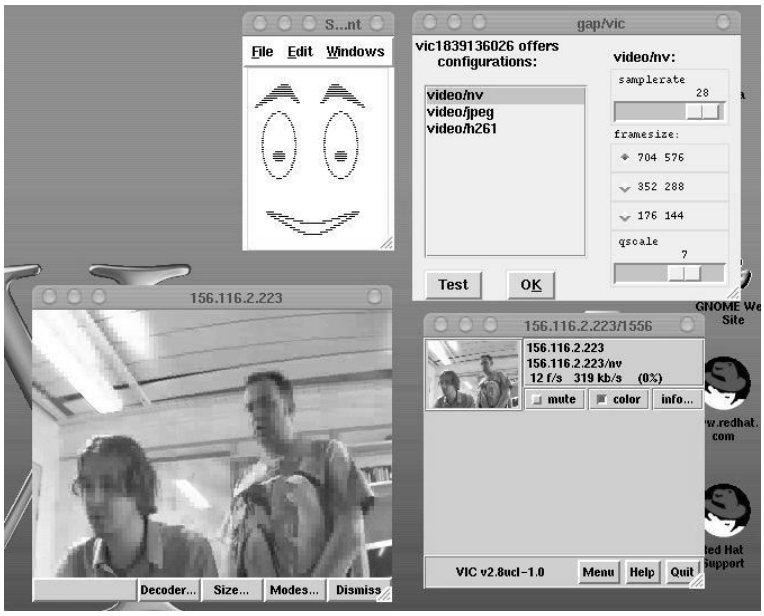
Trading of QoS policies in ENNCE

Norsk Regnesentral
ANVENDT DATAFORSKNING
Norwegian Computing Center/Applied Research and Development

NOTAT/NOTE



DART/03/02

Wolfgang Leister
Gorm Paulsen
Pål Spilling

Oslo
December 2002

Norsk Regnesentral
ANVENDT DATAFORSKNING

Notat / Notat

Tittel/Title:
Trading of QoS policies in ENNCE

Dato/Date: December
År/Year: 2002
Notat nr/:
Note no: DART/03/02

Forfatter/Author:
Wolfgang Leister, Gorm Paulsen, Pål Spilling

Sammendrag/Abstract:

This contribution is the concluding report of the ENNCE project WP1, which elaborates the trading mechanisms that were studied and implemented during the course of the project.

The main goal is to elaborate trading mechanisms for QoS negotiation between the involved parties in a data network. Especially we take into account that the QoS negotiation is done on behalf of an end user. For the negotiation we propose an agent-based system based on code passing between the entities involved. Selected parts of the implemented systems are presented.

Emneord/Keywords: Quality of Service, QoS negotiation, Service Agent, Trading

Målgruppe/Target group: research institutions, NR

Tilgjengelighet/Availability: Open

Prosjektdata/Project data: ENNCE

Prosjektnr/Project no: 801001

Antall sider/No of pages: 40

Norsk Regnesentral/Norwegian Computing Center
Gaustadalléen 23, Postboks 114, N-0314 Oslo, Norway
Telefon (+47) 22 85 25 00, telefax (+47) 22 69 76 60

Contents

Trading of QoS policies in ENNCE

Wolfgang Leister
Norwegian Computing Center
wolfgang.leister@nr.no

Gorm Paulsen
Institute for Informatics
University of Oslo
gormp@ifi.uio.no

Pål Spilling
Center for Technology at Kjeller
paal@unik.no

This contribution is the concluding report of the ENNCE project WP1, which elaborates the trading mechanisms that were studied and implemented during the course of the project. The main goal is to elaborate trading mechanisms for QoS negotiation between the involved parties in a data network. Especially we take into account that the QoS negotiation is done on behalf of an end user. For the negotiation we propose an agent-based system based on code passing between the entities involved. Selected parts of the implemented systems are presented.

Keywords
Quality of service and media scaling, Multimedia-specific intelligent agents, Resource management, trading protocols

1. The ENNCE project

This document is an elaboration of the trading mechanisms in the ENNCE architecture, as part of QoS management in continuous multimedia systems. The ENNCE project is funded by the Research Council of Norway, and consists of two work packages. WP 1 comprises an overall architecture of QoS-management, including the selection of suitable applications and to build up an infrastructure (hardware) for experimentation. The main element of the architecture of WP 1 includes the use of a Service Agent (SA), which will be elaborated further in this document. The progress of this work package is presented in [1, 12, 13, 14].

WP 2 focuses more on the middleware, and underlying logic of the negotiation process. The main elements comprise the reference model for Open Distributed Processing (RM-ODP) [15], and the Model of Binding and Streams (MBS) [2].

This document shows how the two work packages of the ENNCE project melt together in a more general, unified framework. The glue between the two parts can be seen in the trading mechanisms for QoS parameters.

Trading mechanisms for QoS parameters are especially needed for continuous media, and situations where mobility, resource awareness, varying network connections, and client capabilities are involved. Several authors have proposed frameworks; see for instance [16] for the use of an agent technology in mobile environments. The framework of the ENNCE project is meant to be a general tool box to build QoS negotiation support for many of the use cases mentioned above.

