predecessors (**PRED**) defining the worldgraph.

The **REQUIRED** and **FORBIDDEN** conditions on a function's basis may be specified to define the use or non-use of a certain function under certain circumstances, which is expressed within logical expressions formed by symbolic variables (cf. Paragraph 4.5.4 below). Multiple occurrences of the **DEFMECH** keyword are allowed, which defines alternative mechanisms usable for the same protocol function.

The example in Figure 6 defines the function *Segmentation* being part of the worldgraph. If the **REQUIRED** condition is true after the evaluations of an application request, *Segmentation*

```
SPECIFICATION  TEMPO_Module_Resource_Pool
...
#DEFINE msec 1    /* millisecond */
#DEFINE bytes 8   /* number of bits per byte */
...
DEFMECH PDU_Reassembly
        {
        TYPE      SNK;
        SENDER    FALSE;
        RECEIVER  TRUE;
        REQUIRED  (DSQ_Max_TSDU_Length + PCQ_Max_Header) > NSQ_Max_Data_Unit_Size;
        INPUT     data_buf;
        OUTPUT    data_buf;
        PARAMETER Max_Segment_Length = 512 * bytes;
        USERS     5;
        PROVIDERS 5;
        CHILDS    5;
        TIME      9 * ys;
        MEMORY    400 * bytes;
        }
...
/* additional mechanism body descriptions */
...
```

Figure 7: An Extract of a **SPECIFICATION**

belongs to the protocol configuration. In this case only a single mechanisms called `PDU_Segmentation` is allowed, which has a mechanism `PDU_Composition_TEMPO` as a successor and `ITU_In_Sender` as a predecessor. In general, multiple conditions, mechanisms, successors, and predecessors may be specified as necessary (cf. the EBNF of MDL in Appendix C).

The maintenance of this part of the MDL specification is a little more complicated than the following one of the **SPECIFICATION**, since the interconnections of modules have to be added thoughtfully, determining the interdependencies in terms of the processing order of protocol modules (cf. Paragraph 4.5.7). Additionally, all conditions for the function have to be defined, which can be done by the worldgraph programmer only, since then newly introduced conditions may effect already existing one of the function hierarchy level. Furthermore, the worldgraph may be changed completely [2], but without effecting the pool of modules and its internal details.

4.5.3 "SPECIFICATION" of MDL

Secondly, the **SPECIFICATION** includes details on all mechanisms in the *pool of modules*. This part of the MDL specification can be maintained naturally and derived automatically, since only module internal details are included and no module interconnections are considered. The reasons can be seen in the following description of a pool of modules.

Considering the short example in Figure 7, an outermost block of a MDL pool of modules specification consists out of the initial keyword **SPECIFICATION** followed by a name. Besides a number of numerical constant definitions (**#DEFINE**) the keyword **DEFMECH** defines a mechanism body. The description for mechanisms includes at the most 13 different keywords.

[2] This is the definition of new successor/predecessor relations entirely.

The mechanism body starts with the keyword **DEFMECH**. The name following represents the current module's class name of the protocol mechanism as it has been used in the **GRAPH-SPECIFICATION**. A subsequent **TYPE** parameter identifies the module's type according to the TIP environment. These types include management (MGT), service access point (SAP), application end point (AEP), source (SRC), sink (SNK), proxy (PXY), or initial (INI) type. The next two keywords **SENDER** and **RECEIVER** define, whether the mechanism has to be configured for the sender's or receiver's side of the protocol. Logical expressions are used after the following **REQUIRED** or **FORBIDDEN** keywords as well (multiple occurrences are allowed) specifying the conditions under which the mechanism has to be placed, selected, or omitted. Moreover, input (**INPUT**) and output (**OUTPUT**) parameters are used to specify data or control information that have to be passed to or from the regarded mechanism. The **PARAMETER** keywords defines mechanism internal parameters to be set initially to a certain value (currently of the C-type "double" only) that may be calculated according to an arithmetical expression. A **DEFAULT** value is optional, which defines a to be used value if the expression cannot be evaluated at configuration run-time. Therefore, the **DEFAULT** value has to be a number only.

The **USERS**, **PROVIDERS**, and **CHILDS** keywords define the finite number of users, providers (potential predecessors), and childs (potential successors). These numbers are useful for modules that have the capability of multiplexing between certain incoming and outgoing information. These information are used by the Channel Server to instantiate the relevant modules in the run-time environment correctly. The last two keywords encompass mechanism requirements in terms of system resources. **TIME** specifies the time, which is necessary to process the regarded mechanism under worst case conditions. **MEMORY** defines the relevant memory for the mechanism in terms of buffer and code space. These information are used within the configurator to calculate a principal success of the configuration.

4.5.4 Expressions in MDL

Various types of expressions are supported within MDL. Logical conditions consist out of comparisons of arithmetical expressions and logical expressions that have to be evaluated to `TRUE` or `FALSE`, while using "&" (`AND`), "—" (`OR`), "!" (`NOT`), ">", "<", ">=", "<=", "!=", or "=" as operators and symbolic variables as operands (cf. Paragraph 4.5.5). Depending on the evaluated result further action is undertaken, such as for the **SENDER** or **REQUIRED** keywords.

Arithmetical expressions are defined with "+", "-", "/", or "*" as operators. Each expression may include certain symbolic variables, representing the dependency of the regarded expression from certain Quality-of-Service parameters or Network and System Resource parameters (cf. Subsections 5.2 and 5.3 below). Constants defined in the **#DEFINE** section of the **SPECIFICATION** or **GRAPHSPECIFICATION** may be used additionally. Note that any other "names", *e.g.,* the ones defined after the **PARAMETER** keyword, are regarded as text strings that *cannot* be used in any expressions. A **#DEFINE** might include an arithmetical expression that consists out of previously defined constants only.

It is in the responsibility of the specifier to use correct units per arithmetical or logical expression. Any type checks are not performed before run-time or at run-time of the configuration. Erroneous situations will lead to a failure of the configuration task for the requested communication protocol.

4.5.5 The Concept of Symbolic Variables in MDL

A number of symbolic variables for Quality-of-Service and Network and System Resources have been defined in previous reports [33], [34], [35]. They can be used as attributes for certain keywords in MDL and are instantiated with currently valid values that are requested from an application or stated as valid from the system resources maintained by Channels. As described above, arithmetic and logical expressions are combined out of these symbolic variables and additional constants.

Symbolic variables are heavily used within the configuration task. These variables allow for the definition of conditions, comparisons, and expressions, which are used for the specification of a selection or non-use of a certain protocol building block. For that reason, for every defined QoS parameter in the TEMPO QoS structure that is visible to applications, and every NSR parameter that is necessary to define a specific view of the network and system resource state, a single symbolic variable — syntactically, an ASCII-string — has been defined.

Advantages of the symbolic variable concept are the possibility to specify an arbitrary number of different expressions. The semantical use of these expressions is defined by the corresponding keyword of MDL that comes in front of the expression. Additionally, the independence of the task of configuration — defined by the configuration algorithm in Subsection 5.1 — and the MDL specification is important. This results in the possibility to change specifications easily, while the configuration task itself will stay untouched.

The concept of symbolic variables has two minor disadvantages. That is the use of strings to define the symbolic variable, probably resulting in a slower implementation, while handling strings. Fortunately, this problem has been solved using unique tokens (cf. Paragraph 6.3.4 below). Additionally, the introduction of a new symbolic variable, such as for newly identified QoS or NSR parameters, results in the necessary addition of two very simple functions in the initialization process of the configuration task (cf. Appendix D).

A categorization for each of these symbolic variables has been used to identify each of them by a naming convention. Table 4 includes these categories, defining for each category a prefix, that depicts the belonging of a symbolic variable to a certain category. The full list of defined symbolic variables within CogPiT can be found in Tables 5, 6, 7, and 8 as discussed within Subsections 5.2 and 5.3.

4.5.6 Specification of Conditions

MDL allows for the specification of **REQUIRED** and **FORBIDDEN** conditions. These conditions identify the use or non-use of certain functions and mechanisms in certain situations. The reason for the usefulness of a **FORBIDDEN** keyword is depicted by the example of the function `Reassembly`. The condition of `DSB_Segmented_Delivery = TRUE;` has to be specified with the **FORBIDDEN** keyword, since the "negative" definition of `DSB_Segmented_Delivery` does not equal the used **REQUIRED** condition. Additionally, the definition of a **FORBIDDEN** condition is sometimes a quite simpler approach, than the definition of a complex **REQUIRED** condition. In general, these conditions have to be logical expressions (cf. Paragraph 4.5.4 above and Appendix C) to be evaluated to `TRUE` or `FALSE`. Nevertheless, at least exactly one **REQUIRED** condition has to be specified per mechanism and function body separately.

Since multiple **REQUIRED** and **FORBIDDEN** conditions may occur for a single function or mechanism, the following semantic is applied. In case of n **REQUIRED** conditions R_1 ... R_n at least one of them has to be evaluated to `TRUE` for a successful selection, which is called "OR-

```
(a) ...
    REQUIRED R1 | R2 | R3;
    ...

(b) ...
    REQUIRED R1;
    REQUIRED R2;
    REQUIRED R3;
    ...
```

Figure 8: Similarity of **REQUIRED** Using the "OR-Semantics"

```
(a) ...
    FORBIDDEN F1 & F2 & F3;
    ...

(b) ...
    FORBIDDEN F1;
    FORBIDDEN F2;
    FORBIDDEN F3;
    ...
```

Figure 9: Similarity of **FORBIDDEN** Using the "AND-Semantics"

semantics". Therefore, the specification in Figure 8 (a) and (b) are semantically identical. If none of them is evaluated to `TRUE`, the considered function or mechanism is not part of the calculated protocol configuration.

In case of m **FORBIDDEN** conditions F_1 ... F_n, every single one has to be evaluated to `FALSE` to identify the non-use of the function or mechanism, which is called "AND-semantics". Therefore, the specification in Figure 9 (a) and (b) are semantically identical. If a single **FORBIDDEN** condition is evaluated to `TRUE`, the function or mechanisms is not needed.

Additionally, if one **FORBIDDEN** condition has been evaluated to `TRUE` and one **REQUIRED** condition has been evaluated to `TRUE` as well, the configuration request fails, since an inconsistent request has been issued. This is important to recognize, on one hand, since the use of **FORBIDDEN** conditions may result in inconsistent requests that are not solvable within the configuration task. On the other hand, these conditions allow for the explicit definition of forbidden uses of a function and/or mechanism.

Concerning the **SENDER** and **RECEIVER** keyword, a single condition is allowed only, which has to be evaluated to either `TRUE` or `FALSE`. Depending on the result, the mechanism will be part of the sender's side, the receiver's side, or both sides at the same time.

4.5.7 Specification of Successors and Predecessors

The specification of **SUCC** and **PRED** attributes within the *worldgraph* defines the graph of all potential correct interconnections between separate mechanisms. They are necessary for defining the correct order of processing mechanisms in a configuration. Since some of these interconnections are finally depending on the specific configuration result, the validity of an interconnection in the worldgraph has to be refined in the MDL specification using the specific **IF** attribute. While in Figure 11 the generic situation is depicted textually, Figure 10 includes the graphical representation and the excerpt in Figure 12 includes a real

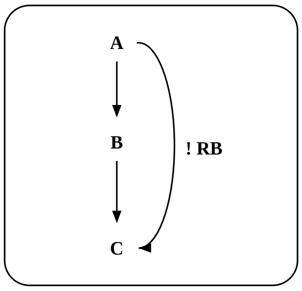

Figure 10: A Generic Example of an Optional Interconnection in a Graphical Representation

```
DEFMECH A
...
SUCC B;
SUCC C IF !RB;
...
DEFMECH B
...
REQUIRED RB;
...
DEFMECH C
...
PRED B;
PRED A IF !RB;
...
```

Figure 11: A Generic Example of an Optional Interconnection in a Textual Representation

example of the TEMPO MDL-Specification, *e.g.,* for a protocol including segmentation (cf. Appendix F.1).

The general example is valid for situations where the direct interconnection between the mechanisms A and C is *optional* [3]. Depending on the configuration of mechanism B the resulting protocol configuration is different. If B is part of the protocol configuration, then the direct interconnection of mechanisms A and C is not part of the protocol configuration. Otherwise, mechanism B is not part of the protocol configuration, but the direct interconnection of mechanisms A and C is. To define these situations within MDL, the **IF** attribute is used after the **SUCC** and **PRED** attribute.

In general for these types of situations, the MDL specification has to be carried out as depicted in Figure 10, since the validity of interconnections depend on the **REQUIRED** conditions of the mechanism B. If B is part of the configuration, than at least one **REQUIRED** condition of B — here called RB entirely — has been evaluated to `TRUE`. Therefore, the existence of this interconnection depends on the negated **REQUIRED** condition of B — called *!RB* — since the non-configuration of B leads to the necessary direct interconnection between A and C. Thus, the correct interconnections between mechanisms can be determined within the configuration task automatically.

Otherwise, if the direct interconnection between A and C is always *mandatory* [4], there

[3] This is referred to as an “OR-semantic” on interconnections of successors and predecessors.
[4] This is referred to as an “AND-semantic” on interconnections of successors and predecessors.

```
GRAPHSPECIFICATION TEMPO_Worldgraph
...
DEFMECH ITU_IN_Sender
...
        SUCC PDU_Segmentation;
        SUCC PDU_Composition_TEMPO IF
             !((DSQ_Max_TSDU_Length + PCQ_Max_Header) > NSQ_Max_Data_Unit_Size);
...
DEFMECH PDU_Segmentation
...
        REQUIRED (DSQ_Max_TSDU_Length + PCQ_Max_Header) > NSQ_Max_Data_Unit_Size;
        SUCC PDU_Composition_TEMPO;
        PRED ITU_In_Sender;
...
DEFMECH PDU_Composition_TEMPO
...
        PRED ITU_In_Sender IF
             !((DSQ_Max_TSDU_Length + PCQ_Max_Header) > NSQ_Max_Data_Unit_Size);
        PRED PDU_Segmentation;
...
```

Figure 12: An Example of an Optional Interconnection Considered as OR-Semantic

is no need for the use of the **IF** attribute. Any non-existing **IF** attribute for either **SUCC** or **PRED** is automatically considered as TRUE, and has not to be specified explicitly. An example within the TEMPO MDL-Specification is included in Figure 13.

The reason for the **IF** attribute becomes additionally clear, if the manner of handling interconnections is encountered. Interconnections are "activated" between mechanisms, if at either side the mechanism is part of the resulting configuration. In the above described situation, the direct interconnections between A and B, and B and C respectively, are "activated", if B is part of the configuration. Otherwise, dangling interconnections of the resulting graph of the configuration are removed. Applying this methodology to the direct interconnection of mechanisms A and C, does not lead to the correct result, since it has been an *optional* interconnection per definition, which is depending on the non-existence of mechanism B. It would not be removed, since both A and C are part of the configuration. Furthermore, this concept of *optional* and *mandatory* interconnections may be applied to longer sequences of mechanisms, such as A, B, C, and D, as well, while the same principal considerations for the **IF** attribute depending on the **REQUIRED** conditions of the mechanisms B **and** C are valid. A special clearance procedure has been designed for considering successors and predecessors only, that are part of the final resulting configuration (cf. Paragraph 6.3.4).

4.5.8 Automatic Generation of a MDL Description

Automatic generation of a module description in MDL from a specification in TIS (Timed Interaction System) [30], which is a LOTOS extension, is impossible in a straight forward approach using the currently valid TIS syntax. Certain extensions in TIS have to be introduced, since some necessary information are missing in TIS but they are indispensably needed in MDL. *E.g.,* this is the definition of explicit successors and predecessors, since the number of predecessors only is not sufficient for configuration purposes. Successors are specified in TIS implicitly only, regarding the name as an unique identifier. All these information

```
GRAPHSPECIFICATION TEMPO_Worldgraph
...
DEFMECH Check_PDU_TEMPO
...
        SUCC Sink_Status_TEMPO, Checksum_Validate_Data_OSI, Checksum_Validate_Data_CCITT,
             Checksum_Validate_Data_ANSI, Checksum_Validate_Data_32;
...
DEFMECH Checksum_Validate_Data_OSI
...
        SUCC Sink_Status_TEMPO
        PRED Check_PDU_TEMPO
...
DEFMECH Sink_Status_TEMPO
...
        PRED Check_PDU_TEMPO, Checksum_Validate_Data_OSI, Checksum_Validate_Data_CCITT,
             Checksum_Validate_Data_ANSI, Checksum_Validate_Data_32;
...
```

Figure 13: An Example of a Mandatory Interconnection Considered as AND Semantic

are absolutely necessary for the worldgraph in MDL to define a correct order of processable modules. But most important, all relevant information on QoS and NSR dependencies of a TIS module — used for the specific configuration conditions — are missing completely. They have to be used for the configuration task to determine relevant interdependencies between parameters and mechanisms, such as via **REQUIRED** and **FORBIDDEN** conditions describe the need for a certain module. The extensions of these information into TIS might be done in terms of additional and mandatory comments including specific keywords.

Furthermore, these extensions could be prepared without changing existing TIS definitions and its processing tools, while a special translator may generate the syntactically and semantically correct MDL specification out of an extended TIS specification.

4.5.9 Separate Sender and Receiver Worldgraphs

One important exception in the specification of worldgraphs for sender and receiver parts has to be dealt with separately. As it has been described in the previous Paragraph 4.5.7, the predecessor and successor attributes are used to define the order of processing between mechanisms. If the use of similar mechanisms on both sides is intended, which are the sender *and* the receiver side at the same time, and the naming of these mechanisms is not different (cf. Paragraph 5.1.1) the unprepared usage of the predecessor and successor attributes is confusing. Therefore, two manners of handling this issue are possible. (1) One solution is to define separate Worldgraphs for the sender and the receiver side respectively. In this case equal names for similar mechanisms may be used on either side and the predecessor/successor relation is still correct. (2) The second alternative encompasses the use of different prefixes for the considered mechanism to allow for the definition of a single worldgraph with a correct predecessor/successor relation.

