Da CaPo++ - communication support for distributed applications

Da CaPo++
– Communication Support for Distributed Applications –

Burkhard Stiller, Daniel Bauer, Germano Caronni, Christina Class,
Christian Conrad, Bernhard Plattner, Martin Vogt, Marcel Waldvogel

Institut für Technische Informatik und Kommunikationsnetze

ETH Zürich, Gloriastrasse 35, CH – 8092 Zürich, Switzerland

E-Mails: <last-name> @ tik.ee.ethz.ch

Abstract

As the variety of applications, especially distributed multimedia applications, explodes, their requirements on communication-relevant tasks increase. Besides a communication architecture for dealing with traditional communication protocol processing, multicast features and security requirements have to be considered in an integrated manner. Therefore, a multicast-capable and security-aware communication subsystem is developed to provide necessary functionality to support an integrated set of reuseable application elements, e.g., *audio/video-presentation, application sharing, picture phone, extended WWW browser, tele-banking, or tele-seminar. The main goal includes the provision of a real-world application framework, where different traditional and emerging applications can be managed modularily. Their needs and communication demands in terms of Quality-of-Service (QoS) attributes are specified by numerical values,* e.g., *bandwidth requirements, delay boundaries, reliability issues. Furthermore, functional features, such as multicast groups, encryption desires, or authentication requests can be selected. In turn, the developed communication subsystem allows for the preparation of flexibly adjusted communication protocols that provide requested functionality,* e.g., *error control schemes, multicast addressing, encryption, or authentication. Finally, a best suited service for these application requests is offered.*

1. Introduction

As bandwidth increased and error rates decreased over the last few years, nowadays multi-media data can be shared between geographically distributed locations. Therefore, within a highly distributed environment communication facilities between different locations, such as universities, companies, research organizations, or enterprises, grow quite significantly. A great number of distributed applications, most of them including multi-media data, have to be supported by well-suited communication subsystems including communication protocols and, additionally, by high-speed networks, *e.g.*, Asynchronous Transfer Mode (ATM). Nevertheless, the communication subsystem performs in many cases as the bottleneck due to the fact that communication protocols do not offer proper protocol functions for handling continuous data adequately or the run-time environment for protocol processing is not able to cope with high data rates.

Under these circumstances, emerging applications tend to integrate various features and functionalities that used to be considered separately in traditional approaches. As an example tele-seminar shows, communication protocols have to provide data transmission functions for audio and video in addition to proper error detection and correction functions dealing with continuous media. At least a uni-directional one-to-many communication between a single trainer and multiple students has to be established also. Scenarios involving financial transactions or transmission of confidential data require a variable degree of security, such as for encryption or authentication. An integrated solution for the communication subsystem has to provide this functionality for real-world applications.

Following the basic principle of a best reachable transparency between applications and communication subsystems, a homogeneous interface between them has to hide all communication-relevant details from the application. On one hand this transparency exploites the independency of applicati-

ons and communication subsystems. On the other hand, a lack of control information transfer from the application to the underlying communication subsystem leads to architectures that are characterized by inherent inefficiencies.

For these reasons, an application programming interface has been designed. Within a set-up phase of a required association between remotely distributed peers this interface allows for the transfer of application requirements in terms of Quality-of-Service (QoS) attributes. During the actual data transfer phase from the application to the remote peer and vice versa, a re-negotiation of these QoS attributes is possible. Consequently, certain requirements can be adjusted and change requests can be satisfied. QoS attributes include numerical values for, *e.g.*, bandwidth, delay, or bit error rates, they allow for the specification of certain security requirements, such as various degrees of privacy or different mechanisms for authentication, or they include attributes for multicast groups and their addressing. The data transfer phase always is strictly bound to previously determined QoS attributes unless they are marked explicitely negotiable. In this case a flexible adaptation of QoS values or of communication protocol functionality has to take place.

1.1 Environmental Prerequisites and Goals

The communication subsystem is based on a concept for Dynamically Configuring Protocols (Da CaPo) where protocol functionality is determined by application requirements in terms of Quality-of-Service attributes. This dynamic process of configuration is supported by a number of components, *e.g.*, the Configuration and Resource Allocation component (CoRA) for configuring the communication protocol, the Connection Manager for setting-up associations between multiple participants, and a Monitor for monitoring current status of QoS values. The communication subsystem /PPVW93/ is already supported by a run-time system specifically adapted for processing protocol-relevant task /VPPW93/. Traditional networks using Transmission Control Protocol (TCP) and Internet Protocol (IP) or high speed networks, *e.g.*, applying ATM communication protocols are supported.

However, security and multicast features within a communication subsystem and an application framework have not been considered in an integrated manner yet. Therefore, current extensions in the so-called “Da CaPo++ communication subsystem“ include a multicast-capable association set-up, handled by an extended Connection Manager, and multicast-capable transport protocol functions. In addition, security modules, *e.g.*, encryption and authentication, are integrated to offer cryptographical functionality. Due to the fact that banking environments require a highly degree of security, the range of configurable and parametrizable security functionality includes varying degrees of authentication and privacy. A set of QoS attributes allows for the specification of security algorithms or for the setting of lifetime properties of cryptographic keys.

In addition to these basic communication-relevant tasks residing within the communication subsystem, the development of an application framework for banking environments and tele-seminars has been done. The framework includes support of modular multimedia applications to be used on top of the communication subsystem. Including application components, applications, and application scenarios the framework offers this set of re-useable application elements to construct future applications. Therefore, a three-level hierarchy of building blocks for these elemets provide traditional, special, and future services as necessary.

Finally, the design of the communication subsystem as well as the application framework is independent of any specific network infrastructure, as long as it offers minimal features or QoS, *e.g.*, bandwidth, delay, or bit error rates that are requested by an application and that are required to fulfill user demands entirely.

This paper is organized in the following manner. The subsequent Section 2 covers briefly main issues of the communication subsystem‘s architecture, its current state of implementation, and related work. Section 3 presents the design and implementation of the application programming interface between the subsystem and multiple applications. Furthermore, Section 4 covers relevant multicast and security enhancements. Section 5 deals with the application framework and its elements. Finally, Section 6 summarizes the work and draws conclusions.

2. Survey of the Da CaPo++ Communication Subsystem

The communication subsystem of Da CaPo++ provides the possibility to configure end-system communication protocols. This process is based on currently available application requirements, local resources, and network prerequisites in terms of QoS attributes. The result of this configuration process is defined as a best possible communication protocol under these specifically considered circumstances. Basic building blocks, in particular protocol functions and mechanisms as well as their properties, form the basis for the configuration process. Configuration is directly supported by a number of components within the subsystem (cf. FIGURE 1.). The attribute translation accepts application requirements and translates them into a structure suitable for the configuration and resource allocation. An appropriate protocol graph is calculated, consisting of a formal description of selected protocol functions /PPVW93/. Finally, the protocol graph is locally instantiated by the data transport component and distributed to peer systems by the Connection Manager. The Security Manager validates users and applications and assures that necessary modules are contained within the protocol graph to comply with requested security requirements. Finally, a monitor supervises the execution of communication protocols within the end-system and issues notifications if application requirements are violated.

An application requests a certain type of service from the subsystem, while delivering a set of QoS attributes and values. The best suitable protocol configuration consisting out of various protocol functions called C-modules (communication-modules), *e.g.*, a selective retransmission function, a compression function, and a segmentation function, will be determined applying a configuration algorithm. A specific A-module (application-module) provides user data access to the communication protocol. Local system resources, such as memory, buffer space, and processing time, and network resources, *e.g.*, available bandwidth or delay, are considered additionally to determine important properties of the run-time environment or the infrastructure access, *e.g.*, in terms of adapters. Different networks are supported, in particular Ethernet, IP, or ATM, by including a special T-module (transport-module) in the protocol.