In the remainder of this section we present an overview of the work done in the ENNCE project prior to the current report. We refer to the architecture document [1], the thesis of Terje Michelsen [8], the thesis of Gorm Paulsen [17], the characterisation of the applications [12], and implementation of the network infrastructure [13, 14].

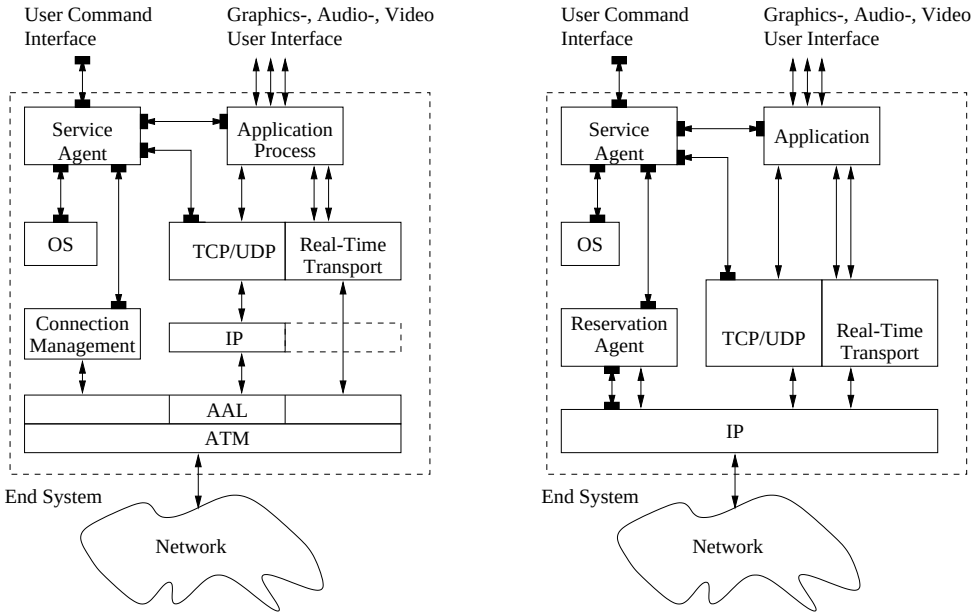


Figure 1: End System Architecture for ATM (left) and IP (right)

1.1. Architecture document

The architecture document [1] elaborates the general framework which the ENNCE project is based upon. Much multimedia research has been devoted to studying networking and middleware aspects of QoS, and how QoS should be managed, without including the user. The novel approach taken in ENNCE, is to place the overall QoS control in the hands of a single entity in the requesting client system, called the Service Agent (SA), see Figure 1.

The basic underlying principle is that the higher the quality of a multimedia application, the more resources it consumes in the network and end systems; hence the requesting user should be charged accordingly. From the user point of view, the negotiation process will involve the chosen application, the user's desired quality, and the user's willingness to pay for that quality. From the network point of view, the negotiation process will involve the application's required quality and the service offered by the network. It is assumed that a random user has no system and multimedia knowledge, and therefore needs to be guided through the complicated negotiation process.

This SA entity (not to be associated with agent technology) can interact with all system components, including the user. Through these interactions, it learns the capabilities of the end system and the application process, and can negotiate QoS attributes and configure the end systems and the network accordingly.

The Service Agent will build up standard application-specific configuration profiles, which may also be specific for each user, and use that as a starting point in negotiating with the user. This means that the user may modify the configuration profile as need be.

An essential functionality is the interfaces between the SA and the various components of the end system. These interfaces specify the services used by the SA and the services offered by the SA. Called SA Control Protocol (SACP), a dedicated control message protocol has been developed for this purpose. It serves the following purposes:

- passes requests and status information between the user and SA,

- passes status and configuration information between SA and end system components,
- supports negotiation with the network,
- provides a syntax for profile information,
- supports negotiation with the remote end system.

1.2. Terje Michelsen's thesis

Distributed communicating application processes are designed and implemented under the assumption that data are exchanged between them with a given minimum service quality. This minimum service quality will vary from application to application. Typical traditional Internet applications, like file transfer and email, require high reliability, but very little regarding delay and delay variations. In contrast, applications involving voice and video streaming across the network, have tighter requirements regarding delay and delay variations and less stringent ones regarding reliability. These requirements will vary depending on the degree of interactivity, simplex, half duplex, and duplex, and the voice and video coding methods.

In multimedia applications there will be a mixture of data, graphics, and streams of voice and video. The streams require higher controllable quality than the data, and will depend on the quality of the voice and video presented to the end users. Quite obviously, higher quality requirements are more costly to fulfil. Hence the end users have to be charged for the extra costs. And quite naturally, a user wants the highest possible quality for the lowest possible costs. In utilising multimedia applications, the user should therefore be consulted regarding what quality he/she is willing to pay for. This means that the end user has to be incorporated in the negotiation process being initiated when a distributed multimedia application is fired up.