Considering the following abstract example, these two alternatives are explained briefly. Assume the usage of the mechanism A and B on the sender and receiver side of the protocol at the same time. For alternative (1) the worldgraphs `EXAMPLE_Sender` and `EXAMPLE_Receiver` my include the mechanisms A and B with similar names. Concerning alternative (2), a single worldgraph called `EXAMPLE` includes the mechanisms `A_Sender`, `A_Receiver`, `B_Sender`, and

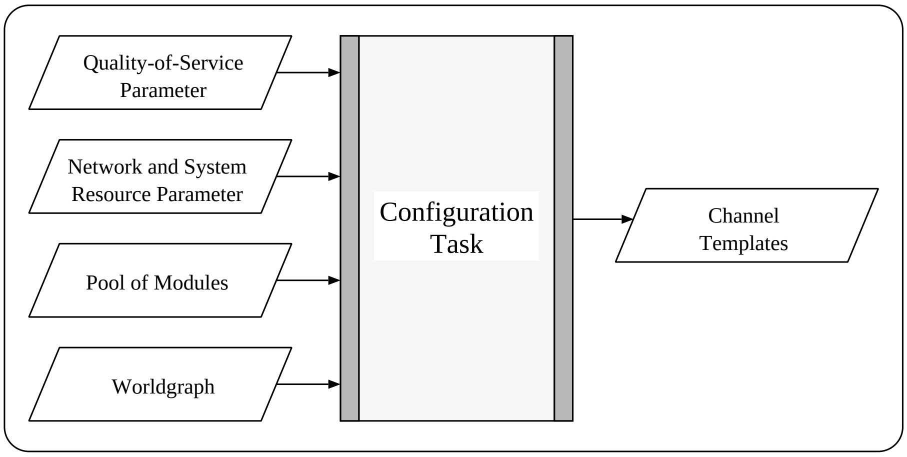

Figure 14: The Configuration Task in an Overview

`B_Receiver`, while the names reflect the location of the mechanism even though the implementation of `A_Sender` and `A_Receiver` as well as `B_Sender` and `B_Receiver` respectively are identical. Obviously, the predecessor/successor relation remains correct in either alternatives. Alternative (1) has been applied to the TIP example pointed out in Paragraph 3.3.3, because of the necessity for similar names for identical modules used in the corresponding implementation.[5]

4.6 The Summarized Configuration

As depicted in Figure 14 the overall view of the configuration task has been summarized in a graphically manner. The inputs to the *Configuration Task* are drawn on the left side, such as the QoS parameter, the NSR parameter, the pool of modules, and the worldgraph. Depending on these information channel templates, drawn on the right side, are generated by the configuration task to represent the configured, application-tailored, and executable communication protocol.

5 The Design of CogPiT

This Section contains the stepwise description of the algorithm developed for fulfilling the configuration task. Additionally, the defined QoS and NSR parameters in their formal shape of symbolic variables are listed. Furthermore, the maintenance of the pool of module and the worldgraph by a protocol resource manager for configuration purposes are presented in concept. Finally, a principal manner of negotiating QoS parameters of channel configurations within multicast scenarios is discussed.

[5] This problem has been avoided within the F-PCL language [21], while including an additional **CODE** attribute per mechanism body, to identify the valid module's name of the implementation, which is completely independent of the naming conventions used within the formal language itself and follows a more strict separation of specification and implementation of mechanisms.

5.1 The Configuration Algorithm COFAL

The configuration algorithm — called COFAL — needs information on application requirements and on resources to be used, which are handed over to COFAL by the application and the local run-time system. Furthermore, COFAL is dependent on the formally specified pool of modules and its included protocol function and mechanism specifications. The task of configuration itself is independent of any underlying hardware architecture used to execute the communication protocol, since the result of a COFAL configuration is a *channel configuration* — a formal representation of a configured protocol consisting out of multiple protocol mechanisms — and a *module configuration* — a set of default values for initial module internal parameter values.

5.1.1 Important Preconditions of the Configuration Algorithm

Main important preconditions for the configuration encompass the following ones that are relevant for the pool of modules, the worldgraph, the QoS and NSR acting as "input" to the algorithm. They have to be observed to guarantee a correct behavior of the configuration algorithm.

Naming Conventions — Every single name used within the pool of modules or the worldgraph has to be part of the symbolic variables re-defined for all QoS or NSR parameter. Additional names are allowed for functions, mechanisms, and constants. The former ones — especially for the mechanisms — have to be identical in the pool of modules and the worldgraph if they represent the same mechanism at the sender's and receiver's side respectively. Otherwise, mechanisms with different names are treated as completely different mechanisms. An initial pre-run-time check is done in the following manner: For every mechanism defined in the function's body of the worldgraph, a mechanism's body hast to appear in the pool of modules. Constants may be used within the pool of module. They have to have different names from all of the ones defined above.

Pool of Modules — Every single module available for configuration purposes has to be part of the pool of modules, while the internal details are included in the mechanisms body according to the syntactical rules defined in the EBNF in Appendix C. Therefore, the following keywords must occur exactly once including a correct value, expression, or condition: **USERS**, **PROVIDERS**, **TYPE**, **CHILDS**, **MEMORY**, and **TIME**. Exactly one of the following two keywords must appear: **SENDER** or **RECEIVER**. At least one of the following keywords must occur: **REQUIRED** or **FORBIDDEN**. One or multiple occurrences of the keyword **INPUT** and **OUTPUT** is required. Completely optional is the keyword **PARAMETER**. A module may be part of the sender's and receiver's side as well. This can be specified, if only syntactically similar module names are required, only if the sender and the receiver are specified within two separate worldgraphs using the similar pool of modules. The necessity for this approach derives from the fact that initially the type of source and sink modules did not occur. Therefore, the evaluation of successor and predecessors in a worldgraph did not regard the sender or receiver attributes of the relevant module. An appropriate enhancement of the implementation can be done, since the conceptual issues do not require any enhancements.

Worldgraph — Every single module of the pool of modules may be included into the worldgraphs functions bodies, which is defined by the **DEFFUNC** keyword. Every body must include at least one of the following keywords: **REQUIRED** or **FORBIDDEN**. One or multiple occurrences of mechanisms definitions **DEFMECH** must appear per function. Inside a mechanism's definition at least one of either **SUCC** or **PRED** must occur. The special unique identifier of a mechanism has to be equal to the one used in the pool of modules, otherwise they are treated as different mechanisms, which may not be intended.

Furthermore, certain restrictions apply to the worldgraph itself, which is specified by the **SUCC** and **PRED** relation. The graph has to be cyclic-free. *E.g.,* the following simple cycle has to be avoided: a mechanism A may not have a mechanism B as its successor, if B has A as a successor. It has to include for sender's and receiver's side separately two dedicated mechanisms for either the upper and lower boundary of the considered world of mechanisms as well. The upper boundary — the interface to the application — is defined as `APP` and the lower boundary — the interface to the network — as `NET` [6]. *E.g.,* in Figure 2 the mechanisms `MGT_Simple_Sender` and `ITU_Out_Sender` define the upper boundary, while `ITP_Out_Sender` and `ITP_In_Sender` define the lower boundary of the sender's side. And in Figure 3 `MGT_Simple_Receiver` and `ITP_Out_Receiver` define the lower boundary, while `ITU_Out_Receiver` and `ITU_In_Receiver` define the upper boundary. One of these mechanisms is always regarded by COFAL as the root node for the sender's and receiver's side, respectively, to start with the configuration task. That is in the mentioned example `MGT_Simple_Sender` for the sender and `MGT_Simple_Receiver` for the receiver. The reasons for the separation of data and control data transfer base in various performance analysis of protocols. Therefore, the separate handling of user and control data allows for the performance-oriented exploitation of protocols, already on the specification level.

Figure 15 depicts the possibilities of valid worldgraphs encountering the following interface modules:

- Mandatory are protocol-inbound and outbound mechanisms for the user-data sending source.
- Mandatory are protocol-inbound and outbound mechanisms for the user-data receiving sink.
- Optional are protocol-inbound or outbound mechanisms for the control-data receiving source.
- Optional are protocol-inbound or outbound mechanisms for the control-data sending sink.

Ordering of Functions and Mechanisms — Finally, the order of the specification of separate mechanisms within the functions bodies effects the selection procedure in terms of alternative choices. The first discovered mechanisms — defined in the function's body — matching the requirements for the sender's and/or the receiver's side will be selected (cf. the corresponding selection step no. 4 of COFAL in Subsection 6.3). Therefore, the most restrictively usable mechanisms have to be specified first. The reason for this implicit ordering is the fact that there is no reliable metrics for selecting one mechanism out of two entirely similar possibilities. A future enhancement may allow for the integration of another step within the configuration algorithm to apply new decision criteria or even backtracking algorithms for defining sub-selections.

[6] These special names have to be used for the relevant successor/predecessor attributes.

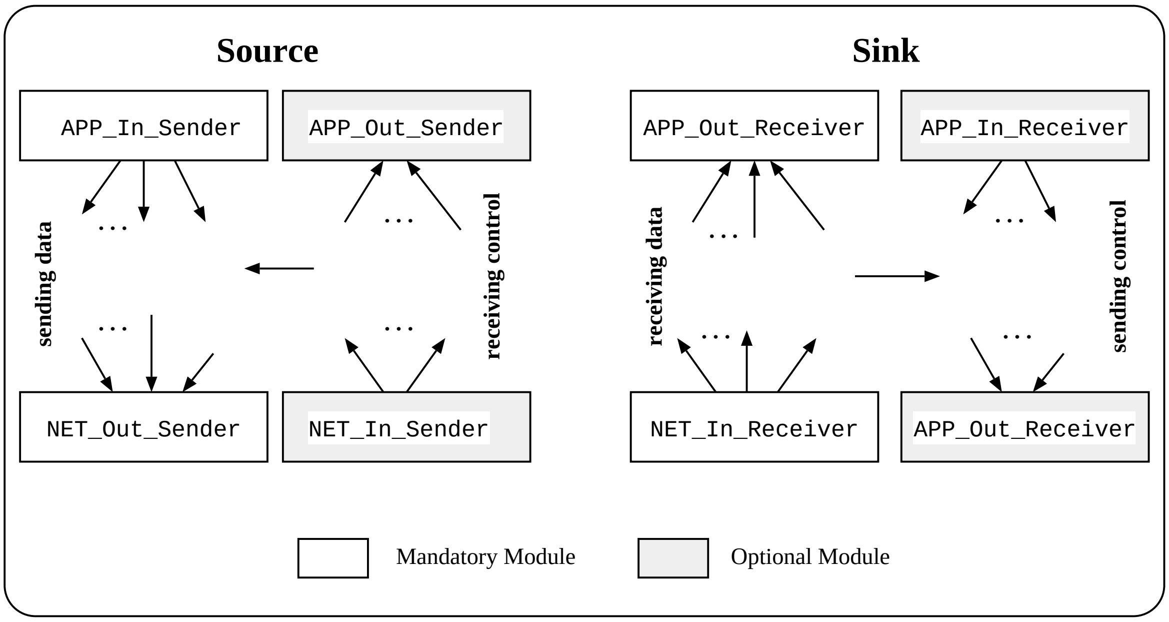

Figure 15: Valid Worldgraph Interfaces to the Application or the Network

5.1.2 Stepwise Description of the Configuration Algorithm

If the mentioned preconditions for the MDL specification in general have been met, the developed configuration Algorithm COFAL may be applied. According to various separate features to be checked, COFAL is designed by eight separate steps. Especially steps 1 to 4 are applied to the MDL specification resulting in two intermediate sets of selected mechanisms, one for the the sender's side the other one for the receiver's side of the configured communication protocol. Steps 5 to 8 are afterwards applied to each of these two sets separately if necessary to allow for the correct calculation for the final channel templates. The final execution of COFAL can be controlled by certain options; one of them is the configuration of the sender's and/or receiver's side of the protocol only (cf. Paragraph 6.3.2). These steps are outlined as follows, while smaller examples are used to clarify certain situations.

Step 1 — Configuration Preparation: Transport stream requirements expressed in terms of QoS parameters are specified by the application. This specification forms one important basis for the selection of protocol functions and mechanisms. All QoS parameter values are associated with certain symbolic variables according to Tables 5, 6, and 7 that are used in the MDL specification for, *e.g.*, defining configuration conditions. An example is the following one, where the maximum delay of the application is set to 15 ms. Another one encompasses the definition of the use of the OSI checksumming algorithm.

```
DSQ_Delay_Max := 15
DSA_Checksum_OSI := TRUE
```

Step 2 — Determination of Available Resources: COFAL requests information about available system and network resources from the local run-time system Channels. An optional request may be handed over to one or multiple remote end-systems of the sinks — depending on the uni- or multicast communication scenario — to receive necessary information as well [7]. The information provided on the first local inquiry are needed for step 4 at least and are made available under symbolic variables, too. Examples are the following ones, where the maximum bandwidth of the local network adapter is set to 100 Mbit/s and for the remote network adapter it is set to 80 Mbit/s.

```
LRQ_Net_Interface_Bandwidth := 100000000
RRQ_Net_Interface_Bandwidth := 80000000
```

Step 3 — Feasibility Check: After having received all necessary information from the application in terms of stream requirements and available system and network resources from Channels, a first thorough analysis of the configuration request is possible. This check of feasibility tests a set of expressions that is used to specify certain pre-conditions to be fulfilled. These pre-conditions are stored separately from the pool of modules, but follow a similar syntax and evaluation as defined for the **REQUIRED**- and **FORBIDDEN**-conditions. Therefore, each symbolic variable for stream QoS and NSR information values may be combined arithmetically or logically — depending on the type of the variable, such as the following example presents.

```
DSQ_Delay_Max > NRQ_Delay_Max;
```

The use of this feasibility check is to avoid unnecessary configuration efforts in particular situations, where it is obvious that a protocol suited for the needs of the application cannot be configured with currently available resources. An obvious situation for a configuration failure is the maximum acceptable delay for an application `DSQ_Delay_Max`, which should always be greater than the maximum delay of the network `NRQ_Delay_Max`.

The available network bandwidth `LRQ_Host_Interface_Bandwidth`, which might be too low to provide the required maximum application throughput requested by the user `DSQ_Workahead / DSQ_Interval` multiplied by an estimated overhead factor (`Overhead_Factor`) for the future protocol machine.

```
(DSQ_Workahead / DSQ_Interval) * Overhead_Factor <=
LRQ_Host_Interface_Bandwidth;
```

The configuration will be continued only, if the evaluation of all specified pre-conditions indicate a positive result.

Step 4 — Selection of Protocol Functions and Mechanisms: The selection of required protocol functions and mechanisms is defined as a two-stage process. For each function in the worldgraph and pool of modules the need for using it in the actual protocol is

[7] Beyond the scope of this project is the description of the use of a signaling protocol between COFAL and one or multiple remote end-systems. Additional protocols, such as an extension of the ATM signaling protocols Q.2931 [36] or UNI 3.1 [37] may be used for sending one or multiple remote resource requests and transferring the answer(s) back to the initiating COFAL. The specific support of TIP for handling local and remote resource requests is for further study, since the originally used type of system resource manager within F-CSS based on different prerequisites has not been requested for being integrated into TIP. The reason for that is the fact that the run-time system Channels implements all relevant functionality already (cf. Subsection 4.4).

determined by evaluating **REQUIRED**- and **FORBIDDEN**-conditions for each protocol function and mechanism as well. If COFAL detects within the worldgraph that a certain protocol function is required for the current protocol — which is logically similar to the evaluation of at least one **REQUIRED**-condition to `TRUE` and all **FORBIDDEN**-conditions to `FALSE` —, an afterward selection must be made among available mechanisms of this protocol function, using additional conditions in the mechanisms body of the MDL specification. The final selections amongst mechanisms are collected in two separate sets, the `SENDER`-set and the `RECEIVER`-set, respectively, which is dependent on the **SENDER** and **RECEIVER** attributes (cf. Subsection 6.3 for further details). As an example the following expression depicts the **REQUIRED**-condition of a segmentation function:

```
REQUIRED  (DSQ_Max_TSDU_Length + PCQ_Max_Header) > NSQ_Max_Data_Unit_Size;
```

Only if the sum of the maximum length of any application data unit `DSQ_Max_TSDU_Length` and the maximum header length of the protocol data unit `PCQ_Max_Header` is greater than the maximum supported data unit length of the network `NSQ_Max_Data_Unit_Size`, a segmentation function is necessary. A second example depicts the configuration conditions for a checksum validation function. Considering Paragraph 4.5.6, these conditions may be specified using a single occurrence of the keywords, while applying the "AND"-semantics and "OR"-semantics, respectively.

```
REQUIRED   DSQ_Bit_Error_Rate = 0 | DSB_Damage_Notify | DSB_Damage_Delete;
REQUIRED   DSB_Damage_Correct | DSB_Loss_Notify;
REQUIRED   NSQ_Bit_Data_Corrupt > DSQ_Bit_Error_Rate;
FORBIDDEN  DSB_Damage_Ignore;
FORBIDDEN  DSB_Loss_Ignore;
```

Various **REQUIRED** conditions define the circumstances under which the use of checksumming is intended. That is, *e.g.,* the request for no bit error rates, or the deletion of damages, which can be only done, if the damages are detected first. Explicit situations for not applying the checksumming function are in the case of ignoring damages or losses.

Step 5 — Protocol Configuration Construction: Once the selection of protocol functions and mechanisms is completed, COFAL constructs channel configurations for the sender and receiver side separately. These protocol configurations describe internal communication relations between selected protocol mechanisms. In this step of the configuration task the **SUCC** (successor) and **PRED** (predecessor) expressions of the protocol mechanism description in a MDL description are used. These expressions determine the relevant processing order of all selected mechanisms as described in Paragraph 4.5.7.