On a module basis a specific run-time system supports protocol processing tasks. Modules are the basic element within the run-time environment to perform every type of processing, similar to STREAMS modules /GoCo94/, but usually of a finer granularity. Modules offer an extended, but generic interface for user data manipulation and control, for protocol configuration, out of band data transmission, and monitoring as well. Within the communication subsystem a communication protocol, represented by a protocol graph, acts as the basic execution unit, which in turn consists of a well-defined number of modules. The communication subsystem associates three processing tasks with each protocol which deal in decreasing order of priority with monitoring, data movement from the network up to the application and from the application down to the network. The monitoring task periodically checks the actual performance of the protocol against application requirements and triggers a reconfiguration if the latter are violated. Both data movement tasks are instances of the data transport algorithm. This algorithm waits for a data packet, either from the application or from the network, and requests the modules to do the protocol processing. Although the basic structure of a Da CaPo++ communication protocol looks similar to a sequence of STREAM modules, it

differs in protocol construction and execution. While a STREAM has to be built by the application and is rather static, Da CaPo++ communication protocols are tailored to the actual application needs and may be adapted dynamically to changing network and end-system properties. Furthermore, Da CaPo++ knows the execution properties of protocols and has control over processing elements. This allows for selecting of optimized strategies of data movement and buffer allocation. Compared to the *x*-Kernel, Da CaPo++ communication protocols usually remain stable for many data packets and the processing is optimized for this protocol. Compared to this, each packet transmitted by the *x*-Kernel carries the protocol information and, therefore, has to be scheduled and processed individually, thus leading to additional switching between processing elements.

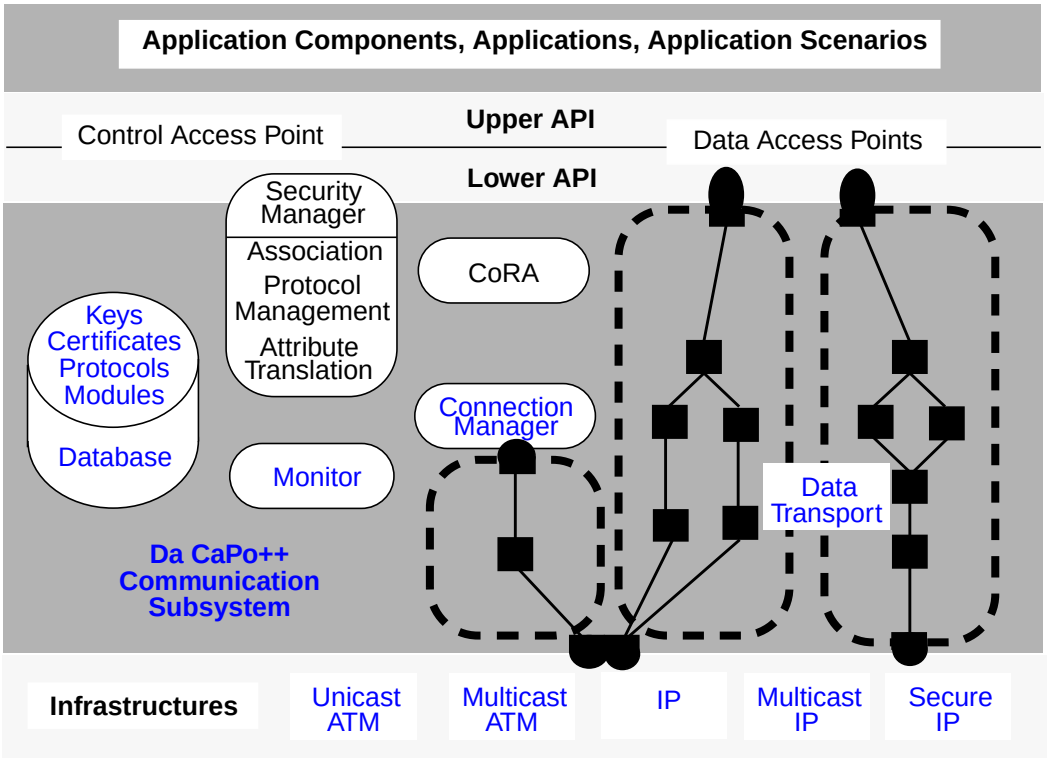

FIGURE 1. The Da CaPo++ Communication Subsystem

Currently, the communication subsystem supports multiple protocols for different data streams at the same time, *e.g.*, a picture phone handles outgoing and incoming audio and video data streams separately which are represented by four different uni-directional flows (sending and receiving audio and video). In general, a number of flows – representing uni-directional transmissions of user data with similar QoS requirements – are coordinated within a session. Sessions are used to enable a certain type of synchronization between multiple flows. The Da CaPo++ communication subsystem is responsible for handling data flows and protocol processing. Da CaPo++ offers unicast- and multicast-services to applications via its application programming interface.

2.1 Related Work

Due to the range of relevant topics that are integrated in the presented approach, a variety of different areas of related work is important. Main aspects are covered by the following areas. Flexible communication subsystems have been designed to support high-performance applications, such as F-CSS /ZSTa93/ or ADAPTIVE /ScSu93/. In addition, QoS concepts have been designed and evaluated in /Dant94/, /CCHu94/, and /Stil96/ to allow for sophisticated characterizations of applications and specification of communication requirements. In detail, security issues are dealt by a

number of approaches, *e.g.*, basic work in /VoKe83/, specific algorithms and protocols are presented in /Schn96/. A good overview of security relevant policies and solutions may be found in /Purs93/. Many algorithms handle multicast communications, such as initial work in /Deer91/ and /CaDe92/ detects. Additionally, an ample spectrum of projects deal with the handling of multimedia applications. Amongst others, examples comprise transmission of continuous media via the World Wide Web /WFWe96/, /Soo94/ or video conference applications /IsTa93/. Finally, application programming interfaces in object-oriented environments have been studied, *e.g.*, in /Schm92/. However, throughout this paper the presented approach handles multi-media applications in an integrated manner, including a close cooperation between them, the application programming interface, Quality-of-Service concepts, and the communication subsystem itself.

3. Interface for Applications and the Communication Subsystem

The Da CaPo++ communication subsystem as presented in Section 2 is implemented as a modular system, including a native application programming interface. This internal interface does not provide the functionality an application programmer would expect. To correct this situation, the presented solution provides a set of high-level abstractions that hide all communication subsystem internal details from applications. These high-level abstractions introduce an additional processing layer between the application and the communication subsystem, which may be coupled with a loss of efficiency, if no care is taken in the design phase. Thus, the main challenge in designing the application programming interface (API) for Da CaPo++ has been a convenient trade-off between “ease-of-use” and efficiency. On one hand the offered abstractions allow for a better understanding of the communication subsystem features and make the application code more readable. On the other hand, they guarantee that the application programmer is only granted access to necessary information. As an important advantage these abstractions provide a greater security and reliability by strictly restricting accesses.

3.1 A-Modules

A-modules are located at the boundary of the Da CaPo++ communication subsystem. They offer at their upper side an interface to the application and upper API, respectively, and on their lower side an interface to the Da CaPo++ data transport algorithm (cf. Section 2). This duality can be best viewed in FIGURE 2. In general, A-modules provide three tasks, namely passing control information to/from the application and user data to/from the communication subsystem, and mapping of QoS requirements. As a sending A-module may receive simultaneously data and control information from the application, a policy has to be set up to deal with this concurrency. The solution selected consists in giving higher priority to control versus data information. The Da CaPo++ communication subsystem provides a set of predefined QoS attributes such as throughput, delay, delay-jitter, error-rate. These basic attributes are not necessarily well-suited to characterize the behavior of some specialized applications directly. *E.g.*, in a video application context, the most significant Da CaPo++ attribute is throughput, although it is likely that a programmer or a user would rather speak in terms of user-level parameters such as frames per second, colour depth, image size, and compression factor. Actually, the effective throughput can be computed by a combination of the three latter values. QoS mapping is performed by every A-module, which is responsible for the mapping of specialized user-level parameters into internal Da CaPo++ attributes.

3.2 Interface Structure and Objects

In order to manage these mappings of resources, such as memory and processing power, a workstation hosts a single communication subsystem that is implemented within one process. Applications have their private domains and communicate with the Da CaPo++ communication subsystem via inter-process communication mechanisms (IPC). Therefore, the upper API (cf. FIGURE 2.) is linked to the application, whereas the lower API remains the interface to the communication subsystem. This part of the API consists of a single control access point which allows for manipulating and configuring entire sessions that consist of several flows. Additionally, data access points serve as means to specify the handling of user data after protocol processing is completed by A-modules (cf. Subsection 3.1). There are basically two data access points (data and control information) for each protocol graph. The control access point might be used to transfer the specification of the window in which, *e.g.*, video has to be displayed. The upper API provides all necessary abstractions to implement an application using the Da CaPo++ communication subsystem. These abstractions are on one hand the `DaCaPoClient` object and on the other hand the `Session` and `Flow` objects. Both abstractions are explained in the two following paragraphs, while applying terminology as used within object-oriented programming languages, such as C++. Finally the inter-process communication between upper and lower API is examined in more detail.