Having no specialised system and multimedia knowledge, the random user needs to be guided through part of the negotiation process with the help of the Service Agent. This has been the focal point of Michelsen's thesis. It involves the following elements:

- a user-to-SA interface; the control interface,
- a voice and/or video interface; user to application interface,
- a quality – cost model.

The exact relationship between cost and quality has not been essential in this thesis, but this will be a key element in real applications.

Since the user is assumed to have no technical knowledge, it is essential that the user can specify subjective quality requirements. These requirements are then mapped into concrete attributes representing the requested quality of service and suitable for negotiation process and configuration of the end systems and the network.

With reference to Figure 1, the main tasks of the Service Agent are the following:

- discovery of the QoS-dependent capabilities of the application, and the middleware and operating system at the client side,
- support the user in the QoS-specification,
- configuring of the end system according to the agreed upon QoS,
- establish the communication channel between the communicating end systems,
- and finally negotiate the QoS requirements with the Service Agent in the remote end system, and as a result have the remote end system configured accordingly.

In negotiating QoS, we envision the following steps:

Step 1; The multimedia application process is fired up and contacts the Service Agent (SA)

Step 2; SA requests the application to specify the supported media types and their coding capabilities. SA may also probe other end system components to discover their ability to provide QoS support. SA also needs to know the cost model.

Step 3; SA, knowing the capabilities of the application, the end system, and the cost model, starts negotiation with the user. SA maps the user's subjective requirements into concrete QoS parameters.

Step 4; SA interacts with the Reservation Agent, see Figure 1, and requests it to establish a path across the network to the specified destination, making the necessary resource reservations along the path according to the required quality. The request will be accepted, or refused because the needed resources are not available. In this example we have assumed the IntServ model. Step 4 can be adapted to the DiffServ model and to the use of MPLS.

Step 5; having established the path to the remote end system, the final step for the local SA is to contact the remote SA, hand over the QoS requirements in order to have the remote end system configured accordingly. If this is not possible, the session is terminated. Otherwise the distributed application processes start the real communication.

The Pricing Model

Networks and end systems have finite capacities. It is obvious that the use of distributed applications requiring controllable performance, both of the network and of the end systems, must be accompanied by a pricing model – higher required performance will be charged more, while lower required performance will be charged less. A properly designed pricing model is therefore needed as a means to regulate the resource consumption.

For simplicity we assume a user requesting a "Video on Demand" application from a remote server. We then have two alternatives regarding the pricing model:

- pricing depends only on the resources consumed in the network and at the server side,
- pricing incorporates also the load on the server side.

The user is interested to optimise the quality perceived for the lowest possible price. From empirical studies the perceived quality M can be expressed as:

$$M = \frac{\alpha \cdot US}{\beta \cdot RC}$$

where US is user satisfaction, RC is resource consumption and α and β are weighing factors. The ratio α/β may be static or made dynamic and dependent on the total load (network and end system). In an interactive distributed multimedia application, US will depend on many factors, like jitter, end-to-end delay, price, skew between picture and voice, picture rate, and packet loss. In a dominant one-way application, end-to-end delay and jitter will be less important. We assume that packet loss is not random and due to transmission errors, but will be load dependent in the routers and end system. If we utilize MPEG coding of the video information and configure the routers and end systems such that packet loss is made selective and load dependent, we can gradually reduce the required network capacity with a gradual decrease in picture quality. The following strategy in reducing the required bandwidth has been followed, in order of increasing importance for the picture quality:

- increase the quantization level of the MPEG coding,
- reduce the picture rate,
- reduce the picture resolution,
- increase the drop rate of B-frames.

User profiles

The Service Agent (SA) maintains different aspects of user profiles, called profile vectors:

- application vector; subjective application requirements are mapped into an application attribute vector. The user participates in negotiating the contents of this vector, so that it is consistent with the available resources (the system vector).
- system vector; contains information on available system resources, like cpu- capacity, bandwidth, buffer capacity, input/output capacity and current resource utilization.

If the negotiated requirements cannot be met, the attempt to utilize the application is refused.

We envision the following processing steps from when the user requests a given application until it is operational:

1. the user notifies SA about its intention, and SA responds with requesting the application requirements from the user.

2. the user specifies the desired quality requirements and possibly a price level. The specification can have the form of subjective user requirements, an application vector, or just a price limit.

3. in case the user specifies subjective requirements, SA is converting these into an initial application vector. The application vector is application specific, because it depends on the media coding formats. Based on the initial application vector, SA is then calculating a traffic specification. In case the user only specifies a price limit, SA may adjust the traffic description to conform to the user's price limit.

4. the system vector is now adjusted to reflect the application requirements regarding system resources and traffic requirements, and that there are sufficient resources available. We assume at this stage that SA knows the address and capabilities of the application server, and that the traffic description is not in contradiction with the server's capability.

5. the resource requirements and the resource availability are used as input in a cost calculation, presented to the user for acceptance.

6. if accepted by the user, the application profile – including the traffic description – is stored for later use.

7. the multimedia session with the server can now be established.