The final result of step 5 is an internal configuration graph of a protocol configuration. Since multiple successors and predecessors respectively may be defined per mechanism with the **SUCC** and **PRED** keywords, the final configuration graph may not have "open" successors or predecessors. *E.g.,* an "open" mechanism A is identified by (1) not being part of the resulting configuration and (2) being referenced (either as a successor or predecessor) from a mechanism B that is part of the configuration. The *clearance* of the configuration graph will be done, using (1) the **IF** conditions for the activation of valid successor/predecessor interconnections and (2) the fact that every successor/predecessor interconnection will be discarded automatically, if either the successor or predecessor is not part of the final configuration.

Additionally, within step 5 the longest path from the mandatory protocol-inbound to the outbound mechanism for the sender's side and the mandatory protocol-inbound to the outbound mechanism for the receiver's side are calculated, using a CPM method. This longest path is defined by the sum of the **TIME** attributes of every mechanism on the longest path. The resulting value specifies under worst case circumstances the time (a maximum delay), a user-data packet needs to be processed within the configured protocol.

Finally, the required resources for the entire protocol configuration in terms of memory, as it has been specified with the **MEMORY** attribute, are calculated.

Step 6 — Protocol Configuration Consistency Check: A consistency check is required to ensure that neighboring protocol mechanisms of a protocol configuration have compatible **INPUT**- and **OUTPUT**-parameters as specified in the MDL description. The graph is considered to be correct if each input parameter for every mechanism may be found in the output parameter list of one of the direct predecessors. If these requirements are not met completely, the constructed configuration graph might not function properly, since left open input parameters will never receive a real incoming value. Therefore, in this case the configuration approach must be considered as a failure. This check may be regarded as optional (cf. Paragraph 6.3.2) for TIP, since specific TIP messages are used for the exchange of information between modules, which are already defined including error handling within the modules.

Additionally, a number of specified post-conditions are considered. They syntactically equal the pre-conditions as defined in Step 3, but include semantical information that is available after the protocol configuration construction, only, such as checking the needed complete memory with available system resources. The configuration will be continued only if the evaluation of all specified post-conditions indicate a positive result. Examples for a post-condition are the following ones, where the test for available memory in the local end-system takes place, and the calculated maximum throughput of the protocol can be supported by the net interface's bandwidth.

```
PCQ_Memory_Needs_Send < LRQ_Avail_Memory;

NSQ_Max_Data_Unit_Size / DSQ_Interval <= LRQ_Net_Interface_Bandwidth;
```

Step 7 — Parameter Evaluation: The internal behavior of protocol mechanisms may be influenced by parameters, defined after the **PARAMETER** keyword. This allows for the specification of, *e.g.*, buffer sizes or initial window sizes, while using arithmetical expressions including all different kinds of symbolic variables and identifiers for constants as well. The actual value may to be calculated by COFAL or defined by a default value. As an example the initial window size of a window-based flow control mechanisms may be used as follows:

```
PARAMETER Flow_Control_Window_Size = 8;
```

Step 8 — Generation and Consistency Check of Channel Templates: Finally, COFAL generates Channel templates, such as the syntactically and semantically correct "channel configuration" and the "module configuration", defining the configured communication protocol — including the protocol configuration and local administration information — to allow for the local instantiation of the executable modules within Channels. Examples are included in the implementation description in Paragraph 6.3.5.

5.2 Relevant QoS Parameter

The following QoS parameters are required and have to be used within the TEMPO QoS structure, since they form the main basis for configuration support. The development of a usable and efficient configuration support within TIP covers the definition of symbolic variables that are used in the automated mapping of QoS parameters, in close cooperation with NSR prerequisites onto protocol building blocks.

The QoS structure in TEMPO has been defined in [18] and introduced briefly in Subsection 3.3. Therefore, Tables 5, 6, and 7 depict for every QoS parameter the defined symbolic variable and the valid unit, such as ms, byte, or boolean. Nevertheless, these symbolic variables may not be regarded as finally sufficient due to incomplete specifications of the QoS structure of TEMPO. Therefore, some useful additions are included in Table 7. Further extensions are possible if needed.[8]

5.3 Relevant NSR Parameter

The network and system resources (NSR) as enlisted in Table 8 are required, since they form a second basis — besides the QoS structure of the TEMPO protocol — for configuration support. The development of a usable and efficient configuration support within TIP covers the definition of symbolic variables that are used in the automated mapping of NSR prerequisites, in close cooperation to QoS-parameters, onto protocol building blocks.

Network and System Resources (NSR) in TIP depend on the currently used network interfaces and adapters and on the run-time environment Channels. These NSR parameters depict, on one hand, currently valid situations of end-systems and network conditions, such as the current bit error rate on the outgoing link or the actual memory usage of the end-system, and on the other hand, certain capabilities of networks and the end-system, such as direct support of isochronous traffic by the network or sequenced processing of data units in Channels internally.

NSR specifications, maintenance, or even monitoring support has not been defined in TIP yet. Therefore, the following approach and recommendation include parameter descriptions that are necessary and useful. It is important to note that a very limited minimal set of NSR parameters is absolutely necessary to allow for the automated task of configuring protocols, since network and system resources limit the range of valid and useful decisions, which protocol building blocks have to be selected according to the provided network services.

The NSR parameters are not complete yet due to a missing specification of any network and system resources in TIP. Additions are possible any time, when new requirements are detected to specify certain circumstances of NSR. Therefore, evaluations of the Channels-environment have to lead to conceptual models that include the possibilities to specify, maintain, and monitor these types of parameters at run-time of TIP. This is necessary to allow for the update of current situations of the network's and end-systems's states that will be taken into account for newly to be configures protocol configurations.

Especially the Protocol Configuration parameters (`PCx_`) allow for the final mapping of the configured protocol onto end-system resources. They are used, *e.g.,* as part of post-conditions

[8] According to recent information on security, the list of added QoS parameters has to be extended. Especially boolean symbolic variables for confidentiality, integrity, and encapsulation have to be defined additionally to the already existing ones, specifying certain algorithms and modes. These parameters have not been part of the originally received TEMPO QoS structure, but they are now necessary for the new security semantics applying to functions and mechanisms as well as to its used algorithms and modes (cf. Table 7).

for the final configuration task. Furthermore, NSR parameters such as the `LRQ_Net_Interface_Bandwidth` are used to define pre-conditions for initial checks of the feasibility of the requested protocol.

5.4 The Protocol Resource Manager

The *Protocol Resource Manager (PROREM)* is responsible for the maintenance of the MDL specification to be of any use for COFAL. This is in detail, a simple syntax checking of two separate files containing the **GRAPHSPECIFICATION** and **SPECIFICATION**, a reasonable checking of conformity of function and mechanism names in both of the files, and a checking of correctly used keywords.

These two parts have been defined separately to specify a syntactical segregation of information that can be generated automatically in parts. The source of the pool of module information may not be maintained manually, but the relevant information may be filled in by a "testing" algorithm. *E.g.,* the need for memory in the end-system could be measured in the run-time system, while instantiating the relevant module as a prototype. Therefore, the automatic generation of main parts of the pool of modules is possible. The worldgraph can not be maintained automatically, but has to be designed carefully, including the successor and predecessor definitions and the **REQUIRED** and **FORBIDDEN** conditions (cf. Paragraph 4.5.8).

5.5 Negotiation Schemes for QoS Parameter Values

The need for exchanging configuration information between peers is quite obvious. As mentioned earlier, the exchange of the configuration result by means of a channel configuration and a module configuration is not part of the project. Nevertheless, a user-to-user signaling protocol may be used. Unfortunately, this is not sufficient, since a negotiation between peers have to be allowed for. Various reasons may be encountered for, that is an adaptation of certain details according to special requirements of peer within a multicast scenario, an initial setting of windows sizes for flow-control or congestion-control functions, definitions of buffer allocation strategies and sizes on the run-time system level, or QoS-parameter values in general. Related work can be found for ideas and definitions of negotiation procedures in [38] and ongoing standardization work in [39] and [40].

Therefore, a basic scenario has been developed [41] that defines two opposite approaches called

- optimistic negotiation and
- pessimistic negotiation.

This scenario takes into account that multicast communications are considered as important for future applications. Basic prerequisites encompass

1. the manner of addressing group members,
2. the quantity of members within the group, and
3. the priority of each member.

A classification of peer members within a communication is done according to the defined behavior of each member. That is the facility of a *right to vote.* Having the right to vote means being an *active peer*, otherwise being a *passive peer.* In general the classification can be done

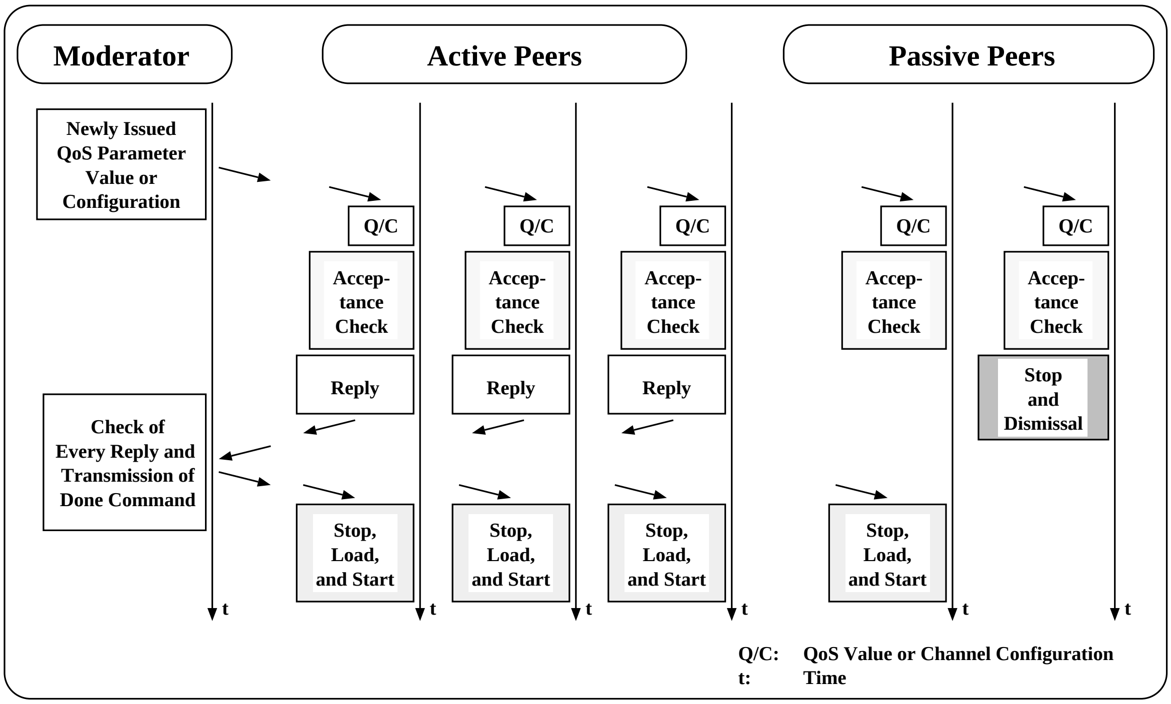

Figure 16: Optimistic Negotiation Scheme

according to Table 9. This classification relies on the reality, concerning a panel discussion. All participants of a panel discussion are either "listeners" (passive peers) or "panelists" (active peers). Listeners are known only by the name of the panel discussion (a group address) and do not have a fixed set of members. In contrary, panelists are known by name (single address) and are fixed at begin of the panel discussion. Mapping this scenario over MBone [42], this type of multimedia-application has been structured in a sufficient manner using passive and active peers.

Based on these assumptions the opposite negotiation schemes are discussed, while assuming that the so-called *moderator* is always an active peer.

Optimistic Negotiation — The basic assumption of an optimistic negotiation scheme is the fact that a moderator will receive QoS parameter values or configuration results that match every peer's requirements. According to Figure 16, the moderator calculates a value and distributes it to every peer. An acceptance check will be run at every peer. Active peers transmit a reply, passive peers do not. If all active peer replies include a positive answer, the moderator initiates a done command containing the agreed upon QoS parameter values or configuration results. Passive peers do not reply and wait for the done command in a positive case or dismiss themselves from the communication in a negative case.

Pessimistic Negotiation — Instead of a quite similar assumption of common requests from all peers, the pessimistic negotiation scheme collects from every active peer local information on resources or QoS parameter values to be used for calculating a moderator-driven conclusion.

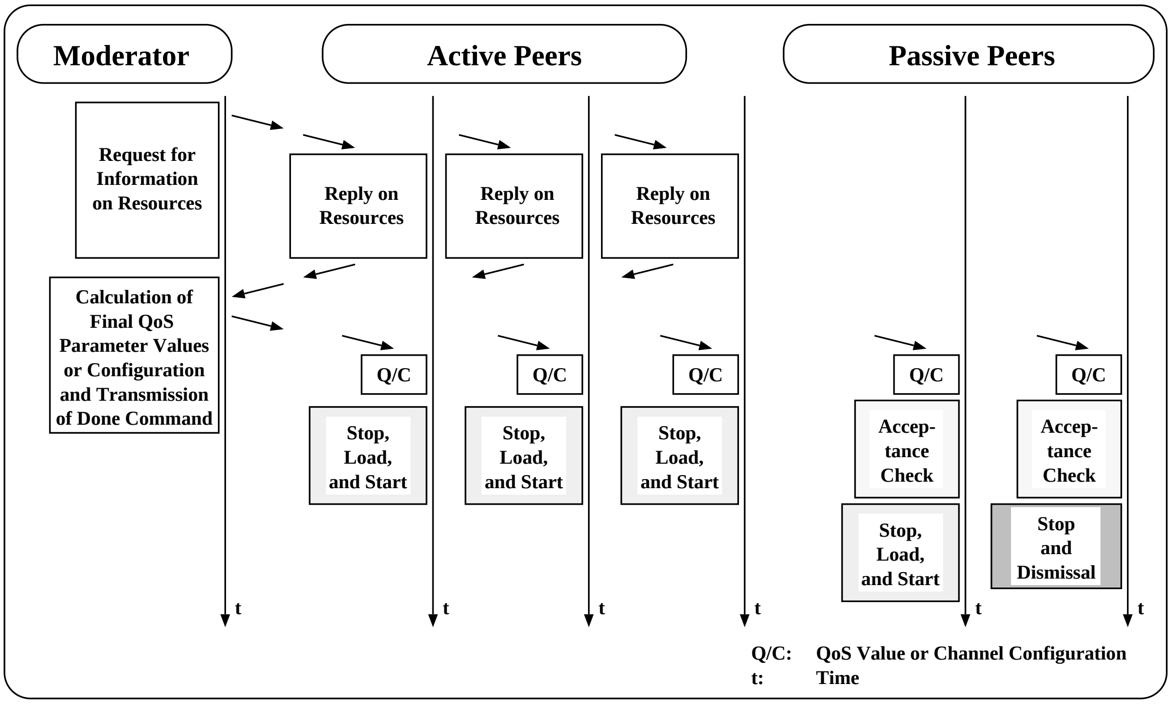

Figure 17: Pessimistic Negotiation Scheme

This result will be distributed afterwards to every active and passive peer and, since a common agreement already has been reached, active peers accept the result for initiating their local configuration or set-up the new configuration. Passive peers act differently, while accepting the result after a positive acceptance check or dismissing themselves otherwise.

Either peer has to implement both alternative negotiation schemes, since in the case of failure of the optimistic approach the pessimistic one has to be used afterwards to calculate a final result. Both schemes need to have a centralized moderator. This does not limit the usage of these approaches, because the final result has to be unique for every participating active peer. Any decentralized or distributed approach would need a large number of two-by-two negotiations that cause a large and unsuitable effort.

6 Implementation of CogPiT

This Section covers main issues of the prototypical implementation of the entire configuration task, its prerequisites, and the configuration algorithm COFAL in greater detail. The intention is not to deliver a complete specification of all relevant implementation details, but to outline main and important features for the user of COFAL. Additional information on the specific implementation can be found in the source code, where in a file header relevant information are included and in a class header specific information are summarized. Generally, the source code is documented and uses of variables and structs are explained briefly. The implementation language is C++, while the compiler used is gcc in version 2.5.8 for SUN machines.

6.1 General Aspects

The implementation of CogPiT has been done in a modularized manner. For easy testing and evaluation purposes a test user and a resource administration have been implemented separately (cf. Subsection 6.2). In the current version of COFAL, pre-defined, but variable initialization files are read and the stored information are used to stimulate a configuration (cf. Subsection 6.3). Before the integration of CogPiT in the entire TIP environment CogPiT is reliably executable and generates channel and module configurations according to Channels requirements (cf. Subsection 3.2).

6.2 A Test User and Resource Administration

For initial configuration purposes, a simple test user and a resource administration have been developed. They manage the relevant parameter values and its data structures used within COFAL.

QoS Requirements — The `user` is used for specifying QoS requirements of a potential application in an object, that is used in a similar manner from the configuration task. Therefore, an instance of a C++ object called `SP_StreamDefinition` includes the relevant data fields for the TEMPO QoS structure. This object is stored in a binary file, that may be handled with a unique file name, such as `ds001` or `ds015`, for various testing scenarios. This file name is requested by COFAL and the contents is being read for stimulating the configuration, which is implemented in the source code file `asi_incl.cxx`.

NSR Prerequisites — Since no resource monitoring and update of network and system resource parameters is performed within CogPiT, Channels has to be extended later on. But for a useful testing scenario valid NSR parameter values are needed to check the configuration itself or valid pre- and post-conditions. Therefore, the resource administration called `sys_sim` has been implemented in a similar manner as the `user`. In a valid binary file local or remote NSR parameter values are stored to be used for the configuration task. The relevant input routines for handling these type of resources preliminarily within COFAL are included in the file `sys_incl.cxx`.