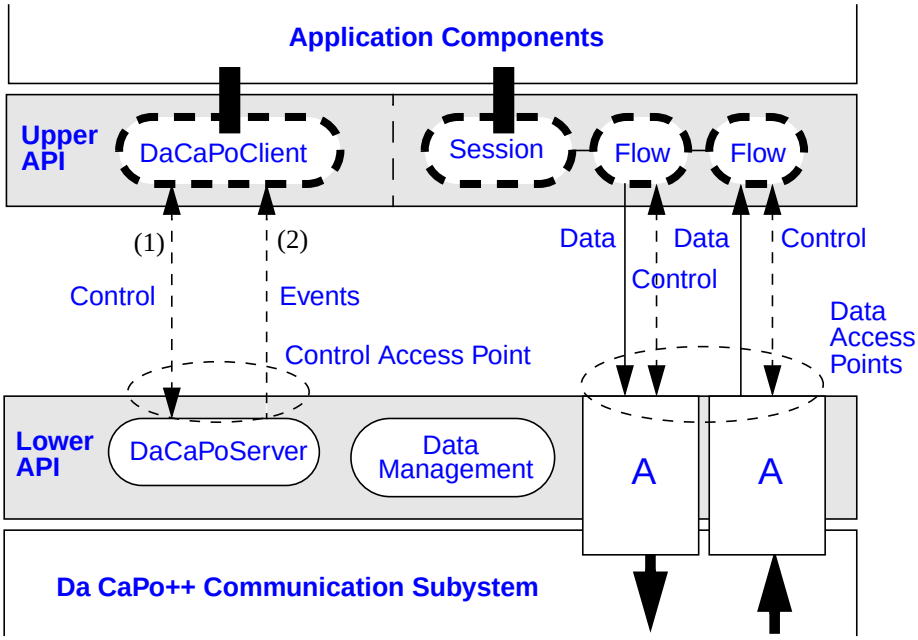

FIGURE 2. The Da CaPo++ Application Programming Interface

3.2.1 The `DaCaPoClient` Object

Before being authorized to work with a Da CaPo++ communication subsystem, a user has to be registered properly through an application. This is performed via the `DaCaPoClient` object which sets up a connection to the DaCaPoServer component (cf.FIGURE 2.). If this registration succeeds, a reference on this instance of the `DaCaPoClient` will be implicitely used for each transaction between upper and lower API identifying further communications. Moreover, the `DaCaPoClient` object is responsible for processing incoming events from the Da CaPo++ communication subsystem. This is performed by maintaining a list of session event callback functions, while the objects act like an event demultiplexor. Therefore, for the application programmer the only visible interface of the `DaCaPoClient` object consists of its constructor which expects as parameters security relevant parameters, such as the user name or key id (cf. Paragraph 4.2.1).

3.2.2 `Session` and `Flow` objects

Sessions and flows are both application abstractions for Da CaPo++ services and protocol graphs, respectively. From an application point of view, a flow encompasses both sending or receiving of data/control information to/from a dedicated A-module. Thus, flows come in different flavors as they have to reflect internal properties of protocol graphs. Flow properties can also be decomposed according to the direction (either a sending or receiving protocol graph), the data type (either audio, video, user data or any user-defined type), and finally the way data is processed in the A-module. A-modules either interact with a file, a hardware device, such as a camera or a microphone, or directly with the application through an IPC mechanism. Therefore, per data type a set of 6 instantiable Flow classes, according to the direction and the origin/destination of data are defined. In order to facilitate the management of several flows, the concept of a session has been introduced. A session encompasses several flows which may be synchronized hierarchically (*e.g.*, audio with video).

```
CREATOR MULTICAST
SESSION PicturePhone;

FLOW VIDEO_SEND_DEVICE VideoOut;
     FPS       10   15        WF_LIN;
     DELAY     0.1  0.15      WF_CONST;
     COLOR     NO;
END FLOW;

FLOW AUDIO_SEND_DEVICE AudioOut;
     DELAY_JITTER   0.1  0.2       WF_EXP;
     DELAY          0.1  0.25      WF_LIN;
END FLOW;

SYNCHRONIZE VideoOut WITH AudioOut;

END SESSION PicturePhone;
```

FIGURE 3. Example of a Session Configuration File

The creation of sessions and flows can either be done by static or dynamic approaches.

- The most straightforward solution consists in declaring a static `Session` class with all necessary `Flow` objects declared as members. The main advantage of this solution is on one hand its simplicity, because members and methods of the session are directly addressed via access operators such as "." or "->". On the other hand, there exist severe drawbacks. Firstly, this is a strong static solution which prevents any ulterior change in the session description, mainly for synchronization and adding/removal of flows within a session. Secondly, a huge number of function calls is necessary to set up a complete `Session` object, including all flows, their requirements, and synchronization relations.
- The dynamic solution through the use of a session configuration file (cf. FIGURE 3.) is flexible as it allows the application programmer to write a human-readable configuration file which can be parsed automatically. `Session` and `Flow` objects are transparently created. This solution reduces the amount of function calls in the application code and permits for further changes in the session's structure. However, this increases the complexity as flow objects can no longer be directly addressed via access operators. Additional functionality has to be provided to access flows via their names.

Due to its greater flexibility, the dynamic variant has been selected. The corresponding upper API functionality is presented in Table 1. `Flow` objects are addressed via their names (*e.g.*, `VideoOut` or `AudioOut` in FIGURE 3.) through session-level functions. Avoiding several string comparisons for

efficiency reasons each time a flow is addressed, a bypass is provided with the `GetFlowDesc()` function which returns a reference on the desired flow in the session.

Function	Description
`Session()`	A new session is locally created with a configuration file and a reference on the current `DaCaPoClient` object. This function is actually the C++ Session object's constructor. It is important to notice that no interaction with the network is performed at this stage.
`Connect()` `Listen()`	These functions allow a session to either actively connect to a peer (several if multicast) or to wait for incoming connect events. A callback function is also provided as parameter to process incoming session related events.
`Configure()`	All flows belonging to the session are configured according to the new application requirements.
`Activate()` `Deactivate()`	All flows start or stop respectively sending/receiving data. This corresponds to the activation or deactivation respectively of the data transport algorithm.
`Close()`	The connection with the peer(s) is stopped and destroyed. Resources are returned to the Da CaPo++ communication subsystem and made available again.
`GetFlowDesc()`	As `Flow` objects can only be accessed in a session through their string name , this functions provides a "direct" reference on the `Flow` object without having to perform each time expensive string comparisons.
`SetReqFlow()`	New application requirements are transmitted to the core system. This can be done when creating a new session or before issueing a reconfiguration. The corresponding flow is identified by the flow descriptor and obtained by `GetFlowDesc()`.
`GetReqFlow()`	The `GetReqFlow()` function returns the actual configured values of required attributes, they may be different to those set initially by `SetReqFlow()`. The corresponding flow is identified by the flow descriptor and obtained by `GetFlowDesc()`.
`SendDataFlow()` `RecvDataFlow()`	Sending/Receiving of data to/from the corresponding A-module. Both these functions are only provided by the flows which receive/send data directly from/to the application. The corresponding flow is identified by the flow descriptor and obtained by `GetFlowDesc()`.
`SendCtrlFlow()` `RecvCtrlFlow()`	Sending/Receiving of control information to/from the corresponding A-module. Both these functions must be made available for each flow. The corresponding flow is identified by the flow descriptor and obtained by `GetFlowDesc()`.

TABLE 1. Functionality of `Session` and `Flow` Objects

The configuration file parsing process can be decomposed into two distinct phases. Firstly, the script syntax is checked in the upper API to ensure that the file obeys grammar rules. Secondly, a `Session` object is created, which opens a connection to the lower API part and registers itself. If it receives a positive acknowledgement from the lower API, it finally dynamically creates every `Flow` object, one at a time, and propagates the responsibility to properly register with its entire application requirements. When completed, the `Session` object asks for a definitive registration identification of the new session and closes the communication with the lower API afterwards.

3.2.3 IPC between Applications and the Da CaPo++ Communication Subsystem

In communicating applications an IPC mechanism is a critical issue for efficiency and reliability. In the API, special care has been given to classical culprits comprising unnecessary data copying and superfluous system calls. The properties of each IPC communication link (cf. FIGURE 2.) have been studied in detail. Three kinds of communication types corresponding to service and data access points are distinguished and characterized as follows:

- A bidirectional communication for control information between applications and the Da CaPo++ communication subsystem: Performance is not a critical issue for this channel as only general control information is propagated. On one hand, it is likely that there will be quite a lot of traffic at the very beginning during session creation and at the end to properly deallocate resources, but

not during life time of the communication association. On the other hand, the channel must be error-free and reliable. For sending of session-related events from the communication subsystem to applications, efficiency is critical and no losses can be tolerated.