The Service Agent SA

The available funding and human resources did not permit a complete implementation of SA. We therefore limit our research to investigating the interface and the negotiating process involving the user. Further we limited our application to "Video-on-Demand" with MPEG video coding format, and simulated a network environment with adjustable bandwidth and

loss rate. This enabled us to investigate the perceived quality as function of bandwidth, loss rate and cost. The cost C is a function of resource consumption and current load on the network, and consists of a linear part and an exponential part:

$$C = X + X^2$$

where $X = RC/RF$ with $RF = 1/(4 \cdot NW)$, NW denoting the network load, and RC the resource consumption.

1.3. Discussions and conclusions

- previous work has mainly focused on end-system to network QoS negotiation, including distributed multimedia applications, but omitting the user,
- a main point in current contribution has been to include a cost model in SA- user quality-of-service negotiation, with focus on user satisfaction. User satisfaction is optimised as function of price and quality (resource consumption),
- the used price model includes a fixed part and an exponential part, both dependent on the required network resources and network load,
- the implementation has been concentrated on testing out the SA-user interface, the relevant part of SA, and a MPEG-based configurable application. The network and the remote end were just simulated, in order to concentrate on the user involvement in the negotiation process,
- the user could only specify objective quality requirements, while the initial intention was to permit both subjective and objective specifications. Funding and time constraints did not permit this.
- the negotiation concept should be tested out in a real situation involving configurable end systems and network,
- the cost model needs improvements to incorporate more dependencies, for a better optimising price and performance
- as a result of the current investigation, our pricing model seems to be an effective means to regulate resource consumptions in multimedia applications.

2. Service Agent

The architecture and the negotiation process is presented in the architecture document [1]. The architecture uses the Service Agent as a main element. The SA negotiates on behalf of the user and the involved applications and controls the setup of the QoS parameters. The trading process for QoS parameters is the main concern of this report. In ENNCE the trading process uses the stream binding paradigm, and a high level Service Agent Control Protocol (SACP) as the main ingredients. SACP is a part of an agent technology, where both data, trading protocols and procedures are exchanged between the parties involved, and their respective agents.

The SA performs a variety of tasks:

- Discern the QoS capabilities of the application media, the middleware, and the network: The SA can establish connections with agents and demon programs on the local or remote hosts, and, where appropriate, retrieve information using the SACP, or other convenient protocols. Additionally, the SA can retrieve information from files, or make use of default or guessed values.

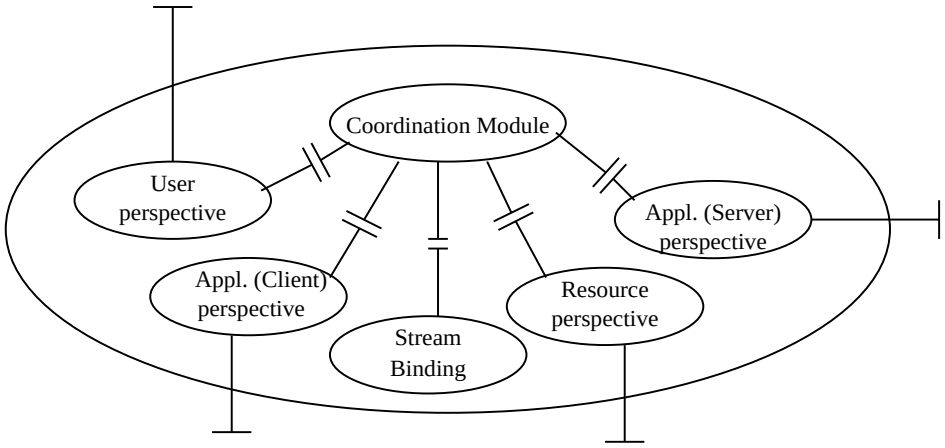


Figure 2: The perspectives and modules within the Service Agent.

- Support the user in negotiating QoS requirements: The SA can open SACP connections with user agents, that offer user interfaces for configuring QoS parameters at user level. Additionally, user profiles and preferences can be built up, and later utilised by the SA in the configuration process.
- Configure the end system according to the negotiated QoS requirements: The SA can configure the end systems by supplying the appropriate parameters in command line mode, or by forwarding the appropriate parameters via a connection using a suitable protocol.
- Establish the network connection with the right QoS properties: The SA negotiates usually by means of a reservation agent (e.g. `rsvpd`) via a suitable protocol or API.
- Negotiate QoS requirements with the other end system: Several SA can be involved in the negotiation part. Stream binding and the SACP protocol are used in order to negotiate among the parties (user, client, server).

The SACP protocol includes:

- A session protocol which permits the exchange of asynchronous (text-based) messages. The session protocol can be used to transport text-based data in whatever syntax, and programs.
- The negotiations are facilitated by the use of a high-level language. We chose the language PF, which implements a code passing mechanism as a means to exchange more than just values. PF is a general purpose high level language, related to PostScript [18]. The interpreter for and its run time system is implemented in C. See Section 3.2 for a more detailed presentation of PF..