6.3 The Configuration

In a straight forward manner the MDL parser is presented within this Subsection, delivering the necessary data structures and its contents for COFAL. Design issues of COFAL are depicted if some complex decisions have to be dealt within the configuration task. The result of COFAL in terms of generated files is explained.

6.3.1 The MDL Parser

Constants, symbolic variables for QoS and NSR, and every necessary condition specified within the MDL specification is maintained by the Protocol Resource Manager (cf. Subsection 5.4), implementing the MDL parsing functionality. PROREM offers a prototypical implementation of these tasks, including some obvious error messages and hints on the currently handled MDL specifications.

Classes and Functions — The complete description of the protocol resources described in the above mentioned files is included in one single object of the class `PROTOCOL_RESOURCE` being part of the Protocol Resource Manager's implementation. While defining a new object of this class using the standard C++- operator `new`, its constructor reads the entire **GRAPHSPECIFICATION** and **GRAPHSPECIFICATION**. Therefore, no other function call is needed. If any error occurs, all dynamic allocated data structures are destroyed and an error report is given.

Since a detailed documentation is included in any of the source code files `prorem.cxx` and `prorem.hxx`, a short overview of the class components is presented, only. The class component `PROCOM` is defined as a list of protocol functions that are described in the pool of modules and the worldgraph. In case of an error, `NULL` is returned. The component `Const` contains the list of constants defined in pool of modules. The error codes `err` are defined in the source code file `prorem.hxx` and might be redefined for other purposes. Its values are set to 0 (value of constant `NO_ERROR`), if no error occured during object initialization. Only in this case, the object was completely defined and can be used by COFAL. In case of error (`err` is not equal to 0) the constant `ERROR_BASE` has to have the value of the lowest error code and the constant `ERROR_COUNT` has to include the value of the number of error codes. The array of error messages `error` defines an error code for each message. In case of error, additional information, *e.g.,* the name of the function, where the error occured, is written into the class component `add_info`. The array of file description `files` contains information of the files, where the worldgraph and the pool of modules specifications are found or the line number, `PROREM` is currently processing. Therefore, errors can be easily located in the MDL specification. In case of using more than two files, new files can be easily included by incrementing the size of this array. Currently, 0 denotes the pool of module description and 1 identifies the worldgraph. Finally, the class component status describes several phases of the object initialization.

Various functions have been implemented to process the required tasks. Their prototypes are defined in the file `prorem.hxx`. A more detailed description of the functions is given in `prorem.cxx`.

Implementation Remarks on the Memory Management — For dynamic allocation of memory, PROREM uses the functions `mem_alloc` and `mem_free`, according to Channels 1.0 manual. These functions are not part of the class definition and can be redefined as necessary. They use the standard C-functions `malloc` and `free`. This can be easily redefined by simply changing the definition of these functions in the source code file named `prorem.cxx` or by deleting the function-definitions, if `mem_alloc` and `mem_free` are defined as global functions. Nevertheless, `mem_alloc` and `mem_free` have to be defined, because PROREM uses them for local memory management.

6.3.2 The Configuration Initialization File

Important initial settings of the current activities of the configuration algorithm are defined in the file `init.pro`. As depicted in the correct order within Figure 18, comments are enclosed in "%". Further initializations define separate specification files (which are read at initialization time of CogPiT once) and options as well.

E.g., the current valid definitions of symbolic variables that have equivalent functions implemented in CogPiT are summarized as ASCII-names in the file `s_ds_spec.dat`. The MDL specification for the pool of modules is contained the file `pf_pool.dat`, while the worldgraph

```
% An Example for an Initialization File of COFAL
%
% SIEMENS CogPiT Project
%
% Data for Initialization of COFAL
%
% Date:   17.03.1995
% Author: Burkhard Stiller, Joachim Elstner, Kai Schubert
%
DS-SPECS:           s_ds_spec.dat
PFP:                pf_pool.dat
PFG:                pf_graph.dat
PRECONDITIONS:      precond.dat
POSTCONDITIONS:     postcond.dat
LOCAL RESSOURCES:   l1
REMOTE RESSOURCES:  r1
SENDER-PM:          No
RECEIVER-PM:        Yes
CONSISTENCY CHECKS: No
```

Figure 18: The Initialization File of the Configuration Algorithm

is specified in `pf_graph.dat`. Relevant pre-conditions are stored in `precond.dat` and post-conditions in `postcond.dat`. As mentioned earlier, the resource management is still simulated, since Channels does not provide a sufficient interface yet, local and remote resources are pre-set with values for every defined NSR symbolic variable in files `l1` for local ones and `r1` for remote ones. Finally, options may be selected for starting the configuration algorithm for the sender's side (`SENDER-PM`) and/or receiver's side. A "No" will invalidate the option, any other text string — normally "Yes" — will set the option to be required. The consistency checks of the specified **INPUT** or **OUTPUT** attributes may be omitted (Step 6(a) in Paragraph 5.1.2).

Especially, the options have to be set carefully, since the final result may not be correct if, *e.g.,* the sender's worldgraph is specified in the `pf_graph.dat` file, but the `SENDER-PM` option has been set to `No` and the `RECEIVER-PM` option has been set to `Yes`.

6.3.3 The Initialization of Symbolic Variables

Initially, all relevant symbolic variables to be of possible use within the configuration process are defined and initialized. That happens in the source code file `deftoken.cxx`. Depending on the nature of a symbolic variable, it is included in different arrays for associating a unique number, which can be dealt with quite easy. This process is called tokenization.

Additionally, the default initialization takes place for all qualitative symbolic variables. `DSB_` variables are initialized with `TRUE` while `DSA_` variables are initialized with `FALSE` (cf. Table 4). Any further values for system resources and quantitative variables are read from the corresponding initialization files described above.

6.3.4 The Configuration Algorithm

The implementation of the configuration algorithm involves the main task of the project. Therefore, important aspects are explained briefly throughout this Paragraph, while further information and details may be retrieved from [21] or the corresponding source code files.

Based on 8 steps, as described in Paragraph 5.1.2, the implementation followed a straight forward approach. Certain classes have been defined [9] for manipulating the pool of modules (`pfp.cxx, pfp.hxx`), the intermediate result of a channel configuration (`pm.cxx, pm.hxx`), single functions (`pf.cxx, pf.hxx`), and separate mechanisms (`pfm.cxx, pfm.hxx`). The configuration algorithm itself has been implemented in the files `procom.cxx` and `procom.hxx`. For each step, a separate well-documented part has been implemented using the input procedures of the Protocol Resource Manager and the data structures for all relevant QoS and NSR information. In `asi.cxx` and `asi.hxx` only a few structures are used, these are the ones containing the application requested QoS parameter values on a per stream basis.

Additionally, the applied concept of tokens has to be mentioned. As it can be seen, *e.g.,* **REQUIRED** conditions have to evaluated at run-time of the configuration task. They may consist out of many separate symbolic variables and certain logical and arithmetical operators. To allow for a fast manipulation of these expressions, every single part of them is "tokenized" that is the association of a unique integer value — the token — per separate part of an expression, such as a variable, an operator, or a constant. These tokens are stored and maintained in a *token table*. Every single expression in a worldgraph or the pool of modules will be tokenized before run-time of the configuration task, and afterwards at configuration time handled in a tokenized manner. This allows for a quite efficient manner of handling complex expressions.

Obviously, step 4 includes the crucial part of selecting relevant protocol functions and mechanisms for the configured communication protocol. A detailed flow chart for all corresponding evaluation and selection steps can be found in [21] on page 97. After this selection procedure the used worldgraph lacks on purpose some entries in terms of "missing" mechanisms — these are the non-selected mechanisms. But still, some selected mechanism may have successor or predecessor information pointing to these non-selected mechanisms. These dangling references will be cleared out in a clearance step. Unfortunately, the **IF** construct is for **SUCC** *and* **PRED** in case of optional interconnections necessary. The reason for that lies in the current implementation of the clearance procedure just mentioned. This may be changed in the future for simplicity reasons.

Within step 7 the **PARAMETER** evaluation takes place. According to possible multiple occurrences of the same mechanism within one final resulting channel configuration, a specifically named parameter, called `Number_of_Modules`, has been introduced. The evaluation of `Number_of_Modules = 2` leads to a duplication of this specific mechanism in the final channel configuration.

6.3.5 The Configuration Result

As mentioned earlier, channel templates are generated in a well defined form to define an output format of the configuration result. In Figure 19 a generated example of a *Channel Configuration* is depicted. This specific example is based on the TIP example as enlisted in Appendix F.2. The currently used format is defined as *FORMAT 1*. The result is based on the initial application QoS request `ds001.snd` and includes at the sender's side 9 modules. For every module the *class_id* is listed in the first column. Valid parents (*pnt*), *users*, predecessors (*pdrs*), and *childs* are listed additionally in the following columns. The column

[9] The following information in brackets depict the relevant source code file names.

```
FORMAT 1
ds001.snd //9
// class_id                        pnt users pdrs  childs    mech implicit index
API:M_146_254_PXY_003_USR_DATA_RAW -1  -1    8|2|1 -1     // 3    0
M_146_254_SAP_DUMMY                -1  0     3     -1     // 4    1
M_146_254_AEP_DUMMY                -1  0     3     -1     // 5    2
M_146_254_NET_BM                   -1  2|1|7 4     -1     // 7    3
M_146_254_PXY_020_DLPI_SUN4_USR    -1  3     -1    -1     // 13   4
M_146_254_003_003_CONF             -1  6     7     -1     // 15   5
M_146_254_001_020_TEMPO_SRC        -1  8     5     -1     // 10   6
M_146_254_001_023_TEMPO_RATE       -1  5     3     -1     // 8    7
M_146_254_001_025_TEMPO_BLK        -1  0     6     -1     // 12   8
```

Figure 19: Channel Configuration — A Generated Example

mech and *implicit index* are used for documentation only, defining the originally used numbering of protocol mechanisms within the configuration task and the implicit numbering of this result, corresponding to the following lines of the channel configuration. *E.g.,* the implicit index 2 identifies the module `M_146_254_AEP_DUMMY`, additionally as the former protocol mechanism 5. These implicit indices are used for defining the interconnections of modules, *e.g.,* `M_146_254_NET_BM` has the users 2 (`M_146_254_AEP_DUMMY`), 1 (`M_146_254_SAP_DUMMY`), and 7 (`M_146_254_TEMPO_RATE`) and a predecessor 4 (`M_146_254_PXY_020_DLPI_SUN4_USR`). A "-1" defines that there is no interconnection or information used for that entry. Summarizing, the channel configuration contains the configuration result in terms of a graph description based on the originally used worldgraph (such as the **PRED** and **SUCC** informations) including the bidirectional specification of interconnections between every interconnected module.

The corresponding module configuration for this example is depicted in Figure 20. After the identical format, QoS request, and sender's side specification, for every configured module — here again 9 — a list of valid internally used parameters are specified, according to the MDL keyword **PARAMETER**. Based on the similar implicit numbering scheme — valid within one single configuration result only — the TIP run-time environment Channels receives for the module `M_146_254_NET_BM` the value of *3.000000* for the first parameter and the value of *2.000000* for the second parameter. Again, an implicit ordering scheme is applied, defining the order of used parameters directly, corresponding to the MDL specification within the pool of modules; in this example it is for module number 3 (`M_146_254_NET_BM`) the `i_aal_type` for the first and the `i_dataflow` for the second parameter.

Finally, the run-time configuration is for further study, since it can not be generated automatically due to the lack of information within the current version of MDL. A suitable set of priorities for selected modules needs further investigations on algorithmical solutions.

6.3.6 Specific Aspects of the Current Implementation

In the current implementation of CogPiT some minor but relevant issues are included, that should be changed in the future, to define a clear view. Especially, the use of the specific parameter `Number_of_Modules` used after the keyword **PARAMETER** is used for multicast purposes. Within the final step 8 of COFAL, every configured module is being checked for this specific parameter to allow a duplication of the considered module according to

```
FORMAT 1
ds001.snd
VOID
VOID
VOID
M_146_254_NET_BM                  3.000000 2.000000
VOID
M_146_254_003_003_CONF            4.000000 5.000000
M_146_254_001_020_TEMPO_SRC       5.000000 10.000000 512.000000 10.000000 60.000000
M_146_254_001_023_TEMPO_RATE      1.000000
M_146_254_001_025_TEMPO_BLK       2048.000000 5.000000
```

Figure 20: Module Configuration — A Generated Example

multicast requirements. Currently, the use of this mechanism within the TIP example (cf. Appendix F.2) is limited to the security modules. Unfortunately, the TEMPO QoS structure does not define any QoS parameter for requesting multicast features, *e.g.,* the number of potential sinks. Therefore, the use of this specific parameter has to be maintained manually within the pool of module description. Furthermore, the current solution has been introduced to show that is is feasible, but because of the late arrivals of this requirement, the correct calculation of memory requirements as defined after the **MEMORY** keyword has not been implemented yet. A careful analysis of this situation and others as well has to lead to an integration of multiple occurrences of modules in an earlier step of COFAL.

6.4 Examples of Initial Performance Evaluations

On a per step-basis according to Paragraph 5.1.2 some initial performance measurements have been done. The results are included for 5 scenarios in Tables 10 and 11, being executed on a SUN machine. Additionally, the performance numbers (in μsec) are separated for dealing with sender and receiver side tasks separately. This allows for the clear view of times spent for calculating the two distinct parts of a communication protocol. In this example, the **RECEIVER** option has been set to `No`, excluding the configuration of the receiver's side of the protocol. Therefore, the receiver columns in Tables 10 and 11 are quite similar for every example. Obviously, if different options would have been selected the times would change.

As it can be seen, the steps no. 1, 2, 3, and 6 (both) are of a similar behavior, since the task did not change for the measurements. For step 1 the application QoS request file of always the same length has to be read and included into the token table. Within step 2, always an equivalent resource request has been handled. The feasibility check as step 3 is similar due to the unchanged pre-conditions to be checked. Obviously, if the number of pre-conditions will increase or decrease, the corresponding processing time behaves accordingly. Finally, the check of defined post-conditions within step 6 remains unchanged, if the post-conditions themselves remain unchanged.

The important step 4 is responsible for the selection of necessary mechanisms used for the future protocol. Depending on the number of to be checked functions in the worldgraph, a basic portion of time is always required, processing and evaluating each of the function's **REQUIRED** and **FORBIDDEN** conditions. Obviously, this time depends on the complexity of the condition itself as well, such as multiple logical, arithmetical, and/or comparative operators. Additionally, the processing time for evaluating mechanisms' conditions have to be added. This portion varies depending on the number of mechanisms to be looked at in the current situation and on the conditions themselves. Therefore, the processing times for the examples `ds010` and `ds015` are quite bigger, since the number of mechanisms for the final protocol is bigger as in the previous examples. As a basis for the examples, the worldgraph specification according to Appendix F.1 and the corresponding pool of modules have been used.

Step 5 is completely dependent on the number of selected mechanisms within the sender and receiver part. Additionally, the number of successors and predecessors play a major role. As it can be seen in Tables 10 and 11, processing times increase for larger numbers of mechanisms to be interconnected. Furthermore, the consistency checks performed on the configured protocol depend on the defined input and output parameters of every mechanism. Linked with the number of specified internally used parameters the parameter evaluation Step 7 requires different processing times. Finally, the generation of the channel templates is depending on the finally selected mechanisms within the configuration result and its interconnections.

7 Summary, Rating, and Conclusions

7.1 Summary

In summary, this final report **CogPiT — Configuration of Protocols in TIP** stated briefly the need for a configuration concept of protocols to overcome the processing and service gap between emerging applications and offered low performance services from existing communication subsystems. Basic ideas and terminology has been introduced due to an unique taxonomy based on TIP, F-CSS, and other related approaches.

The main approach of configuring communication protocols depending on application requirements led to the definition of protocol modules that reflect certain protocol tasks to be processed including certain specifically described features. The necessity of a formal language expressing the interdependency of protocol modules and their capabilities has been concluded. The adaptation of the concept of configuration onto the TIP-specific requirements has been discussed. An informal definition of the semantics of the language MDL and its capabilities has been described with short individual examples to demonstrate the usefulness of the developed approach. The syntactically correct grammar of MDL has been added in the Appendix.

The configuration algorithm COFAL, embedded into the suitable framework TIP has been presented. COFAL configures communication protocols reflecting dedicated application requirements. Prerequisites encompass a application service interface as defined by TEMPO and it is expressed in terms of Quality-of-Service parameters. The separation of the configuration task (algorithm) and configuration condition specifications (in MDL) leads to a flexible system approach, which may be adapted and extended in a convenient manner. Therefore, on one hand, the use of new MDL specifications is simple. On the other hand, enhancements or changes of COFAL may be done without effecting already available MDL specifications.

7.2 Rating the Configuration within CogPiT

The rating of the configuration algorithm COFAL and the module description language MDL is based on the project targets (cf. Section 1), previously established rating criteria (cf. Subsection 4.1), and special targets for MDL (cf. Paragraph 4.5.1).

7.2.1 Evaluation of COFAL

As it can be seen the definition of protocol resources can be done within CogPiT for protocol functions and mechanisms. The specification of configuration criteria can be done using certain constructs in the formal language (cf. Section 4.5). These conditions called **REQUIRED** and **FORBIDDEN** conditions respectively are evaluated operating with valid application requirements, end-system resource parameter values, and network conditions expressed in special parameters called symbolic variables. Based on these Boolean evaluations a protocol function and mechanism is selected for the required channel configuration. Therefore, a complete definition of relevant protocol functionality is possible.