- A uni-directional data communication for each flow of the session to send or receive data directly to or from the A-module: This channel is only made available for flows that exchange data directly with the application. Efficiency is a critical issue as data rates may be extremely large or require low delays.
- A bi-directional control communication for each flow of the session: This channel needs the same properties as the above mentioned uni-directional data channel except for data traffic which is bound to be much smaller.

Although the general control channel and the event channel (noted (1) and (2) in FIGURE 2.) are part of the same logical control access point, different physical IPC mechanisms have to be implemented due to their different properties. The general control channel is implemented over a UNIX domain socket, as it provides a reliable data transfer between two processes located on the same machine. All other data transfers are implemented by shared-memory with semaphore synchronization. This offers efficiency by restricting data copying (compared with traditional socket based solutions). A further distinctions have been made, whether the packet size is constant or not, for events and notifications, or for general application data. Shared-memory is implemented and managed like a circular buffer. Therefore, the properties of the packet size eventually lead to significant optimizations.

3.2.4 Buffer Management

The previous Paragraph 3.2.3 introduced a shared-memory based IPC mechanism to transmit data from the application process to the Da CaPo++ communication subsystem. As Da CaPo++ is located in a user process, this means that unavoidable data copying will take place when leaving the Da CaPo++ process to enter the kernel. A unified buffer management is applied, where the application is allowed to allocate memory for to be transmitted data packets. No data copying takes place until the data packet is ready to be sent into a network. Therefore, the interface to the semaphore synchronized circular buffer is extended to meet this functionality. Additionally, it is possible to mark data packets that must not be deallocated instantaneously when they leave the machine, *e.g.*, when using a retransmission scheme through acknowledgements. Finally, several packets can be linked together, *e.g.*, when gathering several data packets to finally perform a more efficient compression.

4. Integration of Security and Multicast Capabilities

Security and multicasting are both elements that are gaining significance in today‘s networks. Multicasting is especially useful in the context of high-volume multimedia applications, where a group of users wants to share the same information, *e.g.*, follow the same documentary movie, or participate in a teleconference. Efficient multicasting saves considerable ressources in the sending end system and in the network infrastructure. Up to now, ‘reliable’ multicasting was not an integral part of advanced communication systems, and its QoS aspects have not been fully valued. Additionally, security is gaining importance due to the increased commercial use of todays open networks. Data protection and the authentication of participants have to be provided by modern approaches to form the basis for real-world applications.

4.1 Security Capabilities

Securing communication with Da CaPo++ is achieved by defining protocols that include encrypting and authenticating modules. Depending on abstract security requirements that may be specified by the application, the configuration process will employ these modules, taking into account that security may be provided by the underlying infrastructure, *e.g.*, secure IP. A static key and certificate database allows for the application-independant storage and retrieval of public keys and related information. The actual control of security in Da CaPo++ is done by the Security Manager which consists of several building blocks (cf. FIGURE 1.). Security capabilities of the Da CaPo++ communication subsystem cover four different areas. Firstly, users have to identify themselves to Da CaPo++ and have to prove their identity. Secondly, applications that want to use Da CaPo++ in a secure fashion have to be identified and authenticated by Da CaPo++. Another important area is the machine-machine authentication that allows two Da CaPo++ endsystems to communicate in an authenticated and secure manner even if no security aware application or end-user is available. Finally, the fourth area covers the actual encryption and authentication of data that is transmitted over an unprotected network. The second and third area may be coalesced into one, if user authentication is done through the application. Such behavior is not encouraged, as it leads to the necessity of a multitude of 'logins' for the user. All four areas show different behavior depending on whether a delegation of the respective identity to the Da CaPo++ communication subsystem takes place. For simplicity, this is assumed to be the case.

The functionality of the Security Manager can be separated into three interoperating blocks, which comprises the association and authentication of users and applications, the attribute translation for QoS requirements, and protocol management consisting of module rekeying, event propagation, reconfiguration, and key management. To handle time dependant actions and check the current state of a running protocol the Security Manager possesses its private controlling thread in the Da CaPo++ communication subsystem. Over a dedicated application users may directly influence the behavior of Da CaPo++, independently from the application that they are currently using. They may induce actions like rekeying, switching security for one particular protocol graph on or off, generally controlling the behavior of protocols, and they are able to authenticate themselves and security-unaware applications over that interface.

4.1.1 Assuring Authenticity: Associations and Identities

Before employing a secure communication system, participants have to be securely identified and transferred user data must be attributable to them in a reliable fashion. This assumption ignores issues like frequently changing identities and desires for anonymity, but is only relevant if authentication is required. In an extreme scenario, all trusted parties that are involved have to be mutually authenticated. These parties consist of the end-systems on which Da CaPo++ is running, the users involved in the communication, and/or the applications actually producing and consuming data. In the model employed by the Da CaPo++ communication system these three entities are ordered hierarchically. If no user authenticity can be provided, application authenticity, and failing that, machine authenticity will be provided. The instances participating in the communication can express their minimal requirements and are notified upon connection establishment with whom they are actually communicating. Before communication can progress, users and/or applications involved are required to delegate their identities to the Da CaPo++ communication subsystem so that it can authenticate data on their behalf, and prove their identity to the peer. This mainly consists in giving a secret to Da CaPo++ with which identities can be authenticated. The given secret need not be the secret that was originally employed to prove identities, and may be usable only by

Da CaPo++ for a limited period of time and/or for a limited amout of authentications only. The typical case, nevertheless, will be a full delegation.

Public key values and user/application identities are stored in a global key and certificate database, where application identities consist of arbitrary, but structured strings identifying them. User identities may consist of a string containing RFC 822 E-mail addresses, bank account numbers, or any other kind of mutually accepted identifying information. Machines are identified by an address on which they are reachable in the infrastructure that is used to establish the connection.

The association block in the Security Manager of the Da CaPo++ communication subsystem verifys identities and notes which protocols are associated with which applications and users. It communicates this information to other endsystems, if needed and allowable. A dedicated controlling application, which is used for user authentication in the first place, if not done via individual applications, can be used to force modifications in these associations, *e.g.*, if a user wants to force an immediate dissociation from an application which turned byzantine.

4.1.2 Specifying and Translating Security Requirements

To express privacy and authentication requirements, applications have to pass attributes to Da CaPo++. The attributes are hierarchically ordered in a generic sense and may consist either of discrete values from a set of possibilites or specify a range of acceptable values. The attributes specifying security requirements are generally handled exactly like any other QoS attribute. This allows to employ the standard attribute passing and protocol configuration mechanisms of Da CaPo++ for the building of secure protocols. Special treatment is scarcely needed, *e.g.*, for an attribute containing keying material or connection set-up authentication requirements.

Security requirements needed for the configuration of secure protocols span a very wide range. To allow for a more transparent and algorithm independent handling in the application, the concept of requirement translation is introduced in Da CaPo++. An application specifying only generic application requirements will accept the defaults that the translation mechanism concludes as being corresponding concrete QoS parameters. An application may still specify as many detailed parameters as wanted, but may thus create a set of requirements which the system can not fulfill. In that case, no communication can be established. The results of such a translation depend on the available algorithms and machine power and on the state of the art in cryptography. If the translation process is kept up to date and the application uses generic security requirements, they will support adequate cryptographic mechanisms, not only at the time of creation, but in the future too.

The attribute translation block doing the translation actually resides in the A-module of each security-aware protocol and receives application requirements by way of the lower API, together with non-security related attributes. As the attributes are not interpreted by the API, but passed on transparently, no extension thereof is needed for new attributes.

4.1.3 Protocol Management, Reconfiguration, and Keying

A secure protocol which has been configured performes cryptographic operations. These may be of symmetric nature, *e.g.*, DES, IDEA, RC4 for encryption, and MD5, SHA for authentication support, or asymmetric, *e.g.*, RSA, Diffie-Hellman (DH) or El Gamal /Schn96/. In addition to traffic encryption and authentication they will allow for key exchange if rekeying is an issue and may allow for the receipt and processing of tokens providing sender- and receiver non-repudiation functionality.

The key management block of the Security Manager provides access to a database containing public and private keys, as the generation of authenticated keying material has been delegated to the communication subsystem. Key changes in the running protocol can thus automatically take place, the protocol management of the Security Manager provides asynchronous key changes. The only way for an application to change the properties of a secure protocol is to initiate a reconfiguration.