2.1. Perspectives

The ENNCE architecture consists of several loosely coupled components, in order to break the complexity of the system into manageable parts. Each component deals with its own distinct, well-defined area of concern, called perspective. The perspectives and the modules of the Service Agents relate to each other, which is illustrated in Figure 2.

The media perspective. Application level QoS parameters are media specific, and dependent on the media format, which makes the translation and mapping between the various media difficult. Therefore, a component is required that handles the mapping of media specific application level QoS parameters into parameters at the system level.

The media perspective can be implemented through a Media Knowledge Base, which is an agent dealing with media specific information, such as specifying what application level QoS parameters are required for a given media format, and providing appropriate functions for mapping between QoS parameters at the application and system levels.

The resource perspective. Resource management and mapping of system level QoS parameters into resources is done by the Reservation Agent. This entity operates at the system level, and interfaces with the OS and resource management protocols, e.g. RSVP or SNMP (see RFC 1098). Its responsibilities include resource reservation, admission control and monitoring.

The application perspective. Applications have knowledge about what types of media formats they support, and to what extent the various parameters of each media format are adjustable. In addition, application developers are faced with the challenge of having to implement a negotiation procedure and a user interface for negotiating user level QoS parameters for every application. We believe that there is a need for external support in managing these tasks. If such support were present, it would allow applications to state their capabilities in terms of media formats and application level QoS parameters in order to take advantage of negotiation facilities and reservation mechanisms in the architecture. We use an API that allows applications to state their capabilities in terms of media support and parameter adjustments.

The user perspective. The end users of the system can specify their own preferences in terms of media formats and user level QoS parameters. Assistance is offered for the negotiation situation. Users are not a homogeneous group, and we can expect to find users ranging from levels of very little technical insight to highly skilled expert users. Thus, there is a need for user-supporting entities that can take into account the varying levels of user expertise and help select appropriate formats and parameter values.

In our implementation these issues are handled with by the User Agent component. The User Agent's responsibilities include assisting the user selecting appropriate media configurations at negotiation time, and to provide the system with the user's adaptation policies. The agent may use historic data (previous experiences) in guiding the user.

The overall perspective. The Service Agent component plays the role of a coordinator in the architecture, controlling the other components. It has several responsibilities, including:

- Capturing the application's capabilities
- Performing comparability checking
- Receive events from the Reservation Agent
- Interface with the User Agent
- Receive and send messages to the applications during sessions
- Communicate with Service Agents on remote hosts

Any communication between hosts is handled by the Service Agents, such as contacting a host for session initiation. The system is designed in such a way that applications can initiate communication using any session initiation protocol available.

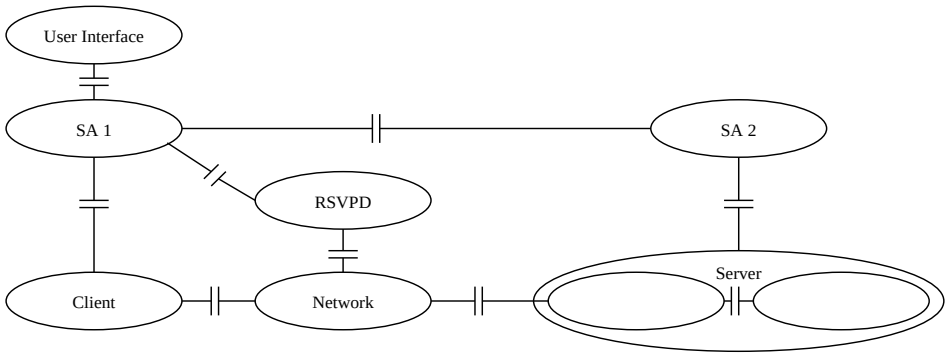


Figure 3: This Figure shows an example of stream binding with SA:The objects *Client*, *Network* and *Server* are controlled by two Service Agents, SA1 and SA2. The network object is controlled by `rsvpd`. The service agents have also interfaces in-between.

2.2. Negotiation of Stream Bindings

The ENNCE model uses the MBS (Model of Bindings and Streams) as outlined by Rafaelsen and Eliassen [2]. In this model a stream interface consists of a collection of a source and/or sink media flows. During the stream binding process, a logical association is established between compatible stream interfaces. The result of the binding action is a control interface through which the binding object can be controlled. In the ENNCE model the Service Agent controls this interface, which includes adding new interfaces, and removing existing ones. The Service Agent also implements or has access to the binding factory (it contains the negotiation protocol). An example on how the binding model for a client-server application could look like is illustrated in Figure 3.

The ENNCE model follows the principles of the MBS, including the rules of conformance. However, the negotiation procedure by Rafaelsen and Eliassen [2] is somewhat restricted, as it implements a fixed algorithm that tries all combinations in a specified order. In contrary to that, the ENNCE model does not define a negotiation strategy. The code-passing paradigm makes it possible to transfer the negotiation strategy from one participant to another, according to the user's profile.

Work within that area includes the feature set model by Klyne in RFC 2533 [19], and Flow type model and the language FIDL (Flow Interface Definition Language) by Mehus [20], which has its roots in the MBS model by Eliassen and Nicol [7].