All relevant configuration information is defined for COFAL and the task of configuring protocol machines. These information, such as **SENDER** or **RECEIVER** attributes, the conditions, **INPUT** and **OUTPUT** parameter, and predecessor- and successor-relations, are sufficient and complete for configuring a channel configuration.

This concept is not limited to a special hardware-environment, but it needs detailed information on the hardware-environment. The developed configuration process is efficient as the initial performance evaluations show (compare results on a different platform as well [21]), since the formal restrictions on the resource pool have been kept small and short. Furthermore, a straight forward configuration algorithm is based on these pre-conditions. This is supported by the two-level hierarchy of protocol functions and mechanisms that allow for the two-step selection of functions first and afterwards its special mechanism. This is the reason for the classification of protocol functions and mechanisms in two categories.

Finally, an adequate specification of environmental information can be done using the network and system resource parameter as well as the Quality-of-Service parameter. These information depict the main input for COFAL allowing for the configuration of an application-tailored communication protocol.

7.2.2 Evaluation of MDL

Concerning the rating criteria as defined in Section 4.1, the definition of the *Module Description Language (MDL)* is rated. First of all, the definition of separate modules is possible in MDL. Each single module can be classified accordingly. The mechanisms itself are specified in the module body, where all relevant details are included. This two level hierarchy directly implies the concept of object-oriented programming for protocol functionalities regarding the protocol function's bodies as the base class and the mechanism body (module) as the derived class.

Any information describing the application requirements in terms of Quality-of-Service parameters are used within conditions. The specification of configuration conditions offers twofold possibilities: **REQUIRED** conditions define the situation in which the specific mechanism can be used, whereas **FORBIDDEN** conditions define the non-use of a mechanism. Both types of conditions offer a much wider range of specifications, since the inconsistency of requests are explicitly visible.

A number of relevant information are accessible via a MDL specification. These details assure the completeness of configuration information within MDL. The just mentioned conditions form a main part for configuration purposes. Other details include information for the Channel Server in the TIP-environment. The worldgraph specification elements are absolutely needed for the order of processing modules. Finally, the two-level hierarchy of protocol functions and mechanisms has been mentioned as part of the MDL specification. It forms the basis for configuring communication protocols dependent on application requirements, since the concept of functions defines direct alternative choices of mechanisms per function.

7.3 Conclusions

Concluding this work, the presented algorithm COFAL embedded in CogPiT, the prototypical implementation, and the preliminary performance results as well show a reasonable behavior of the system components. This includes acceptable processing times dependent on defined configuration conditions and a defined MDL specification. Therefore, *the project targets according to the cooperation contract have been met successfully.* Additional features of upcoming new requirements during the cooperation have been clearly identified and solution approaches within the conceptual work have been suggested. A detailed list of these new features — that have to be dealt within another refined and enhanced implementation — are listed in the following outlook in Subsection 7.4.

7.4 Outlook

The following itemization encompasses newly arisen requirements of COFAL, identified within the final steps of the cooperation that are handled conceptually, but are not included in the prototypical implementation yet. Additionally, conceptual enhancements are listed that might be of interest for future versions of CogPiT.

- The automatic generation of a MDL specification from a TIS (Timed Interaction System) specification is possible via a translator for the pool of modules but not for the worldgraph itself, if relevant information are added into a TIS specification initially.
- The automatic generation of the "run-time configuration", may be possible if necessary details are additionally included in the MDL specification.
- A future enhancement of COFAL may allow for the integration of another step within the configuration algorithm to apply new decision criteria besides the currently used **REQUIRED** conditions, which might allow for the abolition of the relevant order of mechanisms in the worldgraph per considered function.
- Additional features may be implemented admitting for a macro notation of multiple occurrences of conditions or, generally spoken, logical and arithmetical expressions. An example might be a new MDL keyword **SHORT** and the following syntax (**#SHORT** <name> = <text>, which represents a macro notation for the following: <name> will be substituted exactly syntactically by <text> at any occurrence within the **SPECIFICATION** or **GRAPHSPECIFICATION**). This approach involves a certain kind of template format.
- The types of the **PARAMETER** keyword of MDL are at the moment doubles only. This may be enhanced to integer or strings etc. that can be used for different purposes (type binding). An easy integration of the type "int" only, *e.g.,* for a parameter, may be done using naming conventions, such as `i_...` and `d_...`, that allow for module internal type checking.
- The use of variables defined besides the keyword **PARAMETER** within any specification of attributes seems to be helpful for complex expressions to become better readable.

- A refinement of Step 4 of COFAL for an additional metric to distinguish between two, up to now similar modules concerning the selection. The integration of backtracking algorithms for resetting some sub-solutions may introduce fine-grained selection alternatives but encompasses more time-consuming calculations.
- The introduction of additional symbolic variables may be done without re-compilation of the source code using pre-defined patterns that have been added preliminarily in advance.

8 Acknowledgements

Very many thanks go to Joachim Elstner and Kai Schubert, who contributed as part of their HIWI activity lots of implementation, porting, debugging, and testing work in the CogPiT-project [10]. They have been of great and reliable support to fulfill the scheduled deadlines and tasks. Additionally, Per Vindeby and Vera Seidel did their best in a quite restricted time in terms of providing an insight into the TIP environment and Channels as well. Martina Zitterbart managed the local affairs of the project in Karlsruhe, since I am currently on leave from the Institut für Telematik on a PostDoc position at the University of Cambridge, Computer Laboratory, Systems Research Group, England. Therefore, I had to use an extensive amount of e-mails, talks, and ftp-bandwidth to do certain developments for CogPiT at the "remote" place of Cambridge, for that I am very grateful as well.

[10] Responsibilities of possible errors included in this final report cannot be taken. Additionally, there is no liability on any of the developed programs. All rights reserved on ideas, programs, and testing scenarios as agreed upon in the cooperation contract.

Use	Protocol Function	Protocol Mechanism
C T	Segmentation	PDU_Segmentation
C T	Reassembly	PDU_Reassembly
C T	PDU_Composition	PDU_Composition_TEMPO
C T		Reply_Composition_TEMPO
C T	Checksum_Calculate	Checksum_Calculate_Data_OSI
C T		Checksum_Calculate_Data_CCITT
C T		Checksum_Calculate_Data_ANSI
C T		Checksum_Calculate_Data_32
F		Checksum_Calculate_Header
C T	Checksum_Validate	Checksum_Validate_Data_OSI
C T		Checksum_Validate_Data_CCITT
C T		Checksum_Validate_Data_ANSI
C T		Checksum_Validate_Data_32
F		Checksum_Validate_Header
C T	Flow_Control—Sender	Flow_Control_rate_based_Sender
X		Flow_Control_window_based_Sender
C T	Flow_Control_Receiver	Flow_Control_rate_based_Receiver
X		Flow_Control_window_based_Receiver
F		Stop_and_Go
C T	Error_Control	Error_Control_TEMPO
F		Discard_PDU
C T	Check_PDU	Check_PDU_TEMPO
C T	Sink_Status	Sink_Status_TEMPO
C T	Acknowledgement	Acknowledgement_selective
X		Acknowledgement_cumulative
F	Retransmission	Go_Back_N
F		Automatic_Repeat_Request
T		Timer_Driven_Retransmission
X		Retransmission_selective
C T		Retransmission_cumulative
T		Forward_Error_Correction
F	Synchronization	Inter_Stream_Synchronization
F		Intra_Stream_Synchronization

Table 1: TEMPO's Protocol Functionality (Part 1)

Use	Protocol Function	Protocol Mechanism
C T	InterfaceToUser_In_Sender	ITU_In_Sender
C T	InterfaceToUser_In_Receiver	ITU_In_Receiver
C T	InterfaceToUser_Out_Sender	ITU_Out_Sender
C T	InterfaceToUser_Out_Receiver	ITU_Out_Receiver
C T	InterfaceToProvider_In_Sender	ITP_In_Sender
C T	InterfaceToProvider_In_Receiver	ITP_In_Receiver
C T	InterfaceToProvider_Out_Sender	ITP_Out_Sender
C T	InterfaceToProvider_Out_Receiver	ITP_Out_Receiver
E	Management_Sender	MGT_Simple_Sender
E	Management_Receiver	MGT_Simple_Sender
E E	Service_Access_Point_Sender	SAP_Simple_Sender SAP_Secure_Sender
E E	Service_Access_Point_Receiver	SAP_Simple_Receiver SAP_Secure_Receiver
E E	Association_End_Point_Sender	AEP_Simple_Sender AEP_Secure_Sender
E	Association_End_Point_Receiver	AEP_Simple_Receiver
E		AEP_Secure_Receiver

Table 2: TEMPO's Protocol Functionality (Part 2)

Use	**Protocol Function**	**Protocol Mechanism**
	API	
S, R		M_146_254_PXY_002_USR_DATA_RAW
	TEMPO_SAP	
S, R		M_146_254_SAP_DUMMY
	TEMPO_AEP	
S, R		M_146_254_AEP_DUMMY
	Source	
S		M_146_254_TEMPO_SRC
S		M_146_254_TEMPO_PDU_COMP
	Sink	
R		M_146_254_TEMPO_SNK
R		M_146_254_TEMPO_PDU_DISC
	Blocking	
S		M_146_254_001_026_TEMPO_BLK
R		M_146_254_001_025_TEMPO_DEBLK
	Flow_Control	
S		M_146_254_TEMPO_RATE
	Jitter_Control	
R		M_146_254_TEMPO_JITTER
	Security	
S, R		M_146_254_003_001_ENCAP
S, R		M_146_254_003_003_CONF
S, R		M_146_254_003_003_INTEG
	Netglue	
S, R		M_146_254_NET_BM
S, R		M_146_254_NET_UM
	ATM_Proxy	
S, R		M_146_254_PXY_020_DLPI_SUN4_USR

Table 3: TIP's Protocol Functionality

Prefix	Description	Example
	Data Stream Requirements	
DSQ_	Requirement of a **D**ata **S**tream with **Q**uantitative values.	DSQ_Delay
DSB_	Requirement of a **D**ata **S**tream with **B**oolean values.	DSQ_Damage_Ignore
DSA_	Requirement of a **D**ata **S**tream with a certain **A**lgorithm to be used.	DSA_Checksum_OSI
	Capabilities of Networks	
NSQ_	**N**etwork **S**pecification with **Q**uantitative values.	NSQ_Bit_Data_Corrupt
NSB_	**N**etwork **S**pecification with **B**oolean values.	NSB_Broadcast_Support
	Resources of Networks	
NRQ_	**N**etwork **R**esources with **Q**uantitative values.	NRQ_Delay_Min
NRB_	**N**etwork **R**esources with **B**oolean values.	(not used)
	Capabilities of End-systems (local)	
LSQ_	**L**ocal **S**pecification with **Q**uantitative values for end-systems.	LSQ_Number_of_CPU
LSB_	**L**ocal **S**pecification with **B**oolean values for end-systems.	LSB_Sequence_Guarantee
	Capabilities of End-systems (remote)	
RSQ_	**R**emote **S**pecification with **Q**uantitative values for end-systems.	RSQ_Number_of_CPU
RSB_	**R**emote **S**pecification with **B**oolean values for end-systems.	RSB_Sequence_Guarantee
	Resources of End-systems (local)	
LRQ_	**L**ocal **R**esource specification with **Q**uantitative values for end-systems.	LRQ_Avail_Memory
LRB_	**L**ocal **R**esource specification with **B**oolean values for end-systems.	(not used)
	Resources of End-systems (remote)	
RRQ_	**R**emote **R**esource specification with **Q**uantitative values for end-systems.	RRQ_Avail_Memory
RRB_	**R**emote **R**esource specification with **B**oolean values for end-systems.	(not used)
	Capabilities of Protocol Configurations	
PCQ_	**P**rotocol **C**onfiguration capabilities with **Q**uantitative values for end-systems.	PCQ_Max_Header
PCB_	**P**rotocol **C**onfiguration capabilities with **B**oolean values for end-systems.	(not used)

Table 4: Prefixes of Symbolic Variables

Name	Description	Unit
	Guarantee	
DSB_Best_Efford	QoS values provided best as possible	bool
DSB_Soft_Guarantees	QoS values provided under soft conditions	bool
DSB_Hard_Guarantees	QoS values provided under worst conditions	bool
	Delay	
DSQ_Delay_Max	Max. Delay between T-Data.req and T-Data.ind	ms
DSQ_Delay_Min	Min. Delay between T-Data.req and T-Data.ind	ms
	Interval	
DSQ_Interval	Time between two successive T-Data.req or T-Data.ind, which are equal.	ms
	Workahead	
DSQ_Workahead	(cf. Paragraph 3.3.1)	byte
	Size	
DSQ_Max_TSDU_Length	Maximum number of octetts of a TSDU	byte
	Damage	
DSQ_Bit_Error_Rate	Residual Bit Error Rate	neg. dec. log.
DSB_Damage_Ignore	No detection and correction of damage	bool
DSB_Damage_Notify	Detection and notification of damage	bool
DSB_Damage_Delete	Detection and deletion of damaged PDUs	bool
DSB_Damage_Correct	Detection and correction of damaged PDUs	bool
DSA_Best_Fit	Best algorithm for fulfilling BER	bool
DSA_Checksum_OSI	Use OSI checksum	bool
DSA_Checksum_CCITT	Use CCITT CRC	bool
DSA_Checksum_ANSI	Use ANSI CRC	bool
DSA_Checksum_32	Use 32-bit CRC	bool
DSA_FEC	Use Forward Error Correction	bool
	Loss	
DSB_Loss_Ignore	No loss detection and correction	bool
DSB_Loss_Notify	Loss detection and notification	bool
DSQ_Max_Retransmissions	Number of maximum retransmission retries for detected losses	int

Table 5: QoS Symbolic Variables (Part 1)

Name	Description	Unit
	Confidentiality	
DSA_DES_Con	Use Data Encryption Standard (DES)	bool
DSA_Triple_DES_Con	Use Triple-DES-algorithm	bool
DSA_IDEA_Con	Use IDEA-algorithm	bool
DSA_CBC_Mode	Block cipher operates in CBC-mode	bool
DSA_CFB_Mode	Block cipher operates in CFB-mode	bool
DSA_OFB_Mode	Block cipher operates in OFB-mode	bool
DSA_ECB_Mode	Block cipher operates in ECB-mode	bool
	Integrity	
DSA_DES_Int	Use Data Encryption Standard (DES)	bool
DSA_Triple_DES_Int	Use Triple-DES-algorithm	bool
DSA_IDEA_Int	Use IDEA-algorithm	bool
	Encapsulation	
DSA_DES_Enc	Use Data Encryption Standard (DES)	bool
DSA_Triple_DES_Enc	Use Triple-DES-algorithm	bool
DSA_IDEA_Enc	Use IDEA-algorithm	bool
DSA_Constant_Enc	Use a fixed constant as integrity check	bool
DSA_OSI_Enc	Use OSI CRC for integrity check	bool
DSA_CCITT_Enc	Use CCITT CRC for integrity check	bool
DSA_ANSI_Enc	Use ANSI CRC for integrity check	bool
DSA_8032_Enc	Use 32-bit CRC for integrity check	bool
DSA_ICV_Enc	Use cryptographic integrity check	bool
	Authentification	
DSB_Authentification	Authentification service selected	bool

Table 6: QoS Symbolic Variables (Part 2)

Name	Description	Unit
	Added Parameters in General	
DSQ_Packet_Loss_Rate	Acceptable packet loss rate	neg. dec. log.
DSQ_Packet_Error_Rate	Acceptable packet error rate	neg. dec. log.
DSB_Segmented_Delivery	Deliver segmented data to the user of the sink, without reassembling (streaming mode).	bool
DSB_Interactive_Behavior	Any type of message form the user of a sink can be returned to the source.	bool
DSQ_Priority	Levels of priority	set of int
DSB_Expedited_Data	Serves as a special level of priority	bool
DSB_Inter_Stream_Sync	Use of inter-stream synchronization	bool
DSB_Intra_Stream_Sync	Use of intra-stream synchronization	bool
	To Be Added Parameters for Security	
DSB_Confidentiality	Use modules for providing confidentiality	bool
DSB_Integrity	Use modules for providing integrity	bool
DSB_Encapsulation	Use modules for providing encapsulation	bool
DSQ_Chaining_Mode	Use the specified chaining mode	set of int
DSQ_ICV_Type	Use the specified icv type	set of int
DSQ_Algorithm_Number	Use the specified algorithm	set of int
DSA_XOR	Use XOR-algorithm	bool
DSA_Blowfish	Use Blowfish algorithm	bool
DSA_PCBC	Block cipher operates in PCBC-mode	bool
DSA_MD5_128	Use Message Digest Algorithm No. 5	bool
DSA_MD5_64	Use Message Digest Algorithm No. 5	bool
DSA_ISO10118_128	Use ISO Hashing algorithm	bool
DSA_ISO10118_64	Use ISO Hashing algorithm	bool
DSA_SHA	Use S Hashing Algorithm	bool
DSA_MD4	Use Message Digest Algorithm No. 4	bool

Table 7: QoS Symbolic Variables (Part 3)