4.2 Integration of Security into the Da CaPo++ Communication Subsystem

Introducing security has an impact on nearly all parts of the original Da CaPo++ communication subsystem. This section identifies aspects, *e.g.*, QoS parameter, C-modules, protocols, or the Security Manager, that have been created or modified. Additionally, the Connection Manager may be changed to use a protocol that insures privacy, especially within the connection set-up time. These changes mainly result in the Connection Manager using a secure protocol graph by itself. The system architecture is included in FIGURE 1. Secure IP represents an infrastructure that already offers security. Future T-modules will have the intelligence needed to understand the existence of such a service and thus optimize the configured communication procol.

4.2.1 Security Support by the Application Programming Interface

The application programming interface (cf. Section 3.) transparently forwards application requirements which an A-module can translate into QoS parmeters (cf. Paragraph 4.2.2). Additionally, the API handles the identification/authentication issues and provides a method to forward events to the application. A transparent control channel is available, by which dedicated applications may communicate with the Security Manager, *e.g.*, to access the key manager for generating, storing, retrieving, and certifying keys and certificates.

The upper API has to process and forward the following information upon establishment of a controlling connection between the Da CaPo++ communication subsystem and applications: local user name, process id, global user name, global application identifier, user key ID, and passphrase. These information are passed on to the Security Manager (association block) by the lower API and verified. Afterwards the lower API receives a clearance or denial message from the Security Manager and acts accordingly. As secure protocols and keys can be defined and changed using the generic (re-)configuration mechanisms, no addition to the API is needed for this purpose.

For end-to-end authentication with non-repudiation the API offers two sets of functions allowing for an extended security protocol to perform, using slightly different semantics. When a flow is created or reconfigured to use a receiver-non-repudiation protocol, a proof of receipt will be generated for each received message. Above the API a message may be limited by arbitrary bounds, defined by start/stop pairs in the control flow. For the application level, the concept of a message has to be introduced, or, alternatively, synchronous end-to-end authentication/return-receipt requests can be initiated by one of the peers. Although continuous media have no fixed boundaries, they can be added by the application, *e.g.*, by requesting a return-receipt after each frame or per second.

4.2.2 Application Requirements and QoS Parameter

The following three classes of application requirements in terms of QoS parameter exist and are being elaborated in closer detail:

- Abstract application requirements are algorithm independant name-value pairs which define abstract properties. They may be of quantitative or qualitative nature and are mainly used by the attribute trsanslation to derive low-level requirements.

- Low-level requirements indicate concrete algorithms and specify parameters pertaining to them. They are either derived from the abstract requirements through the attribute translation process or directly specified by the application.
- Peer authentication requirements specify a minimal authentication to be achieved with the peer side, such that a connection can be established successfully. The actual identity of the peer side has still to be passed up to the application, such that admission control on application-dependant authorisation can be performed.

Abstract application requirements pertain to three different classes. They describe the algorithm independant properties to be used for encryption purposes, for data authentication, or they provide access to per-protocol keying material and policy. The algorithm independant representation of encryption is a name-value pair which expresses a quantitative amount of security to be achieved.

```
attribute abstract privacy = (NONE, 0.1, 0.5, 1..100, max.)
```

This security is expressed by assessing the strength of an algorithm to be used as currently known and estimating the amount of years it would take an enemy to break this particular transaction, assuming he invests one million dollars per year and takes the best possible non-invasive approach at breaking it. This assessment depends on rapidly changing data and the corresponding translation tables in the Da CaPo++ communication subsystem and has to be adapted regularly, to take newly discovered weaknesses of algorithms and price development of components into account.

At the same time, the encryption protocol can be optimized for the type of data to be transmitted. These are implicit preconditions which are evaluated by the protocol configuration process itself. Various preconditions could be:

```
preconditions set of values = JPEG, MPEG, H.261, CellB, ULaw, G.721, Wavelet,
error-free, ordered, none
```

This allows the deployment of encryption algorithms that are adapted to a specific kind of data, taking advantage of intimate knowledge of the inherent semantics to achieve a cost reduction for the encryption or authentication process. The currently supported preconditions are ‘ordered’ and ‘none’, as no special protocols are designed up to now. Depending on above preconditions, different modes of partial encryption/authentication are possible:

```
attribute partial processing = spacial axis, time axis, embedded control data
only, variance mode
```

Variance indicates that only areas with ‘significant content’ are to be encrypted, *e.g.*, areas in an image where ‘something happens’. Further requirements of this type exist, *e.g.*, to specify needed authentication features such as symmetric, asymmetric, and authentication providing receiver-non-repudiation, and to allow for partial or coarse-granular authentication.

As the name implies, **low-level requirements** specify or depend on very concrete algorithm specific behavior. Applications accessing them can influence the behavior of the Da CaPo++ communication subsystem concerning security very directly at the risk of not being always up-to-date on the actual strength of an algorithm.

While application requirements concerning the data transfer in itself are specified by the receiving and the sending side, **peer authentication requirements** do not currently exist. Data transfer properties are actually checked on runtime and compared to the application requirements. As soon as the specified limits are surpassed, the monitor is assumed to produce an appropriate event. Nevertheless, these requirements are implemented in the application itself, respectively in the upper API. Before accepting a communication peer, its authentication method and identity are passed up

from Da CaPo++ communication subsystem to the application. Afterwards, the application can decide if the authentication method that was used is sufficient or if the access is denied, because end-to-end authentication is not adequate concerning its requirements.

4.2.3 C-Modules and Communication Protocols

The provided C-modules for introducing security into the data transfer offer well known encryption algorithmssuch as DES, Triple DES and RC5 in electronic codebook (ECB) and cipher block chaining (CBC) mode, and provide alleged RC4 as stream cipher, and MD4 or MD5 for message digesting. A module for DH key exchange is used to establish shared secrets, if certified DH public values are used in the public key database. Alternativley, a RSA module encrypts and signs data to convey traffic encryption keys to peers and to sign message digests for non-repudiation purposes. They behave like Da CaPo++ modules. For the purpose of internal rekeying and user driven (not application driven) security control, they have an additional interface directly linked with the protocol control block of the Security Manager and announce themselves to the association block on initialization. Protocol functionality includes encryption, authentication, a combination, and non-repudiation of send or received data.

4.3 Multicasting

The Da CaPo++ communication subsytem offers unidirectional point-to-multipoint multicasting based on a multicast capable infrastructure. Multicast flows are created by the application programing interface of the Da CaPo++ communication subsystem that wants to send user data. Similar to unicast sessions, multicast flows are part of sessions, however, multicast sessions are exclusively controlled by the creator of the session which is exactly the only sender. The Da CaPo++ multicast paradigm uses receiver initiated join. New participants are allowed to join a running session, while joining automatically all flows inside this session. During the set-up phase multicast connections are supported by a multicast-capable Connection Manager and they are supported by multicast error control C-modules within the configured communication protocol.

4.3.1 Connection Set-up Support for Multicast Sessions

The Connection Manager (cf. FIGURE 1.) guarantees that a compatible protocol is used inside a Da CaPo++ session. Control over a multicast session lies solely with the connection manager of the creator. Additionally, it triggers local configuration and reconfiguration, distributes the resulting protocol graph and starts/stops the protocols. Actions taken by the Connection Manager are initiated either by the local application or by a remote connection manager, who is responsible for exactly one session. In case of a multicast session, the multicast-capable Connection Manager is used. A reconfiguration is based entirely on the creator's application requirements. Participants are not allowed to reconfigure a multicast session. Starting and stoping at a participant's site leads to a stop of the local execution of the lift algorthm, especially of the protocol processing. Therefore, no data packets are delivered to the participant's application. Moreover, the stop operation should only be executed by applications that tolerate data loss such as audio or video applications.

The tasks of the multicast-capable Connection Manager are different for the creator and the participants. The creator defines the access point where participants have to register when they want to join a multicast session. In order to do so, the participants send a join request to the creator. The creator replies with the current protocol graph and its properties, the application requirements. Participants use this protocol graph to instantiate the protocol. As soon as the protocol is instantiated, added participants are ready to receive data. Participants leave sessions by sending a leave message

to the creator. Leaving participants have no effect on other participants. Finally, if a creator leaves the session, the session is completely released.

4.3.2 Multicast Transport Protocols

Basically, two different classes of multicast transport protocols exist: reliable point-to-multipoint protocols and native point-to-multipoint protocols for audio and video. They are based on a multicast capable infrastructure.