Klyne presents a syntax to describe media feature sets IN RFC 2533 [19], that are expressed by a combinations and relations of individual media characteristics. Additionally an algorithm to match feature sets is included.

2.3. QoS negotiation

Negotiation is the process of coming to an agreement on parameters or determine the quality of a data transmission. Negotiation between a sender and a receiver usually consists of a series of negotiation meta-data exchanges that proceeds until either party determines specific data to be transmitted. This process implies an open-ended exchange of information between sender and receiver. Another issue is service discovery.

QoS negotiation in general has been available for special purposes. Examples include the fax negotiation protocol or the negotiation protocol for modems. For fax negotiation the

recipient declares its capabilities, and the sender chooses a message variant to match.

The negotiation of QoS and media content is not yet generally available for standard applications. However, there are attempts to implement QoS negotiation for media, like the Transparent Content Negotiation (TCN) for the HTTP protocol [22]. TCN is experimental with the following strategy:

- The recipient requests a resource with no variants, in which case the sender simply sends what is available.
- A variant resource is requested, in which case the server replies with a list of available variants, and the client chooses one variant from those offered.
- The recipient requests a variant resource, and also provides negotiation meta-data (in the form 'Accept' headers) which allows the server to make a choice on the client's behalf.

According to Danthine and Bonaventure [23], three actors are involved in a (peer-to-peer) negotiation: *calling user*, *called service*, and *service provider*. This is a likely situation for the ENNCE project. Note, that the ENNCE framework still can handle other situations as well. Danthine and Bonaventure propose a 4-primitive information exchange: *request – indication – response – confirmation*.

In the same paper, the authors discuss several types of QoS negotiations, e.g., **Triangular negotiation for information exchange**, *Triangular negotiation for a bounded target*, *Triangular negotiation for a contractual value*, *Bilateral Negotiation*, and *Unilateral Negotiation*. The ENNCE framework is not bound to one of these negotiation types; either of them can be implemented in SACP.

The ENNCE framework allows to specify different kinds of negotiation protocols. Data (i.e. characteristics) and procedures (i.e. policies) can be exchanged between agents due to the code-passing capability. The SACP is used for negotiation, which includes a session protocol, and the use of the interpreted programming language PF.

In a simple case the service agents exchange the characteristics of alternative stream flows with different qualities determined by a given protocol/policy. Then one agent calculates a list of compatible stream flows, which then are attempted set up (possibly including user interaction).[1]

The specific advantage of the ENNCE framework is that the protocols, policies, and formulae for calculating derived characteristics can be exchanged between agents. The information is then originated from a place where it is known best, even if not all information is available there. The information can also be updated easily, and user profiles can be stored directly. This gives much flexibility.

In order to respect the specifications of the end user, her profile is used in the negotiation process. The end user is the consumer of the stream, and will eventually also pay for the QoS in the stream delivery. More on accounting, charging, billing and pricing of network services in connection with QoS can be found elsewhere [24, 25]. In the negotiation process an appraisal function is defined, which is a function of QoS parameters, the user's profile, and application requirements. The appraisal function can be used as a measure on the quality of the stream.

The negotiation procedures of the stream binding process outlined in [20, 2] are somewhat too rigid. Smaller deviations, or the utilisation of adaptation filters, cannot be handled by these procedures. This may result in that the negotiation/binding attempt fails, even though slightly worse alternatives exist, that are not covered by the user's specifications.

For each stream characteristic (or attribute) we can define an appraisal function, that is defined as a sum of weighted costs of the appraisal of QoS capabilities. This function gives a

[1] This procedure is quite similar to the scenario in [2].

value to the quality of a stream. By comparing two appraisals we can calculate the difference or distance between two streams.

In ENNCE the media model consists of the following entities:

media description: frame work for description of qualities / characteristics of media at the end points.

compatibility: a method for describing the degree of interoperability between end points must be developed, so that the common media characteristics at the end points can be used.

appraisal function: information that ties the characteristics of a media format to QoS, both on user-, network- and system level.

The appraisal function and the service agent concept can be used in connection to all entities in a network, including routers. Each router will in the ideal case forward the stream with the required quality, or may reduce the quality for various reasons. We define the appraisal function to compare the users specifications with the quality description of the forwarded stream at node n. When a stream passes through several nodes the stream description is updated to represent the actual value from the sender to this node, and a new appraisal is calculated.

The following method can be applied in this case:

1. Since the user most likely will pay the bill for the delivery of the multimedia data, an appraisal function is attached to all streams that are passing any entity.
2. In order to compare a stream definition with the user's specification, we use a function that describes the distance between the two streams. This function is a measure for the compatibility between these two streams, where low values represent a good match, and high values a bad match. This function is called "badness".
3. We add the (weighted) costs with the (weighted) badness to get a value, that expresses the costs of setting up the stream. This function is called "appraisal function".
4. We create an appraisal functions for all nodes the stream passes through.
5. Values that are not known are assumed, possibly using statistical values (from the past) or data from simulations (e.g. for the network behaviour).
6. When this value later shows to be wrong, e.g. a connection cannot be reserved, we set this value later to a high badness.
7. When the outcome of the appraisal function is larger than a user-defined limit, the stream is not set up, and the user will be notified about that fact.
8. The appraisal function (which expresses the quality of a stream) can be expressed as being dependent on other variables.