Name	Description	Unit
	Capabilities of Networks	
NSQ_Bit_Data_Corrupt	Bit Error Rate of the network	neg. dec. log.
NSQ_Packet_Data_Loss	Packet Loss Rate of the network	neg. dec. log.
NSQ_Packet_Data_Corrupt	Packet Error Rate of the network	neg. dec. log.
NSQ_Max_Data_Unit_Size	Maximum size of packet in the network	byte
NSB_Broadcast_Support	Network supports broadcasting directly	bool
NSB_Multicast_Support	Network supports multicasting directly	bool
NSB_Expedited_Data	Network supports expedited data directly	bool
NSB_Sequence_Guarantee	Network supports sequences data	bool
NSB_Isochronous_Support	Network supports isochronous traffic directly	bool
	Resources of Networks	
NRQ_Throughput_Min	Minimum throughput of the network	byte/s
NRQ_Throughput_Ave	Average throughput of the network	byte/s
NRQ_Throughput_Max	Maximum throughput of the network	byte/s
NRQ_Delay_Min	Minimum delay of the network	ms
NRQ_Delay_Ave	Average delay of the network	ms
NRQ_Delay_Max	Maximum delay of the network	ms
NRQ_Jitter_Min	Minimum jitter of the network	ms
NRQ_Jitter_Ave	Average jitter of the network	ms
NRQ_Jitter_Max	Maximum jitter of the network	ms
	Capabilities of end-systems (local)	
LSQ_Number_of_CPU	Number of CPUs	int
LSB_Memory_Complete	Complete memory available	bool
LSB_Sequence_Guarantee	Sequenced processing of data units	bool
	Resources of end-systems (local)	bool
LRQ_CPU_Performance	Available CPU performance	int
LRQ_Avail_Memory	Available memory	byte
LRQ_Net_Interface_Bandwidth	Available network interface bandwidth	byte/s
LRQ_Host_Interface_Bandwidth	Available host interface bandwidth	byte/s
	Capabilities of end-systems (remote)	
RSQ_Number_of_CPU	Number of CPUs	int
RSB_Memory_Complete	Complete memory available	bool
RSB_Sequence_Guarantee	Sequenced processing of data units	bool
	Resources of end-systems (remote)	
RRQ_CPU_Performance	Available CPU performance	int
RRQ_Avail_Memory	Available memory	byte
RRQ_Net_Interface_Bandwidth	Available network interface bandwidth	byte/s
RRQ_Host_Interface_Bandwidth	Available host interface bandwidth	byte/s
	Capabilities of Protocol Configurations	
PCQ_Max_Header	Size of maximum PDU header	byte
PCQ_Memory_Needs_Send	Memory for the sender's part of the protocol	byte
PCQ_Memory_Needs_Recv	Memory for the receiver's part of the protocol	byte
PCQ_Delay_Send	Processing time for the sender's part of the protocol	ms
PCQ_Delay_Recv	Processing time for the receiver's part of the protocol	ms

Table 8: NSR Symbolic Variables

Criteria	**Active Peer**	**Passive Peer**
Addressing	explicitly (List of separate addresses)	implicitly (Group address)
Quantity	fixed and known	dynamical and unknown
Priority	similar and high	similar and low

Table 9: Classification of Peers

μsec	`ds001`			`ds002`			`ds007`		
Step	both	snd	rec	both	snd	rec	both	snd	rec
Step 1	536			436			443		
Step 2	8764			8900			8769		
Step 3	349			349			349		
Step 4	5175			4801			5268		
Step 5		3365	112		3507	290		3479	106
Step 6	143	102	49	147	203	50	147	205	48
Step 7		267	94		273	95		273	94
Step 8		833	104		882	102		889	107
Sums	14967	4567	359	14633	4865	537	14996	4846	355
SUM	19893			20035			20197		

Table 10: Preliminary Performance Evaluations (Part 1)

μsec	`ds010`			`ds015`		
Step	both	snd	rec	both	snd	rec
Step 1	437			442		
Step 2	8808			8736		
Step 3	347			530		
Step 4	9599			10547		
Step 5		4838	106		4624	109
Step 6	149	238	69	151	242	69
Step 7		324	99		328	99
Step 8		1088	96		1087	103
Sums	19340	6488	377	20406	6281	378
SUM	26198			27065		

Table 11: Preliminary Performance Evaluations (Part 2)

References

[1] ISO Standard, IS 8072, *Information processing systems — Open Systems Interconnection — Transport Service Definition*, 1986.

[2] ISO Standard IS 8073, *Information processing systems — Open Systems Interconnection — Transport Protocol Definition*, 1988.

[3] L. Svobodova, "Measured Performance of Transport Service in LANs," in *Computer Networks and ISDN Systems*, vol. 18, pp. 31–45, 1989.

[4] N. C. Hutchinson and L. L. Peterson, "The *x*-kernel: An Architecture for Implementing Network Protocols," *IEEE Transactions on Software Engineering*, vol. 17, pp. 64–76, January 1991.

[5] Defense Advanced Research Project Agency, *Transmission Control Protocol*, September 1981.

[6] D. D. Clark and D. L. Tennenhouse, "Architectural Considerations for a New Generation of Protocols," *Computer Communication Review (SIGCOMM)*, pp. 200–208, 24.-27. September 1990.

[7] B. Stiller, "PROCOM: A Protocol Configuration Manager in the Function-based Communication Subsystem," in *First International Workshop on High Performance Protocol Architectures*, (Sophia-Antipolis, France), pp. Session 3, Paper 9, 15.-16. December 1994.

[8] A. Danthine, "ESPRIT Project OSI 95 — New Transport Service for High-Speed Networking," in *Computer Networks and ISDN Systems*, vol. 25, pp. 384–399, 1992.

[9] W. T. Strayer, B. J. Dempsey, and A. C. Weaver, *XTP: The Xpress Transfer Protocol*. Reading, Massachusetts, U.S.A.: Addison Wesley, 1992.

[10] M. Zitterbart, "High-Speed Transport Components," *IEEE Network Magazine*, vol. 5, pp. 54–63, January 1991.

[11] R. W. Watson and S. A. Mamrak, "Gaining Efficiency in Transport Services by Appropriate Design and Implementation Choices," *ACM Transactions on Computer Systems*, vol. 5, pp. 97–120, May 1987.

[12] A. A. R. Cockburn, "Efficient Implementation of the OSI-Transport Protocol Checksum Algorithm Using 8/16 bit Arithmetic," tech. rep., IBM Research Division, Zürich Research Laboratories, Rüschlikon, Switzerland, 13. April 1987.

[13] T. Braun, *Ein paralleles Transportsubsystem für zellenbasierte Hochgeschwindigkeitsnetze*, vol. 10, no. 254. Düsseldorf, Germany: VDI, 30. June 1993.

[14] T. Braun, "PATROCLOS: A Flexible and High-Performance Transport Subsystem," in *4th International IFIP Workshop on Protocols for High Speed Networks*, (Vancouver, Canada), 10.-12. August 1994.

[15] T. Braun, B. Stiller, and M. Zitterbart, "XTP and VMTP on Multiprocessor Architectures," in *2nd Joint Workshop on High Speed Networks*, (Stuttgart, Germany), pp. 41–43, 19.-20. September 1992.

[16] M. A. Sidenius, "Hardware Support for Implementation of Transport Protocols," in *Protocols for High Speed Networks, II*, (Amsterdam, Netherlands), pp. 251–267, North Holland, 1991.

[17] D. R. Cheriton, "VMTP: A Transport Protocol for the Next Generation of Communication Systems," *Computer Communication Review (SIGCOMM)*, pp. 406–415, 5.-7. August 1986.

[18] S. Böcking, "The Configurable Transport Service of TEMPO," in *11th Annual Conference on European Fiber Optic Communications and Networks, EFOC&N 93*, (Den Haag, Netherlands), pp. 151–156, 30. June – 2. July 1993.

[19] S. Böcking and V. Seidel, "TIP's Protocol Run-Time System," in *12th Annual Conference on European Fiber Optic Communications and Networks, EFOC&N 94*, (Heidelberg, Germany), pp. 145–149, 21. – 23. June 1994.

[20] M. Zitterbart, B. Stiller, and A. Tantawy, "A Model for High-Performance Communication Subsystems," *IEEE Journal on Selected Areas in Communication*, vol. 11, pp. 507–518, May 1993.

[21] B. Stiller, *Flexible Protokollkonfiguration zur Unterstützung eines diensteintegrierenden Kommunikationssubsystems*, vol. 10, no. 306. Düsseldorf, Germany: VDI, 16. February 1994.

[22] D. F. Box, D. C. Schmidt, and T. Suda, "ADAPTIVE: An Object-Oriented Framework for Flexible and Adaptive Communication Protocols," in *High Performance Networking, IV*, (Amsterdam, Netherlands), pp. 367–382, IFIP Transactions C-14, North Holland, 1993.

[23] T. Plagemann, B. Plattner, M. Vogt, and T. Walter, "A Model for Dynamic Configuration of Light-Weight Protocols," in *IEEE 3rd Workshop on Future Trends of Distributed Systems*, (Taipeh, Taiwan), pp. 100–106, 14. – 16. April 1992.

[24] S. Böcking and P. Vindeby, "TIP: A Transport and Internetworking Package for ATM," (SIEMENS, ZFE, München), 1993.

[25] R. Händel and M. Huber, *Integrated Broadband Networks: An Introduction to ATM-based Networks.* Reading, Massachusetts, U.S.A.: Addison-Wesley, 1991.

[26] F. Ross, "FDDI — A Tutorial," *IEEE Communications Magazine*, vol. 24, pp. 10–17, May 1986.

[27] DARPA, *Transmission Control Protocol — DARPA Internet Protocol Specification, RFC 791*, September 1981.

[28] DARPA, *Internet Protocol — DARPA Internet Protocol Specification, RFC 793*, September 1981.

[29] S. Böcking, *TEMPO – Transport Protocol Specification.* München, Germany, 1. October 1993.

[30] M. Walch, *A Formal Specification of the TEMPO Protocol.* Berlin, Germany, 1. March 1994.

[31] O. Steinmeier, "Automatische Konfiguration von Protokolltürmen mittels PROCOM," Master's thesis, Diplomarbeit at Universität Karlsruhe, Institut für Telematik, Germany, March 1993.

[32] ISO Standard, IS 7498, *Information processing systems — Open Systems Interconnection — Basic Reference Model*, 1985.

[33] B. Stiller, *CogPiT Report No. 1: Rating the Concept of Configuration and Definition of a Module Description Lamguage (Final Revision).* Universität Karlsruhe, Institut für Telematik, Germany, D-76128 Karlsruhe, Germany, Updated: 2. August 1994.

[34] B. Stiller, *CogPiT Report No. 2: Evaluation and Development of Configuration Support within TIP: QoS-Parameter.* Universität Karlsruhe, Institut für Telematik, Germany, D-76128 Karlsruhe, Germany, 1. June 1994.

[35] B. Stiller, *CogPiT Report No. 3: Evaluation and Development of Configuration Support within TIP: Network and System Resources.* Universität Karlsruhe, Institut für Telematik, Germany, D-76128 Karlsruhe, Germany, 1. June 1994.

[36] ITU-T Draft Recommendation Q.2931, *Edinburgh TD 155, Broadband Integrated Services Digital Network (B-ISDN), Digital Subscriber Signalling System No.2, User Network Interface Layer 3 Specification for Basic Call/Connection Control.* Geneva, Switzerland, 13. - 21. June 1994.

[37] ATM-Forum, *ATM User Network Interface Specification, Version 3.1*, 21. July 1994.

[38] A. Danthine, *The OSI'95 Transport Service with Multimedia Support — Research Reports ESPRIT, Project 5341, Volume 1.* Berlin, Germany: Springer, 1994.

[39] International Organization for Standardization, "Quality-of-Service — Basic Framework — CD Text," Tech. Rep. ISO/IEC JTC1/SC21 N9309, ISO, 9. – 13. January 1995.

[40] International Organization for Standardization, "Quality-of-Service — Methods and Mechanisms — Working Draft No. 2," Tech. Rep. ISO/IEC JTC1/SC21 N9310, ISO, 9. – 13. January 1995.

[41] B. Stiller, "Aushandlungsverfahren für QoS-Parameter und (Re-) Konfigurationsbeschreibungen," in *Anwendungsunterstützung für heterogene Rechnernetze*, no. ISBN 3-86012-013-1, pp. 79–88, 30. – 31. March 1995.

[42] H. Eriksson, "MBone: The Multicast Backbone," *Communications of the ACM*, vol. 37, pp. 54–60, Aug. 1994.

A Abbreviations

ADAPTIVE	A Dynamically Assembled Protocol Transformation, Integration, and Validation Environment
AEP	Association End Point
AG	Aktiengesellschaft
ANSI	American National Standards Institute
API	Application Programming Interface
APP	Application
ATM	Asynchronous Transfer Mode
BER	Bit Error Rate
BLK	Blocking
BM	Bidirectional Multicast
bool	boolean
C	Configuration
CBC	Cypher Block Chaining
CFB	Cypher Feedback
COFAL	Configuration Algorithm
COMP	Composition
CONF	Confidentiality
CPM	Critical Path Method
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CogPiT	Configuration of Protocols in TIP
DEBLK	Deblocking
dec.	decimal
DEFFUNC	Define Function
DEFMECH	Define Mechanism
DES	Data Encryption Standard
DISC	Discomposition
DSA	Data Stream Algorithm
DS	Data Stream
DSQ	Data Stream Qualitative Values
Da CaPo	Dynamic Configuration of Protocols
E	Environmental
EBNF	Extended Backus-Naur Form
ECB	Electronic Code Block
ENCAP	Encapsulation
F	Future
F-CSS	Function-based Communication Subsystem
F-PCL	Function-based Protocol Configuration Language
FDDI	Fiber Distributed Data Interface
FTP	File Transfer Protocol
GOSIP	US Government Open System Interconnection Protocols

ICV	Initial Chaining Value
IDEA	International Data Encryption Algorithm
ind.	Indication
INI	Initial
int	integer
INTEG	Integrity
IP	Internet Protocol
ISO	International Organization for Standardization
ITP	Interface to Provider
ITU	Interface to User
log.	logarithmical
LRB	Local Resource specification with Boolean values
LRQ	Local Resource specification with Quantitative values
LSB	Local Specification with Boolean values
LSQ	Local Specification with Quantitative values
MAP	Manufacturing Automation Protocol
max	maximum
MDL	Module Description Language
MGT	Management
ms	milliseconds
neg.	negative
NET	Network
NRB	Network Resources with Boolean values
NRQ	Network Resources with Quantitative values
NSB	Network Specification with Boolean values
NSQ	Network Specification with Quantitative values
NSR	Network and System Resources
OFB	Output Feedback
PCB	Protocol Configuration capabilities with Boolean values
PCBC	Propagating Cipher Block Chaining
PCQ	Protocol Configuration capabilities with Quantitative values
PDU	Protocol Data Unit
PF	Protocol Function
PFP	Protocol Function Pools
PFG	Protocol Function Graph
PRED	Predecessor
PROCOM	Protocol Configuration Manager
PROREM	Protocol Resource Manager
PXY	Proxy
QoS	Quality-of-Service
R	Receiver
rcv	Receiver
req.	Request
RG	Research Grant
RRB	Remote Resource specification with Boolean values

RRQ	Remote Resource specification with Quantitative values
RSB	Remote Specification with Boolean values
RSQ	Remote Specification with Quantitative values
s	seconds
S	Sender
SAP	Service Access Point
snd	Sender
SNK	Sink
SRC	Source
SUCC	Successor
T	TEMPO or Transport
TCP	Transmission Control Protocol
TIP	Transport and Internetworking Package
TIS	Timed Interaction System
TOP	Technical and Office Protocol
TP4	Transport Protocol Class 4
TSDU	Transport Service Data Unit
UNI	User-Network Interface
UM	Unidirectional Multicast
US	United States
USR	User
VMTP	Versatile Message Transaction Protocol
X	Extended
XTP	Xpress Transfer Protocol
ZFE	Zentralstelle für Forschung und Entwicklung

B Synonymous Terminology between CogPiT and F-CSS

CogPiT	**F-CSS**
Channel Configuration	Protocol Configuration
COFAL	PROCOM
MDL Specification	Protocol Resource Pool
Module	Implemented Protocol Mechanism
Protocol Configuration	Protocol Machine
Protocol Function	Protocol Function
Protocol Mechanism	Protocol Mechanism
Worldgraph	Protocol Function Graph

C EBNF for MDL

```
<StartSpec>     ::=    <GraphSpec> <PoolSpec>

<GraphSpec>     ::=    <GraphSpecDef> [<ConstDef>] <GraphDef>
<GraphSpecDef>  ::=    GRAPHSPECIFICATION <Name>';'
<GraphDef>      ::=    {<FunctDef>}

<FunctDef>      ::=    DEFFUNC <Name>
                       '{'
                           {<BoolKey> <BoolList>';'}
                           {<GraphMechDef>}
                       '}'
<BoolKey>       ::=    FORBIDDEN | REQUIRED
<BoolList>      ::=    <BoolExpr>[','<BoolList>]

<GraphMechDef>  ::=    DEFMECH <Name>
                       '{'
                           {<CondKey> <CondList>';'}
                       '}'
<CondKey>       ::=    SUCC | PRED
<CondList>      ::=    <CondExpr>[','<CondList>]
<CondExpr>      ::=    <Name>[' IF '<BoolExpr>]

<PoolSpec>      ::=    <PoolSpecDef> [<ConstDef>] {<MechDef>}
<PoolSpecDef>   ::=    SPECIFICATION <Name>';'

<ConstDef>      ::=    {#DEFINE <Name> <Expr> |

<MechDef>       ::=    DEFMECH <Name>
                       '{'
                           <Declaration>
                       '}'
<Declaration>   ::=    [RECEIVER <BoolExpr>';']
                       [SENDER <BoolExpr>';']
                       USERS <DecNumber>';'
                       PROVIDERS <DecNumber>';'
                       TYPE <Name>';'
                       CHILDS <DecNumber>';'
                       MEMORY <Expr>';'
                       TIME <Expr>';'
                       {Attribute}

<Attribute>     ::=    <BoolKey> <BoolList>';' |
                       <ParaKey> <ParaList>';' |
                       <NameKey> <NameList>';'
```

```
<ParaKey>        ::=    PARAMETER
<ParaList>       ::=    <Name>'='<Expr>' DEFAULT '<Number>[','<ParaList>]
<NameKey>        ::=    INPUT | OUTPUT
<NameList>       ::=    <Name>[','<NameList>]

<ExprList>       ::=    <Expr>[','<ExprList>]
<Expr>           ::=    <Name> | <Number>| '-'<Expr> | <Expr>'+'<Expr> |
                        <Expr>'-'<Expr> | <Expr>'*'<Expr> |
                        <Expr>'/'<Expr> | '('<Expr>')'
<BoolExpr>       ::=    <BoolConstant> | <Name> | <Comparison> |
                        '('<BoolExpr>')' | <BoolExpr>'&'<BoolExpr> |
                        <BoolExpr>'|'<BoolExpr> | '!'<BoolExpr>
<BoolConstant>   ::=    TRUE | FALSE
<Comparison>     ::=    <Expr> '>' <Expr> | <Expr> '<' <Expr> |
                        <Expr> '>=' <Expr> | <Expr> '<=' <Expr> |
                        <Expr> '=' <Expr> | <Expr> '!=' <Expr> |
                        <BoolExpr> '=' <BoolExpr>

<Name>           ::=    <Letter>{<Letter>|<Digit>|'_'|'/'|'\'|':'
<Letter>         ::=    'a'|'b'|'c'|'d'|'e'|'f'|'g'|'h'|'i'|'j'|'k'|'l'|'m'|
                        'n'|'o'|'p'|'q'|'r'|'s'|'t'|'u'|'v'|'w'|'x'|'y'|'z'|
                        'A'|'B'|'C'|'D'|'E'|'F'|'G'|'H'|'I'|'J'|'K'|'L'|'M'|
                        'N'|'O'|'P'|'Q'|'R'|'S'|'T'|'U'|'V'|'W'|'X'|'Y'|'Z'
<Digit>          ::=    '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'
<Number>         ::=    [<Sign>]<DecNumber>
<DecNumber>      ::=    <Digit>{<Digit>}
<Sign>           ::=    '+' | '-'
```