The reliable multicast protocol uses an error control mechanism based on retransmissions for assuring correctness of user data transport. In order to avoid a packet implosion at the sender, a negative acknowledgement scheme is used. Receivers detect loss of data by comparing sequence numbers in arriving packets with the expected sequence number. Encountering a gap in the sequence, missing packets are requested from the sender. If the sender has no data to send, it sends a so-called heartbeat packet that contains the last sequence number only. The heartbeat packet enables recipients to detect packet losses, while regular user data is not being transmitted. Heartbeats are sent in well-defined increasing intervals. Requested retransmissions are multicast to the entire group. Duplicates are detected by receivers and discarded.

The native multicast protocol is used as a transport protocol for audio and video. It consists of a segmentation and reassembly mechanism and a transport mechanism. Error detection is optional since both IP multicasting and ATM Adaptation Layer 5 already provide error detection methods. This protocol is used in conjunction with a specialized A-module that directly issues or receives data from a multimedia device such as a video board or an audio device.

5. Distributed Application Framework

A broad variety of traditional and modern applications offers an ample range of user-oriented services. Therefore, a certain structure of functionality, control, and graphical user interfaces of these applications form an inherent part of application elements. They include, *e.g.*, audio and video transmissions, picture phones, video conferencing, tele-banking, tele-seminar, tele-shopping, tele-teaching. A number of characterizable differences between these applications exist, however, this spectrum of applications looks quite unstructured. For instance, a tele-seminar includes features and functionality of video conferencing; a picture phone includes inevitably the issueing and presentation of audio and video data. Additionally, the type of control applied and used within these applications is different. While data transfer requires a simple interface only, a picture phone has to offer a separate graphical user interface for sufficiently controling the handling and manipulation of audio and video data. Finally, a tele-seminar involves meta-control for integrating floor-control issues, managing and synchronizing video as well as audio, or adding new participants.

The basics for defining the application framework for Da CaPo++ comprise a leveled hierarchy. Especially a **three-level hierarchy** for application elements allows for a very flexible and modular design and implementation of a variety of application scenarios. The lowest level comprises application components that are placed directly via the specified application programming interface on top of the Da CaPo++ communication subsystem. In the center level applications are constructed of application components, additional application functionality, and a separately useable graphical user interfaces. In the upper level application scenarios are used to consolidate multiple applications. They provide extensive functionality and features for complex user requirements, including a specifically designed graphical user interface for control and meta-control purposes. These applica-

tion elements are placed in one of the levels based on their specific objectives and features and are defined as follows:

The **application component** forms the basic building block for the application framework and resides in the lowest level of the hierarchy. An application component defines differentiated and separately useable parts of traditional applications. Each of those provides a separated functionality only, a set of tightly bound features including an application programming interface, containing a native graphical user interface. Examples include but are not limited to, audio/video presentation, messaging service, or application sharing. Traditional **applications**, such as picture (video) or standard (voice) phone, or video conferencing, have been placed in the center of the hierarchy. However, within the framework they are functionally structured out of single or multiple application components. Additionally, applications provide a separate graphical user interface offering user control features for controling exactly this specific one only. Nevertheless, an application in this sense is able to run completely stand-alone. Finally, a huge variety of applications may be combined for designing complex **application scenarios** that provide functionality, graphical user interfaces, and meta-control interfaces to fulfill emerging user requirements in tele-operating environments. In the defined terminology, modern applications such as tele-seminar or tele-teaching belong to the level of application scenarios.

The approach of using reuseable application elements and code respectively within the application framework and the leveled structure describes a promising route towards flexible concepts. Application building blocks allow for the adjustable construction of applications, their control parts, and their graphical user interfaces. The integration of future applications and ergonomic user interfaces is possible by re-using already existing application components. Furthermore, the provision of new algorithms for, *e.g.*, group management or synchronization with increased performance behavior can be seamlessly migrated into existing application elements without effecting upper levels of the hierarchy. To exemplify the usefulness of a level below the application level, a number of applications is presented in the following Subsection 5.1. Application components are described in Subsection 5.2 afterwards.

5.1 Investigations of Applications

A subset of two applications in use for a real-world scenario have been investigated. As the main focus of the work is on the provision of a comfortable tele-banking and tele-seminar environment, a video conference application, and an extended WWW client (browser) and WWW server application.

5.1.1 Video Conference

The video conference application is a many-to-many (m : n) generalization of a native audio and video communication between two participants (1 : 1). Usually the number of senders (m) and receivers (n) is identical, but an asymmetry may be possible. The video conference application provides at each participant's site videos for the speakers and mixed audio coming from all other participants. Thus, based on the session and flow concept of Subsection 3.2 each participant has to create a multicast session containing a flow for sending audio and a flow for sending video to every other participant. Each time a new user joins the video conference, a new receiving session (containing two flows for audio and video) has to be dynamically created on every participating machine. Considering the symmetrical case with n senders and n receivers, at each participant's site there will be one multicast sending session and (n-1) receiving sessions. An example including one creator (C) and 2 participants (P1 and P2) is depicted in FIGURE 4. Firstly, a multicast control session is

set up from the creator to all participants via channel (1). Secondly, a multicast session for audio/video is established from C to P1 and P2 (2). On channel (1), C sends a token to P1 notifying to set up a multicast session to C and P2 (3). It is important to note that participant P1 is the creator of the multicast session (3). After P1 has completed the setup of the session, it informs C who transmits the token to P2. Similarly, P2 establishes a session to C and P1 (4). This application-level protocol is necessary because the Da CaPo++ communication subsystem by itself does not allow currently dynamical adding/removal of flows in a session. At this point it is either possible to define a more sophisticated Connection Manager handling dynamic flow creations within an existing session, or to design a group management application component in the upper API, which would have a counterpart in the Da CaPo++ communication subsystem. The second solution allows for tailoring dedicated solutions for various Computer Supported Cooperative Work (CSCW) applications, as not only the video conference application is likely to need this functionality. These properties motivate the set-up of an application protocol. Each potential receiver can be informed when it has to create a new session and who the creator (sender) is. Therefore, an application component called Multicast Support is provided (cf. Paragraph 5.2.2).

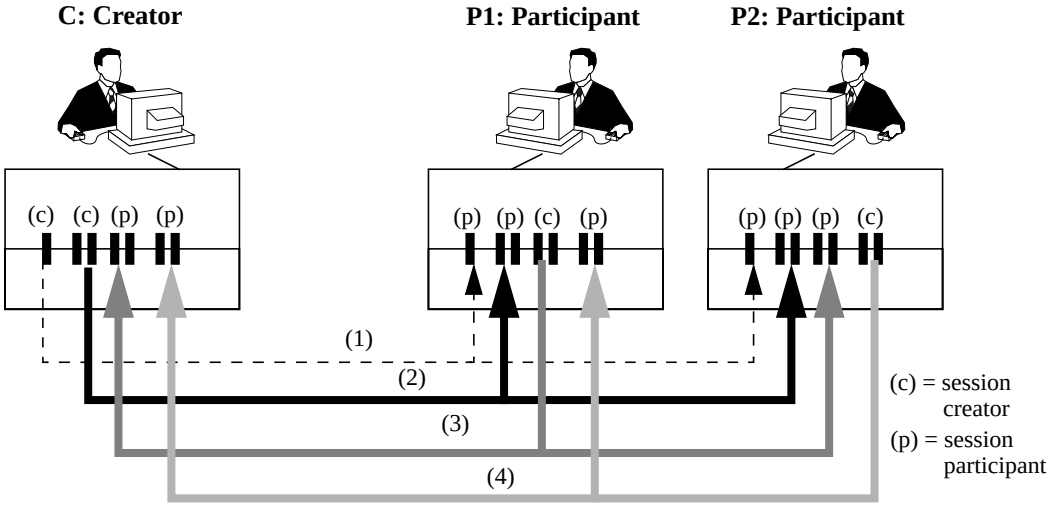

FIGURE 4. Example of a Video Conference Application Setup

5.1.2 Extended WWW Browser and Server

A World Wide Web (WWW) application in the traditional sense offers remote access to numerous information stored on multiple servers in the Internet. Applying a hypertext markup language, *e.g.*, HTML /BeCo95/, for the information presentation, a special purpose transfer protocol for transmitting these information, *e.g.*, Hypertext Transport Protocol (HTTP) /BFFr96/, is used. As HTTP defines a stateless request-response mechanism, the process of retreiving information follows an elementary approach. A WWW user selects a hyperlink, *e.g.*, a link to Internet draft documents, by selecting a specific keyword. In general, data is transmitted via an HTTP link and presented or processed on the user machine afterwards. This situation may result in inconvenient behavior. A time gap exists before the requested data is presented/processed. Disk space requirements on user machines may be exceeded due to an unexpectedly huge file. Finally, wasting bandwidth in case the user does not require the entire file data, does not utilze resources in a sufficient manner.