2.4. Reference model for negotiation

RFC 2703 [21] contains a framework for setting up bindings for multimedia content. The document is related to RFC 2533 [19], and covers content negotiation for application resources, that goes beyond the handling of MIME headers. The overall framework is shown in Figure 4.

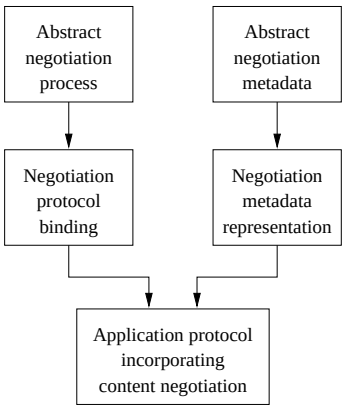


Figure 4: Negotiation framework, from RFC 2703

The goal of RFC 2703 is to provide a protocol-independent content negotiation framework, including the identification of some technical issues. Not all terms might be applicable to all multimedia purposes in general.

The other parts in RFC 2703 outline a framework for describing protocol-independent content negotiation, including some general goals and technical issues. The goals mentioned in the document can be seen as a general checklist that is useful for designers of negotiation protocols. In general, content negotiation covers the following three elements:

1. expressing the capabilities of the sender and the data resource, to be transmitted;
2. expressing the capabilities of a receiver;
3. a protocol by which capabilities are exchanged.

Comments on goals of RFC 2703

The framework document RFC 2703 lists goals that a negotiation framework should meet. We comment on these goals, and compare these with our work. Some parts of RFC 2703 cannot be applied to ENNCE, since ENNCE uses a code-passing paradigm. See Section 3 for more details on the underlying mechanisms.

Goals in RFC 2703	How ENNCE meets these goals
General deployment goals	
A common vocabulary for designating features and feature sets.	The code passing paradigm, with the possibility of PF to define objects and procedures, even enables the partners of the negotiation to exchange a common vocabulary.
A stable reference for commonly used features.	The definition of SACP, and the appraisal function build such a reference.
An extensible framework, to allow rapid and easy adoption of new features.	The code passing paradigm, with the possibility of PF to define objects and procedures, enables the partners of the negotiation to exchange a common vocabulary.

Goals in RFC 2703	How ENNCE meets these goals
Permit an indication of quality or preference.	The appraisal function is such an indication.
Capture dependencies between feature values	As PF is a programming language, dependencies between feature values can be expressed
A uniform framework mechanism for exchanging negotiation meta-data should be defined that can encompass existing negotiable features and is extensible to future (unanticipated) features.	SACP and PF implement this.
Efficient negotiation should be possible in both receiver initiated ('pull') and sender initiated ('push') message transfers.	The use of the Service Agent permits both pull and push operation in the negotiation.
The structure of the negotiation procedure framework should stand independently of any particular message transfer protocol.	Even though the ENNCE framework has several hooks for other protocols, the negotiation is based on SACP and PF.
Be capable of addressing the role of content negotiation in fulfilling the communication needs of less able computer users.	The Service Agent can communicate with all subsystems, and retrieve information on the capabilities of a computer. This can be included in the formulas, that are evaluated by the PF interpreter.
Protocol-specific deployment goals	
A negotiation should generally result in identification of a mutually acceptable form of message data to be transferred.	yes.
If capabilities are being sent at times other than the time of message transmission, then they should include sufficient information to allow them to be verified and authenticated.	The SACP protocol has several mechanisms to provide this. Connections are identified by names, and the negotiation objects can be accessed by names in the PF name space.
A capability assertion should clearly identify the party to whom the capabilities apply, the party to whom they are being sent, and some indication of their date/time or range of validity. To be secure, capability assertions should be protected against interception and substitution of valid data by invalid data.	An indication of date/time has not been included yet; however this should be straight forward. The other parts are included in SACP. This does imply the use of security features like authentication, integrity protection, etc.