D How to Add New QoS or NSR Parameter

The addition of new symbolic variables into the existing environment is quite easy, but needs some enhancements to be done. First of all, the syntactical form, *e.g.,* the name has to be defined, checking for non-overlapping names with existing symbolic variables. This can be done using the file `s_ds_spec.dat` (cf. Figure 18). Additionally, the newly created name has to be added in the same file as an ASCII string. Afterwards, the new symbolic variable has to be included into the relevant array of `qualt_var`, `quant_var`, or `_var`, which happens in the source code file `deftoken.cxx`. The corresponding `deftoken.hxx` file needs the definition of a `#define name value` statement, where the name equals the syntax of the ASCII string and the value an non-existing value of the already defined symbolic variables within `deftoken.hxx`.[11] Afterwards a new compilation has to be done, except for the case of using already pre-defined patterns for symbolic variables (cf. Subsection 7.4).

E How to Prepare a Correct MDL Specification for CogPiT

The preparation of a MDL specification requires careful analysis of the protocol and its functionality. Therefore, an informal method of preparing a correct MDL specification is explained. The initial building blocks have to be identified, which includes the functions and its mechanisms of the considered communication protocol or entire environment. Every part that may be configured in the future has to be taken into account at this step.

First of all the worldgraph will be constructed in the worldgraph file. The order of processing between each mechanism has to be defined using the predecessor and successor relation applying the appropriate MDL keywords. All relevant prerequisites have to be taken into account, such as a loop-free graph, correct interface mechanisms, or sender and receiver worldgraphs. Additionally, the optional and mandatory modules have to be marked clearly to specify the correct worldgraph and all relevant interconnections for all possible combinations having a certain module within the graph or leaving it outside. It has to be remembered that the worldgraph includes all possible interconnections. Because of fairly often existing limitations of optional and mandatory module combinations, the number of possible combinations is usually quite small.

On the classification of the identified functionality into functions and mechanisms no general guidelines are available, but the intention is to collect quite similar mechanisms in terms of their processed functionality and by means of required conditions to define a function. Some intuitive tries and examples have to be experienced personally first. An example is the function "security" used within the TIP environment (cf. Paragraph 3.3.3). The definition of similar **REQUIRED** conditions are based on the fact that security is required. The special sub-decisions can be done afterwards on the mechanism level by distinguishing, *e.g.,* the necessity for confidentiality or integrity.

Once these classifications have been done, some rearrangements of the worldgraph might be necessary to simplify the structure. The next step has to deal with the **if**-conditions to be connected to the successor and predecessor attributes on a mechanism basis within the worldgraph specification. On a per function/mechanism basis the **REQUIRED** conditions are defined and added to the successor and predecessor attributes. They define the optional

[11] An update of the source code files `printtab.cxx`, `user_simu.cxx`, and `sys_sim.cxx` may be necessary for adapting the testing and evaluation environment for useful printouts.

necessity of an arrow within the worldgraph that can be selected within the configuration process afterwards. Normally, that picture includes now all relevant worldgraph information. A close check has to be done for all "arrows", the correct order of processing between mechanisms, and the defined conditions. There should not occur any generally contrary conditions for a function, since in this case the corresponding function will never be selected for a future channel configuration. This part forms in general the crucial task of the specification. Great care has to be taken for a correctly specified worldgraph, since these information form a main basis for the configuration algorithm.

Finally, the specification follows for each separate mechanism for the pool of modules written into the pool file. Starting with the hard part first, the alternatives between multiple mechanisms within one single function have to be specified using the **REQUIRED** conditions on the mechanism level. The most specialized mechanism has to appear first within the order of alternative mechanisms within the worldgraph. Therefore, some rearrangements might be necessary in the worldgraph file. All additional keywords have to used with appropriate values as described elsewhere in this report to achieve at least the minimal specification information per mechanism.

All required conditions have to be expressed within logical expressions that are based on defined symbolic variables, only. There is no way of adding additional information, except for constants previously defined in the files. Care has to be taken for the use of boolean and arithmetical symbolic variables that they are not mixed together, such as adding a boolean value to an integer. That will be not recognized within the prototypical implementation as an error, but it is an unidentified specification error. Furthermore, a careful definition of arithmetical expressions must consider the units for symbolic variables. An unidentified specification error would be the subtraction of an integer variable defining throughput as megabyte per second from another throughput variable defined in byte per second.

In an informal description the experience of some examples has been explained. Unfortunately, there is no algorithm applicable for the process of generating a MDL specification, but the reader or configuration developer will experience smaller examples on their own.

It may be useful to remember that the pool file can be used for different worldgraph files, if every used mechanism of the worldgraph file appears within the pool file. Otherwise an error will occur during configuration time.

F Examples

F.1 Extended TEMPO Specification in MDL

This Appendix includes some parts of a MDL specification for the TEMPO protocol as it has been used for configuration demonstrations only. This is in complete detail the common worldgraph for sender and receiver side, while applying the defined Quality-of-Service as well as Network and System Resource parameters.

```
GRAPHSPECIFICATION TEMPO_Worldgraph;

/* Functions and mechanisms for the TIP environment */

DEFFUNC Management_Sender
        {
        REQUIRED  TRUE;
        DEFMECH   MGT_Simple_Sender
                {
                SUCC SAP_Simple_Sender;
                SUCC SAP_Secure_Sender;
                PRED APP;
                }
        }

DEFFUNC Management_Receiver
        {
        REQUIRED  TRUE;
        DEFMECH   MGT_Simple_Receiver
                {
                SUCC SAP_Simple_Receiver;
                SUCC SAP_Secure_Receiver;
                PRED NET;
                }
        }

DEFFUNC Service_Access_Point_Sender
        {
        REQUIRED  TRUE;
        DEFMECH   SAP_Secure_Sender
                {
                SUCC AEP_Simple_Sender;
                SUCC AEP_Secure_Sender;
                PRED MGT_Simple_Sender;
                }
        DEFMECH   SAP_Simple_Sender
                {
                SUCC AEP_Simple_Sender;
                SUCC AEP_Secure_Sender;
                PRED MGT_Simple_Sender;
                }
        }

DEFFUNC Service_Access_Point_Receiver
        {
        REQUIRED  TRUE;
        DEFMECH   SAP_Secure_Receiver
                {
                SUCC AEP_Simple_Receiver;
                SUCC AEP_Secure_Receiver;
                PRED MGT_Simple_Receiver;
                }
        DEFMECH   SAP_Simple_Receiver
                {
```

```
		SUCC AEP_Simple_Receiver;
		SUCC AEP_Secure_Receiver;
		PRED MGT_Simple_Receiver;
		}
	}

DEFFUNC Association_End_Point_Sender
	{
	REQUIRED  TRUE;
	DEFMECH   AEP_Secure_Sender
		{
		SUCC ITU_In_Sender;
		PRED SAP_Secure_Sender;
		PRED SAP_Simple_Sender;
		}
	DEFMECH   AEP_Simple_Sender
		{
		SUCC ITU_In_Sender;
		PRED SAP_Secure_Sender;
		PRED SAP_Simple_Sender;
		}
	}

DEFFUNC Association_End_Point_Receiver
	{
	REQUIRED  TRUE;
	DEFMECH   AEP_Secure_Receiver
		{
		SUCC ITP_In_Receiver;
		PRED SAP_Simple_Receiver;
		PRED SAP_Secure_Receiver;
		}
	DEFMECH   AEP_Simple_Receiver
		{
		SUCC ITP_In_Receiver;
		PRED SAP_Simple_Receiver;
		PRED SAP_Secure_Receiver;
		}
	}

/* Functions and mechanisms for the extended TEMPO protocol */

DEFFUNC InterfaceToUser_In_Sender
	{
	REQUIRED  TRUE;
	DEFMECH   ITU_In_Sender
		{
		SUCC PDU_Segmentation;
		SUCC PDU_Composition_TEMPO  IF !((DSQ_Max_TSDU_Length +
			PCQ_Max_Header) > NSQ_Max_Data_Unit_Size);
		PRED AEP_Simple_Sender;
		PRED AEP_Secure_Sender;
		}
	}

DEFFUNC InterfaceToUser_In_Receiver
	{
	REQUIRED  DSB_Interactive_Behavior;
	REQUIRED  DSB_Loss_Notify;
	DEFMECH   ITU_In_Receiver
		{
		SUCC Reply_PDU_Composition_TEMPO;
		PRED APP;
		}
	}
```

```
DEFFUNC InterfaceToUser_Out_Sender
        {
        REQUIRED  DSB_Interactive_Behavior;
        REQUIRED  DSB_Loss_Notify;
        DEFMECH   ITU_Out_Sender
                {
                SUCC APP;
                PRED Error_Control_TEMPO IF (DSQ_Bit_Error_Rate = 0 |
                        DSB_Damage_Notify | DSB_Damage_Correct |
                        DSB_Loss_Notify | DSB_Soft_Guarantees |
                        DSB_Hard_Guarantees |
                        (NSQ_Packet_Data_Loss <= DSQ_Packet_Loss_Rate));
                }
        }

DEFFUNC InterfaceToUser_Out_Receiver
        {
        REQUIRED  TRUE;
        DEFMECH   ITU_Out_Receiver
                {
                SUCC APP;
                PRED Sink_Status_TEMPO;
                }
        }

DEFFUNC InterfaceToProvider_Out_Sender
        {
        REQUIRED  TRUE;
        DEFMECH   ITP_Out_Sender
                {
                SUCC NET;
                PRED Checksum_Calculate_Data_OSI;
                PRED Checksum_Calculate_Data_CCITT;
                PRED Checksum_Calculate_Data_ANSI;
                PRED Checksum_Calculate_Data_32;
                PRED PDU_Composition_TEMPO  IF !(DSQ_Bit_Error_Rate = 0 |
                         DSB_Damage_Notify | DSB_Damage_Delete |
                        DSB_Damage_Correct | DSB_Loss_Notify) &
                        !(DSB_Soft_Guarantees | DSB_Hard_Guarantees);
                PRED Retransmission_selective IF
                        !(DSB_Soft_Guarantees | DSB_Hard_Guarantees);
                PRED Flow_Control_rate_based_Sender;
                PRED Flow_Control_window_based_Sender;
                PRED Retransmission_cumulative IF
                        !(DSB_Soft_Guarantees | DSB_Hard_Guarantees);
                }
        }

DEFFUNC InterfaceToProvider_Out_Receiver
        {
        // Necessary, if receiver needs to generate replies
        REQUIRED  DSB_Interactive_Behavior;
        // Necessary, if the backward direction to the sender is
        // required. e.g., for acks
        REQUIRED  DSQ_Bit_Error_Rate = 0;
        REQUIRED  DSB_Damage_Notify;
        REQUIRED  DSB_Damage_Correct;
        REQUIRED  DSB_Loss_Notify;
        // Comparison of exponents, therefore, "<=" is valid
        REQUIRED  NSQ_Packet_Data_Loss <= DSQ_Packet_Loss_Rate;
        // Necessary for control information as well
        REQUIRED  DSB_Soft_Guarantees;
        REQUIRED  DSB_Hard_Guarantees;
        DEFMECH   ITP_Out_Receiver
                {
                SUCC NET;
```

```
                PRED Reply_PDU_Composition;
                }
        }

DEFFUNC InterfaceToProvider_In_Sender
        {
        // Necessary, if the backward direction to the sender is
        // required. e.g., for acks or flow control information,
        // and the error cases have to be maintained
        REQUIRED  DSQ_Bit_Error_Rate = 0;
        REQUIRED  DSB_Damage_Notify;
        REQUIRED  DSB_Damage_Correct;
        REQUIRED  DSB_Loss_Notify;
        REQUIRED  DSB_Soft_Guarantees;
        REQUIRED  DSB_Hard_Guarantees;
        // Comparison of exponents, therefore, "<=" is valid
        REQUIRED  NSQ_Packet_Data_Loss <= DSQ_Packet_Loss_Rate;
        // Required if interactive behavior is requested
        REQUIRED  DSB_Interactive_Behavior;
        DEFMECH   ITP_In_Sender
                {
                SUCC Error_Control_TEMPO;
                SUCC ITU_Out_Sender  IF !(DSQ_Bit_Error_Rate = 0 |
                        DSB_Damage_Notify | DSB_Damage_Correct |
                        DSB_Loss_Notify | DSB_Soft_Guarantees |
                        DSB_Hard_Guarantees |
                        (NSQ_Packet_Data_Loss <= DSQ_Packet_Loss_Rate));
                 PRED NET;
                }
        }

DEFFUNC InterfaceToProvider_In_Receiver
        {
        REQUIRED  TRUE;
        DEFMECH   ITP_In_Receiver
                {
                SUCC Check_PDU_TEMPO;
                PRED AEP_Simple_Receiver;
                PRED AEP_Secure_Receiver;
                }
        }

DEFFUNC Segmentation
        {
        REQUIRED  (DSQ_Max_TSDU_Length + PCQ_Max_Header) >
                         NSQ_Max_Data_Unit_Size;
        DEFMECH   PDU_Segmentation
                {
                SUCC PDU_Composition_TEMPO;
                PRED ITU_In_Sender;
                }
        }

DEFFUNC Reassembly
        {
        // Necessary only if max PDU size too big and no segmented
        // delivery allowed
        REQUIRED  (((DSQ_Max_TSDU_Length + PCQ_Max_Header) >
                       NSQ_Max_Data_Unit_Size)
                        & DSB_Segmented_Delivery = FALSE);
        DEFMECH   PDU_Reassembly
                {
                SUCC Sink_Status_TEMPO;
                PRED Check_PDU_TEMPO;
                }
        }
```

```
DEFFUNC PDU_Composition
        {
        REQUIRED  TRUE;
        DEFMECH   PDU_Composition_TEMPO
                {
                SUCC Flow_Control_rate_based_Sender;
                SUCC Flow_Control_window_based_Sender;
                SUCC ITP_Out_Sender IF !(DSQ_Bit_Error_Rate = 0 |
                        DSB_Damage_Notify | DSB_Damage_Delete |
                        DSB_Damage_Correct | DSB_Loss_Notify) &
                        !(DSB_Soft_Guarantees | DSB_Hard_Guarantees);
                SUCC Checksum_Calculate_Data_OSI;
                SUCC Checksum_Calculate_Data_CCITT;
                SUCC Checksum_Calculate_Data_ANSI;
                SUCC Checksum_Calculate_Data_32;
                PRED PDU_Segmentation;
                PRED ITU_In_Sender IF !((DSQ_Max_TSDU_Length +
                        PCQ_Max_Header) > NSQ_Max_Data_Unit_Size);
                }
        DEFMECH   Reply_PDU_Composition_TEMPO
                {
                SUCC ITP_Out_Receiver;
                PRED Acknowledgement_cumulative;
                PRED Acknowledgement_selective;
                PRED Flow_Control_window_based_Receiver;
                PRED Flow_Control_rate_based_Receiver;
                PRED ITU_In_Receiver;
                }
        }


DEFFUNC Checksum_Calculate
        {
        // Each of the following REQ must appear in the MECHs as well
        // to allow for the selection of a specific one if requested
        REQUIRED  DSQ_Bit_Error_Rate = 0;
        REQUIRED  DSB_Damage_Notify;
        REQUIRED  DSB_Damage_Delete;
        REQUIRED  DSB_Damage_Correct;
        REQUIRED  DSB_Loss_Notify;
        FORBIDDEN DSB_Damage_Ignore;
        FORBIDDEN DSB_Loss_Ignore;
        DEFMECH   Checksum_Calculate_Data_OSI
                {
                SUCC ITP_Out_Sender;
                PRED PDU_Composition_TEMPO;
                }
        DEFMECH   Checksum_Calculate_Data_CCITT
                {
                SUCC ITP_Out_Sender;
                PRED PDU_Composition_TEMPO;
                }
        DEFMECH   Checksum_Calculate_Data_ANSI
                {
                SUCC ITP_Out_Sender;
                PRED PDU_Composition_TEMPO;
                }
        DEFMECH   Checksum_Calculate_Data_32
                {
                SUCC ITP_Out_Sender;
                PRED PDU_Composition_TEMPO;
                }
        }
```

```
DEFFUNC Checksum_Validate
        {
        // Each of the following REQ must appear in the MECHs separately
        // to allow for the selection of a specific one if requested
        REQUIRED  DSQ_Bit_Error_Rate = 0;
        REQUIRED  DSB_Damage_Notify;
        REQUIRED  DSB_Damage_Delete;
        REQUIRED  DSB_Damage_Correct;
        REQUIRED  DSB_Loss_Notify;
        FORBIDDEN DSB_Damage_Ignore;
        FORBIDDEN DSB_Loss_Ignore;
        DEFMECH   Checksum_Validate_Data_OSI
                {
                SUCC Sink_Status_TEMPO;
                PRED Check_PDU_TEMPO;
                }
        DEFMECH   Checksum_Validate_Data_CCITT
                {
                SUCC Sink_Status_TEMPO;
                PRED Check_PDU_TEMPO;
                }
        DEFMECH   Checksum_Validate_Data_ANSI
                {
                SUCC Sink_Status_TEMPO;
                PRED Check_PDU_TEMPO;
                }
        DEFMECH   Checksum_Validate_Data_32
                {
                SUCC Sink_Status_TEMPO;
                PRED Check_PDU_TEMPO;
                }
        }

DEFFUNC Flow_Control
        {
        REQUIRED  DSB_Soft_Guarantees;
        REQUIRED  DSB_Hard_Guarantees;
        DEFMECH   Flow_Control_rate_based_Sender
                {
                SUCC ITP_Out_Sender;
                PRED PDU_Composition_TEMPO;
                PRED Retransmission_selective;
                PRED Retransmission_cumulative;
                PRED Error_Control_TEMPO IF
                        (DSQ_Bit_Error_Rate = 0 | DSB_Damage_Notify |
                        DSB_Damage_Correct | DSB_Loss_Notify |
                        (NSQ_Packet_Data_Loss <= DSQ_Packet_Loss_Rate));
                }
        DEFMECH   Flow_Control_window_based_Sender
                {
                SUCC ITP_Out_Sender;
                PRED PDU_Composition_TEMPO;
                PRED Retransmission_selective;
                PRED Retransmission_cumulative;
                PRED Error_Control_TEMPO;
                }

        DEFMECH   Flow_Control_rate_based_Receiver
                {
                SUCC Reply_PDU_Composition_TEMPO;
                PRED Sink_Status_TEMPO;
                }
        DEFMECH   Flow_Control_window_based_Receiver
                {
                SUCC Reply_PDU_Composition_TEMPO;
                PRED Sink_Status_TEMPO;
```

```
                }
        }

DEFFUNC Error_Control
        {
        // Necessary, if the backward direction to the sender is
        // required. e.g., for acks or flow control information,
        // and the error cases have to be maintained
        REQUIRED  DSQ_Bit_Error_Rate = 0;
        REQUIRED  DSB_Damage_Notify;
        REQUIRED  DSB_Damage_Correct;
        REQUIRED  DSB_Loss_Notify;
        REQUIRED  DSB_Soft_Guarantees;
        REQUIRED  DSB_Hard_Guarantees;
        // Comparison of exponents, therefore, "<=" is valid
        REQUIRED  NSQ_Packet_Data_Loss <= DSQ_Packet_Loss_Rate;
        DEFMECH   Error_Control_TEMPO
                {
                SUCC Retransmission_selective;
                SUCC Retransmission_cumulative;
                SUCC Flow_Control_window_based_Sender;
                SUCC Flow_Control_rate_based_Sender ;
                PRED ITP_In_Sender;
                }
        }

DEFFUNC Check_PDU
        {
        REQUIRED  TRUE;
        DEFMECH   Check_PDU_TEMPO
                {
                SUCC Sink_Status_TEMPO;
                SUCC Checksum_Validate_Data_OSI;
                SUCC Checksum_Validate_Data_CCITT;
                SUCC Checksum_Validate_Data_ANSI;
                SUCC Checksum_Validate_Data_32;
                SUCC PDU_Reassembly;
                PRED ITP_In_Receiver;
                }
        }

DEFFUNC Sink_Status
        {
        REQUIRED  TRUE;
        DEFMECH   Sink_Status_TEMPO
                {
                SUCC Acknowledgement_cumulative;
                SUCC Acknowledgement_selective;
                SUCC Flow_Control_window_based_Receiver;
                SUCC Flow_Control_rate_based_Receiver;
                SUCC ITU_Out_Receiver;
                PRED Checksum_Validate_Data_OSI;
                PRED Checksum_Validate_Data_CCITT;
                PRED Checksum_Validate_Data_ANSI;
                PRED Checksum_Validate_Data_32;
                PRED Check_PDU_TEMPO;
                PRED PDU_Reassembly;
                }
        }

DEFFUNC Retransmission
        {
        REQUIRED  DSQ_Bit_Error_Rate = 0;
        REQUIRED  DSB_Damage_Notify;
        REQUIRED  DSB_Damage_Correct;
        REQUIRED  DSB_Loss_Notify;
```

```
        // Comparison of exponents, therefore, "<=" is valid
        REQUIRED  NSQ_Packet_Data_Loss <= DSQ_Packet_Loss_Rate;
        DEFMECH   Retransmission_selective
                {
                SUCC ITP_Out_Sender IF
                        !(DSB_Soft_Guarantees | DSB_Hard_Guarantees);
                SUCC Flow_Control_rate_based_Sender;
                SUCC Flow_Control_window_based_Sender;
                PRED Error_Control_TEMPO;
                }

        DEFMECH   Retransmission_cumulative
                {
                SUCC ITP_Out_Sender IF
                        !(DSB_Soft_Guarantees | DSB_Hard_Guarantees);
                SUCC Flow_Control_rate_based_Sender;
                SUCC Flow_Control_window_based_Sender;
                PRED Error_Control_TEMPO;
                }
        }

DEFFUNC Acknowledgement
        {
        REQUIRED  DSQ_Bit_Error_Rate = 0;
        REQUIRED  DSB_Damage_Notify;
        REQUIRED  DSB_Damage_Correct;
        REQUIRED  DSB_Loss_Notify;
        // Comparison of exponents, therefore, "<=" is valid
        REQUIRED  NSQ_Packet_Data_Loss <= DSQ_Packet_Loss_Rate;
        DEFMECH   Acknowledgement_cumulative
                {
                SUCC Reply_PDU_Composition_TEMPO;
                PRED Sink_Status_TEMPO;
                }
        DEFMECH   Acknowledgement_selective
                {
                SUCC Reply_PDU_Composition_TEMPO;
                PRED Sink_Status_TEMPO;
                }
        }
```