Due to these drawbacks, an Extended WWW Browser residing on top of the Da CaPo++ communication subsystem invokes a so-called Da CaPo++ File Client that establishes a Da CaPo++ link between the user machine and the server, where the requested data is stored. In this case data is transmitted over an adjusted Da CaPo++ link, which takes into account every data requirement,

such as animation data in real-time or video data with isochronous characteristics. As depicted in FIGURE 4., relevant information needed to establish the Da CaPo++ link encompasses, in particular session and flow specifications, name and address of the server storing the requested data, and an identifyer of the file(s) to be transmitted. These are included in a dedicated specification file identified by a “user-defined” Da CaPo++ Multipurpose Internet Mail Extensions (MIME) type /BoFr93/ and /Moor93/. Running an HTTP connection over TCP, this specification file is demanded by the Extended WWW Browser (1) and returned from the Extended WWW Server (2). Each hyperlink pointing to such a specification file is referred to as a Da CaPo++ hyperlink and is needed whenever a Da CaPo++ data transmission shall be invoked within a WWW environment. In general, the data to be transmitted is not restricted to be stored in a file, instead any Da CaPo++ application or application scenario may be invoked by such a Da CaPo++ hyperlink. The Extended WWW Browser recognizes the Da CaPo++ MIME type sent with the specification file and automatically starts up a Da CaPo++ File Client, which in turn establishes the Da CaPo++ link to the Da CaPo++ File Server to receive the requested data. This specialized data transmission and presentation/processing is handled by two different application components called Da CaPo++ File Server and Da CaPo++ File Client (cf. Paragraph 5.2.1). Furthermore, the Da CaPo++ File Server may be located on the same machine as the Extended WWW Server.

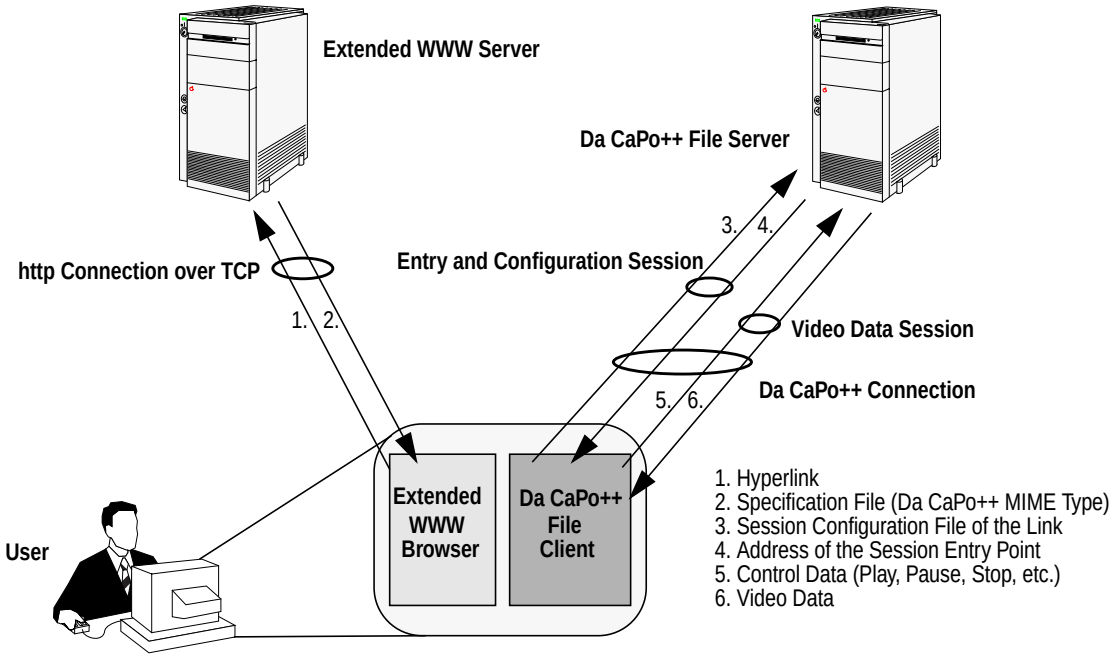

FIGURE 5. Extended WWW Browser in Case of Video Data Transmission

5.2 Study and Definition of Application Components

The following application components form basic ingredients of the application framework. Obviously, they are not limited to the described ones, but they offer a beneficial approach for supporting already presented applications, which in turn are important elements for application scenarios.

5.2.1 Da CaPo ++ File Server and Client

A specification file operates as input into the Da CaPo++ File Client (cf. FIGURE 4.), specifying, amongst others, name and address of the entry and configuration session on the Da CaPo++ File

Server and the session and flow specifications for the Da CaPo++ link. An initial Da CaPo++ link is established between client and server for the purpose of transmitting the session configuration to the server (3). The server creates a new session satisfying every flow specification requirement and transmits the address of the implicitely requested session's entry point to the client (4). Depending on the specifications in the specification file multiple sessions may be created. Afterwards, the initial Da CaPo++ link between client and server is closed to allow further client connects to this server at its specific initial entry point. Additionally, a Da CaPo++ link is established between the client and the newly created session of the server. This link satisfies all specified requirements and is used to transmit file data from the server to the client (6), and control data from the client to the server (5). In case of video and audio data, data presentation is performed by the corresponding A-module (cf. Subsection 3.1) on the client's machine. In general, data processing is performed within a specific application routine linked to or invoked by the Da CaPo++ File Client. A potential user is offered a variety of control facilities, *e.g.,* open file, close file, transmit data, in case of video and audio additionally fast forward, fast rewind, that are made available applying a graphical user interface.

5.2.2 Multicast Support

As mentioned in the video conference application example, an application protocol is necessary to allow for dynamic creating new session on top of the communication subsystem. Instead of having to re-implement each time such an application protocol each time a CSCW application is conceived, an additional application component is provided, located in the application component level. Namely this application support is offered from the Multicast Support Component . The purpose of this component is to provide a functionally extended API to the application programmer, while using well-known application components. This is performed through the Multicast Support Component which implements the desired application-level protocol. It becomes possible to create a "true" m : n multicasting session, actually composed of several session objects at the "traditional" API. To allow reuse of this Multicast Support Component for various CSCW applications, a clear separation between mechanism and policy has to be provided. The mechanism part describes how the application protocol actually can be supported between several machines control information is exchanged, whereas the policy part specifies which concrete actions must be taken upon receiving control information, *e.g.,* dynamic creation of a new receiving session after a new participant has joined a video conference application.

5.2.3 Application Sharing

The application component called application sharing provides a multi-user interface for cooperative and interactive communications. Especially, a dedicated single-user application can be used concurrently from many users at the same time. The application sharing component allows for collaboration transparency due to the fact that a simultaneous usage of a single application, which are remotely distributed between many workstations, is supported /Gute95/. Therefore, the use of the application sharing leads towards a support of interactive and cooperative applications that are needed for handling design issues, management decisions, teaching matters, or agreements that involve many users in different geographical locations.

5.3 Definition of Application Scenarios

An application scenario is composed of a single or multiple applications in addition to a meta-control interface and a corresponding graphical user interface. Obviously, a large number of scenarios can be defined, but a limited set of scenarios provides a useful functionality only. Therefore, the fol-

lowing plain scenarios have been identified, which directly arise from a stand-alone useage of applications. A World Wide Web scenario encompasses a traditional WWW browser and/or the features of an extended browser and shared browser respectively. Within this scenario the use of regular HTML documents as well as audio and video data is possible. Especially, the integration of multimedia data in the WWW environment offers various chances for up-to-date layouts for numerous information. Furthermore, sharing of a multimedia-capable WWW browser allows for the interactive, joint, and remote surfing through internet sites. Finally, a stand-alone video conference may offer various enhancements to the video conference application. This happens in terms of a parametrizable group management interface for dealing with dynamic joins and leaves of group members, and various synchronization functionalties that can be selected for different group members due to different destinations or connection requirements.

Additionally, two important application scenarios for real-world environments have been investigated. A tele-banking scenario /GuRü96/ offers a modern approach to support bank customer requirements in an efficient and cost decreasing manner. Basically, a customer uses an extended automatic teller machine on a self-serve basis, *e.g.*, for cash services, electronic fund transfers, investments in capital stock, or information retrieval of foreign currencies. For these reasons, the applications of secure messaging, video watch/send, audio listen/send, and extended WWW browsers are used. Due to unexpected difficulties or undecidable questions the customer may need an interactive support. Therefore, a bank´s clerk may enter the session and helps out with the current detected problems. Especially, the application sharing component allows for an interactive advice in addition to the picture phone.