F.2 A Preliminary TIP Specification in MDL

This Appendix includes some parts of a MDL specification for the currently available and implemented TIP system, which has been applied for configuration and downloading purposes. This is in detail the complete pool of modules, , while applying the defined Quality-of-Service as well as Network and System Resource parameters.

```
SPECIFICATION TEMPO_Pool_of_Modules;

// Definition of expressions and constants

#DEFINE ms 1000                 // milliseconds
#DEFINE ys 1000                 // microseconds

#DEFINE no_unit 1
#DEFINE bytes   1
#DEFINE word    2

#DEFINE i_max_sdu_size  128000  // bytes
#DEFINE i_sdu_size         512  // bytes
#DEFINE i_pdu_size        4000  // bytes
#DEFINE i_window_size       10  // bytes
#DEFINE i_max_queued_no     10  // bytes
```

```
#DEFINE i_watermark       2048  // bytes
#DEFINE i_sink_no            1

// Mechanism definitions

DEFMECH API:M_146_254_PXY_003_USR_DATA_RAW
       {
       TYPE      SRC, SNK;
       SENDER    TRUE;
       RECEIVER  TRUE;
       REQUIRED  TRUE;
       OUTPUT    db;
       INPUT     db;
       USERS     999;                        // unlimited number of users
       PROVIDERS 1;
       CHILDS    1;
       TIME      10 * ys;
       MEMORY    100 * bytes;
       }

DEFMECH M_146_254_SAP_DUMMY
       {
       TYPE      SAP;
       SENDER    TRUE;
       RECEIVER  TRUE;
       REQUIRED  TRUE;
       OUTPUT    db;
       INPUT     db;
       USERS     1;
       PROVIDERS 1;
       CHILDS    1;
       TIME      10 * ys;
       MEMORY    50 * bytes;
       }

DEFMECH M_146_254_AEP_DUMMY
       {
       TYPE      AEP;
       SENDER    TRUE;
       RECEIVER  TRUE;
       REQUIRED  TRUE;
       OUTPUT    db;
       INPUT     db;
       USERS     1;
       PROVIDERS 1;
       CHILDS    1;
       TIME      10 * ys;
       MEMORY    50 * bytes;
       }

DEFMECH M_146_254_NET_UM
       {
       TYPE      SAP, AEP, SRC, SNK;
       SENDER    TRUE;
       RECEIVER  TRUE;
       REQUIRED  DSB_Interactive_Behavior;
       OUTPUT    db;
       INPUT     db;
       PARAMETER i_aal_type = 3;             // valid 0, 3, 4, 5
       PARAMETER i_dataflow = 2;             // 2 = multicast, 0 = unicast
       USERS     10;
       PROVIDERS 1;
       CHILDS    1;
       TIME      25 * ys;
       MEMORY    1000 * bytes;
       }
```

```
DEFMECH M_146_254_NET_BM
        {
        TYPE      SAP, AEP, SRC, SNK;
        SENDER    TRUE;
        RECEIVER  TRUE;
        REQUIRED  TRUE;
        OUTPUT    db;
        INPUT     db;
        PARAMETER i_aal_type = 3;              // 0, 3, 4, 5
        PARAMETER i_dataflow = 2;              // 2 = multicast, 0 = unicast
        USERS     10;
        PROVIDERS 1;
        CHILDS    3;
        TIME      30 * ys;
        MEMORY    1500 * bytes;
        }

DEFMECH M_146_254_PXY_020_DLPI_SUN4_USR
        {
        TYPE      PXY;
        SENDER    TRUE;
        RECEIVER  TRUE;
        REQUIRED  TRUE;
        OUTPUT    db;
        INPUT     db;
        USERS     1;
        PROVIDERS 0;
        CHILDS    1;
        TIME      100 * ys + i_sdu_size * 100000 * ys;
        MEMORY    10 * bytes;
        }

DEFMECH M_146_254_001_023_TEMPO_RATE
        {
        TYPE      SRC;
        SENDER    TRUE;
        RECEIVER  FALSE;
        REQUIRED  TRUE;
        OUTPUT    db;
        INPUT     db;
        PARAMETER d_rate = 1;                  // rate seconds
        USERS     1;
        PROVIDERS 1;
        CHILDS    1;
        TIME      30 * ys;
        MEMORY    100 * bytes;
        }

DEFMECH M_146_254_100_RATE_CREDIT
        {
        TYPE      SRC;
        SENDER    TRUE;
        RECEIVER  FALSE;
        REQUIRED  TRUE;
        OUTPUT    db;
        INPUT     db;
        PARAMETER d_rate = 0.1;
        PARAMETER i_max_queued_no = 10;
        PARAMETER i_water_level = 2;
        PARAMETER Number_of_Modules = 1;
        USERS     1;
        PROVIDERS 1;
        CHILDS    1;
        TIME      30 * ys;
        MEMORY    i_max_queued_no * i_max_sdu_size * bytes;
```

```
        }

DEFMECH M_146_254_003_001_ENCAP
        {
        TYPE      SRC, SNK;
        SENDER    TRUE;
        RECEIVER  TRUE;
        REQUIRED  DSB_Damage_Notify | DSB_Damage_Delete | DSB_Loss_Notify;
        OUTPUT    db;
        INPUT     db;
        PARAMETER i_algorithm = DSQ_Algorithm_Number  DEFAULT 4;
        PARAMETER i_icv_type  = DSQ_ICV_Type DEFAULT 1;
        PARAMETER Number_of_Modules = 1;
        USERS     1;
        PROVIDERS 1;
        CHILDS    1;
        TIME      135 * ys + i_sdu_size * 0.539 * ys;
        MEMORY    2000 * bytes;
        }

DEFMECH M_146_254_003_003_INTEG
        {
        TYPE      SRC, SNK;
        SENDER    TRUE;
        RECEIVER  TRUE;
        REQUIRED  DSB_Damage_Notify | DSB_Damage_Delete | DSB_Loss_Notify;
        OUTPUT    db;
        INPUT     db;
        PARAMETER i_algorithm = DSQ_Algorithm_Number  DEFAULT 4;
        PARAMETER i_chaining_mode = DSQ_Chaining_Mode DEFAULT 5;
        PARAMETER Number_of_Modules = 1;
        USERS     1;
        PROVIDERS 1;
        CHILDS    1;
        TIME      20 * ys + i_max_sdu_size * 30 * ys;
        MEMORY    1500 * bytes;
        }

DEFMECH M_146_254_003_003_CONF
        {
        TYPE      SRC, SNK;
        SENDER    TRUE;
        RECEIVER  TRUE;
        REQUIRED  TRUE;
        OUTPUT    db;
        INPUT     db;
        PARAMETER i_algorithm = DSQ_Algorithm_Number  DEFAULT 4;
        PARAMETER i_chaining_mode = DSQ_Chaining_Mode DEFAULT 5;
        USERS     1;
        PROVIDERS 1;
        CHILDS    1;
        TIME      135 * ys + i_max_sdu_size * 0.539 * ys;
        MEMORY    1500 * bytes;
        }

// Sender side only

DEFMECH M_146_254_001_020_TEMPO_SRC
        {
        TYPE      SRC;
        SENDER    TRUE;
        RECEIVER  FALSE;
        REQUIRED  TRUE;
        OUTPUT    db
        INPUT     db;
        PARAMETER i_max_sdu_size = 128000;  // max. TSDU size in bytes
```

```
        PARAMETER i_pdu_size = 4000;        // block_len: size of segments
        PARAMETER i_No_of_Retransmissions = DSQ_Max_Retransmissions DEFAULT 5;
        PARAMETER i_window_size = 10;
        PARAMETER i_sink_no = 1;            // number of sinks
        PARAMETER i_ack_timeout = 10;       // timeout (integer) value sec
        PARAMETER i_probe_timeout = 60;     // timeout (integer) value sec
        USERS     1;
        PROVIDERS 1;
        CHILDS    1;
        TIME      40 + (i_max_sdu_size/i_pdu_size+1) * 25    // sending PDUs and
                  + (i_max_sdu_size/i_pdu_size+1) * 50 * ys; // receiving ACKs
        MEMORY    i_window_size * i_max_sdu_size * bytes;
        }

DEFMECH M_146_254_001_030_TEMPO_PDU_COMP
        {
        TYPE      SRC;
        SENDER    TRUE;
        RECEIVER  FALSE;
        REQUIRED  TRUE;
        OUTPUT    db;
        INPUT     db;
        USERS     1;
        PROVIDERS 1;
        CHILDS    1;
        TIME      40 * ys;
        MEMORY    50 * bytes;
        }

DEFMECH M_146_254_001_025_TEMPO_BLK
        {
        TYPE      SRC;
        SENDER    TRUE;
        RECEIVER  FALSE;
        REQUIRED  TRUE;
        OUTPUT    db;
        INPUT     db;
        PARAMETER i_watermark = 2048;
        PARAMETER i_timeout = 5;
        USERS     1;
        PROVIDERS 1;
        CHILDS    1;
        TIME      50 * ys;
        MEMORY    i_watermark * bytes;
        }

// Receiver side only

DEFMECH M_146_254_001_026_TEMPO_DEBLK
        {
        TYPE      SNK;
        SENDER    FALSE;
        RECEIVER  TRUE;
        REQUIRED  TRUE;
        OUTPUT    db;
        INPUT     db;
        USERS     1;
        PROVIDERS 1;
        CHILDS    1;
        TIME      50 * ys;
        MEMORY    i_max_sdu_size * bytes;
        }

DEFMECH M_146_254_001_021_TEMPO_SNK
        {
        TYPE      SNK;
```

```
       SENDER    FALSE;
       RECEIVER  TRUE;
       REQUIRED  TRUE;
       OUTPUT    db;
       INPUT     db;
       PARAMETER i_max_sdu_size = 128000;  // max. TSDU size in byte
       PARAMETER i_pdu_size = 4000;        // block_len: size of segments
       PARAMETER i_window_size = 10;
       PARAMETER i_timeout = 120;
       USERS     1;
       PROVIDERS 1;
       CHILDS    1;
       TIME      40 + (i_max_sdu_size/i_pdu_size+1) * 25    // receiving PDUs and
                 + (i_max_sdu_size/i_pdu_size+1) * 50 * ys; // sending ACKs
       MEMORY    i_window_size * i_max_sdu_size * bytes;
       }

DEFMECH M_146_254_001_031_TEMPO_PDU_DECOMP
       {
       TYPE      SNK;
       SENDER    FALSE;
       RECEIVER  TRUE;
       REQUIRED  TRUE;
       OUTPUT    db;
       INPUT     db;
       USERS     1;
       PROVIDERS 1;
       CHILDS    1;
       TIME      40 * ys;
       MEMORY    50 * bytes;
       }

DEFMECH M_146_254_001_024_TEMPO_JITTER
       {
       TYPE      SNK;
       SENDER    FALSE;
       RECEIVER  TRUE;
       REQUIRED  TRUE;
       OUTPUT    db;
       INPUT     db;
       PARAMETER d_delay = 1.0;            // initial delay in sec
       PARAMETER d_rate  = 0.2;            // rate in sec
       USERS     1;
       PROVIDERS 1;
       CHILDS    1;
       TIME      50 * ys;
       MEMORY    1000 * bytes;
       }
```