The scenarios of tele-seminar and tele-report are quite similar. Both scenarios offer functionality to interactively teach a remotely distributed audience. Tele-seminar is based on a virtual classroom concept, where multiple participants watch and listen to one teacher or speaker. The participants may interact, like in a video, conference or may be restricted to talk one by one to the teacher only. Relevant applications included encompass the video conference application or a plain video watch/send and audio listen/send application to transfer video and audio data in addition to an optional messaging service. Tele-report is based on the idea to interconnect one speaker with a single remote audience, that has a whole a single access possibility to request questions. In this case video watch/send and audio listen/send applications are sufficient, since no video conferencing features are needed.

Finally, a number of additional scenarios may be defined using existing and future application elements. These scenarios encompass at least the following ones: tele-teaching, tele-work, tele-learning, tele-shopping, tele-marketing, electronic commerce, tele-libraries, or infotainment. Due to these emerging scenarios a set of to be identified applications (*e.g.*, electronic cash or smart money) and application components (*e.g.*, electronic fund transfer or data base queries) have to be added into the application framework. Furthermore, additional enhancements to the Da CaPo++ communication subsystem, especially in terms of protocol modules for protocol processing, *e.g.*, application coding or data base access, are relevant.

6. Summary and Conclusions

The three-level application framework comprising application scenarios, applications, and application components allows for the design and implementation of reusable building blocks that can be combined to define complex real-world application scenarios. Currently, the implementation architecture applies an object-oriented approach in close correspondence with the design architecture. Application components and applications are native objects simply described by their names and

interfaces. Creating new application scenarios, an elementary reuse of objects and code is especially important as there are still many unanswered questions concerning the user acceptance of multimedia applications in general. Many of these questions have to be investigated in future field trials. The design of a flexible and modular application framework based on top of an adjustable communication subsystem offers a broad range of experimental choices and defines building blocks within an application toolkit.

In addition, the design and implementation of an application programming interface offers the required degree of transparency between applications and the communication subsystem. While the complexity of the communication subsystem and of communication-relevant tasks is completely hidden from the application programmer, a useful exploration of the subsystem is still possible due to desired Quality-of-Service attribute specifications. Although breaking the transparency in the first place does not introduce disadvantages, but offers a magnitude better alternatives in providing from the communication subsystem´s point of view a best-suited communication protocol and service. Even in case of ignorant applications, communication facilities are provided by the communication subsystem relying on pre-defined standard communication protocols.

The Da CaPo++ communication subsystem itself accomodates a variety of applications due to its internal configuration capability for communication protocols and services. Multicasting and security, being part of the undertaken extensions compared to existing communication subsystems, offer necessary prerequisites for supporting real-world applications in an information distributing/retrieving and insecure environment of networks and workstations. The novel approach to this problem in Da CaPo++ integrates security into the existing framework of QoS attributes, thus reducing its originally isolated status. The properties of secure protocols are as changeable as those of an insecure protocol, provided that both parties agree to the change. Underlying secure infrastructures can be used as well. Key management using a web-of-trust like infrastructure, handles public keys of participants, and Da CaPo++ automatically tries to determine their authenticity.

Finally, the overall approach of integrating applications and supporting them by a modern communication subsystem is promising. As a number of future field trials with real-world scenarios are planned and exhaustive evaluations of application functionalities will follow, a control of user acceptance for the designed graphical user interfaces is important as well. Only if an extensive user approval of modern and emerging multimedia applications can be reached, the future of shared applications, tele-learning, and tele-working in the general public and productive environments may be successful. Da CaPo++ supplies basic tools and a suitable environment for investigating these issues in closer detail.

Acknowledgements:
The authours would like to express many thanks to their project partners, namely to J. Fominaya, T. Gutekunst, E. Rütsche, and M. Soland for discussing application and security requirements for the tele-banking and tele-seminar scenarios, the application framework, and its components.

7. References

/BeCo95/ T. Berners-Lee, D. Connolly: *Hypertext Markup Language-2.0*; RFC 1866, November 1995.

/BFFr96/ T. Berners-Lee, R. Fielding, H. Frystyk: *Hypertext Transfer Protocol – HTTP/1.0*; HTTP Working Group, Internet Draft, February 1996.

/BoFr93/ N. Borenstein, N. Freed: *MIME (Multipurpose Internet Mail Extensions): Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies*; RFC 1521, September 1993.

/CaDe92/ S. Casner, S. E. Deering: *First IETF Internet Audiocast*; ACM Computer Communication Review, Vol. 22, No. 3, July 1992, pages 92 – 97.

/CCHu94/ A. Campbell, G. Coulson, D. Hutchinson: *A Quality-of-Service Architecture*; ACM Computer Communications Review, Vol. 24, No. 2, April 1994, pp 6 – 27.

/Dant94/ A. Danthine: *The OSI´95 Transport Service with Multimedia Support – Research Reports ESPRIT, Project 5341; Volume No. 1*, Springer, Berlin, Germany, 1994.

/Deer91/ S. E. Deering: *Multicast Routing in a Datagram Internetwor*k; Ph.D. Thesis, Stanford University, California, U.S.A., December 1991.

/GoCo94/ B. Goodhart, J. Cox: *The Magic Garden Explained, The Internals of UNIX System V Release 4*; Prentice Hall, New York, U.S.A., 1994.

/GuRü96/ T. Gutekunst, E. Rütsche: *KWF-Projekt "Da CaPo++": Anwendungsszenarien*; Internal Project Report, SBV Basel/ETH Zürich/XMIT AG Zürich, Switzerland, January 1996.

/Gute95/ T. Gutekunst:*Shared Window Systems*; Ph.D. Thesis no.11120, ETH Zürich, TIK, Switzerland, 1994.

/IsTa93/ E. Isaacs, J. Tang: *What Video Can and Can't do for Collaboration: A Case Study*; ACM Multimedia, June 1993, pp 199 – 206.

/Moor93/ K. Moore: *MIME (Multipurpose Internet Mail Extensions) Part Two: Message Header Extensions for Non ASCII Text ;* RFC 1522, September 1993.

/PPVW93/ T. Plagemann, B. Plattner, M. Vogt, T. Walter: *A Model for Dynamic Configuration of Light-Weight Protocols*; IEEE 3rd Workshop on Future Trends of Distributed Systems, Taipeh, Taiwan, April 1992, pp 100 – 106.

/Purs93/ M. Purser: *Secure Data Networking*; Artech House, London, England, 1993.

/Schm92/ D. Schmidt: *IPC_SAP: An Object-oriented Interface to Interprocess Communication Services*; C++ Report, November/December 1992.

/Schn96/ B. Schneier: *Applied Cryptography*; 2nd edition, Wiley & Sons, New York, U.S.A., 1996.

/ScSu93/ D. Schmidt, T. Suda: *Transport System Architecture Services for High-Performance Communication Subsystems*; IEEE Journal on Selected Areas in Communications, Vol. 11, No. 4, May 1993, pp 489 – 506.

/Soo94/ J. Soo: *Live Multimedia over HTTP*; 2nd International World Wide Web Conference, Mosaic and the Web, Chicago, U.S.A., October 1994.

/Stil96/ B. Stiller: *Quality-of-Service – Dienstgüte in Hochleistungsnetzen*; International Thomson Publishing, Bonn, Germany, 1996

/VoKe83/ V. Voydock, S. Kent: *Security Mechanisms in High-Level Network Protocols*; ACM Computing Surveys, vol. 15, no. 2, June 1983, pp 135 – 171.

/VPPW93/ M. Vogt, T. Plagemann, B. Plattner, T. Walter: *A Run-time Environment for Da CaPo*; International Networking Conference Internet Society, San Francisco, U.S.A., August 1993, pp 100 – 110.

/WFWe96/ K. Wolf, K. Froitzheim, M. Weber: *Interactive Video and Remote Control via the World Wide Web*; European Workshop on Interactive Distributed Multimedia Systems and Services, IDMS'96, B. Butscher, E. Moeller, H. Pusch (Eds.), Berlin, Germany, March 1996, pp 91 – 104.

/ZSTa93/ M. Zitterbart, B. Stiller, A. Tantawy: *A Model for Flexible High-Performance Communication Subsystems*; IEEE Journal on Selected Areas in Communications, Vol. 11, No. 4, May 1993, pp 507 – 518.