Goals in RFC 2703	How ENNCE meets these goals
A request for capability information, if sent other than in response to delivery of a message, should clearly identify the requester, the party whose capabilities are being requested, and the time of the request. It should include sufficient information to allow the request to be authenticated.	With the exception of time, these parts are provided by SACP.
The negotiation mechanism should include a standardised method for associating features with resource variants.	Provided by PF, which is a programming language that allows the definition of structured data.
Negotiation should provide a way to indicate provider and recipient preferences for specific features.	Yes. Profiles for user, etc. can be stored, retrieved, and set by standard mechanisms.
Negotiation should have the minimum possible impact on network resource consumption, particularly in terms of bandwidth and number of protocol round-trips required.	The code passing paradigm makes it possible to retrieve a recipe. Therefore, instead of communicating with clients for every request, the client can calculate values of known parameters and functions. Decisions can be taken locally in the service agent without too many round trips necessary. Round trips are only necessary to retrieve actual data.
Systems should protect the privacy of users' profiles and providers' inventories of variants.	This feature can be implemented; however it is not used in the current prototype.
Protocol specifications should identify and permit mechanisms to verify the reasonable accuracy of any capability data provided.	Possible to define, but not implemented in the current prototype.
Negotiation must not significantly jeopardise the overall operation or integrity of any system in the face of erroneous capability data, whether accidentally or maliciously provided.	While the ENNCE prototype is not secured against attacks, this does not affect the concept as such. The main negotiation functionality is provided by a separate application, the Service Agent. Should the Service Agent be non-functional, the applications could continue to work as without negotiation.

Goals in RFC 2703	How ENNCE meets these goals
Intelligent gateways, proxies, or caches should be allowed to participate in the negotiation.	Interfaces and agents representing the gateways, proxies, etc. can be added.
Negotiation meta-data should be regarded as cacheable, and explicit cache control mechanisms provided to forestall the introduction of ad-hoc cache-busting techniques.	SACP/PF provides this feature.
Automatic negotiation should not pre-empt a user's ability to choose a document format from those available.	The user's profile can contain this.

3. Negotiation with the Service Agent

3.1. Implementation of the Perspectives

The implementation of the Service Agent follows the model of perspectives described previously. Whether these perspectives are separate modules, or implemented as PF-code is dependent on the implementation.

The architecture of the prototype system implemented by Paulsen [17] has the following components:

- service agent (each host)
- user agent (for each user)
- media knowledge base (connected to SA)
- reservation agents (connected to SA)
- applications (with or without user).

Steps in a Session

In experiments with the ENNCE concept, the following negotiation steps were chosen as an example. Note, other procedures and negotiation protocols might be suitable as well, and these will probably be supported by the framework. The steps of a session are as follows:

- set up connections between service agents.
- information exchange (specifications from the applications to the service agents)
- compatibility checking
- resource checking
- negotiation with the user(s). (what do users want)
- admission checking (can system support the user's desire?)
- set up the streams.
- adaptation
- renegotiation (if necessary)

The functionality should make sure that the adaptation phase is controlled by the users' adaptation functions. These are generated by the user agent in the negotiation phase with the user. Therefore, it exists on a per-session basis, and therefore we have achieved a dynamic

user-controlled binding. Technically, the adaptation function is a PF-function, that is sent to the SA by the user's agent by using the code-passing paradigm.

The steps involved when setting up a session are as follows:

- An application initiates a session by making a call to its local Service Agent. At the same time it provides its SA with its capabilities.
- The involved Service Agent sets up a communication channel, exchanging capability information on behalf of the application and performing compatibility check. An intersection of the capabilities of the applications is computed using media information from the Media Knowledge Base. Intersected compatibility information then propagates back to the respective agents.
- The Service Agents, now having sufficient compatibility information, contact the user's User Agents (if any), initiating user negotiation.
- The User Agents retrieve the user's profile.
- When no profiles exist, the User Agents execute some negotiation routine, making use of the compatibility information and media information provided to them by the Service Agent. This process may involve the displaying of example media clips to illustrate various levels of QoS to the user. The required output of this negotiation is a selected media configuration and an adaptation policy. This policy, in the architecture termed "Adaptation Function" governs the user's adaptation to changes in transmission conditions.
- Having obtained the user's selected configurations and adaptation strategies, the agents perform admission testing, to check whether sufficient resources are available. If this is the case, the applications are told to set up the media streams. Otherwise, renegotiation is initiated.
- During transmission, Service Agents react to events received from the Reservation Agent, and take appropriate actions based on the resource information provided. An example of such an action is to invoke the user's adaptation function in response to a resource degradation message from the Reservation Agent. The adaptation function may perform operations on the sending application's configuration, e.g. lowering the sample rate, or initiate renegotiation.

Principles of the Service Agent

The service agent has

- a control interface for the user (possibly)
- control interfaces between agents
- control interfaces to application

Code passing

- Connection need not be available at all times. An agent can make assumptions when the corresponding peer agent is unavailable.
- Send policies directly instead of data. Advantage when data change, the changes can be adjusted locally (no network traffic).
- constraints and parameters are all passed by a standard mechanism.
- Practical issues: text substitution ...

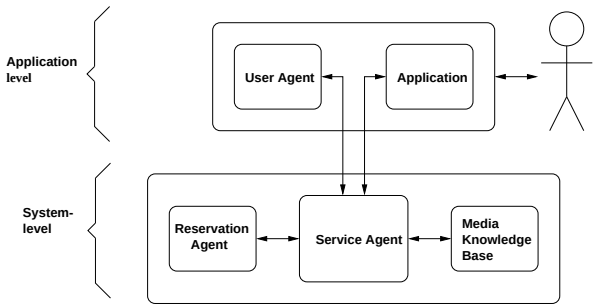


Figure 5: Main structure in the SACP architecture

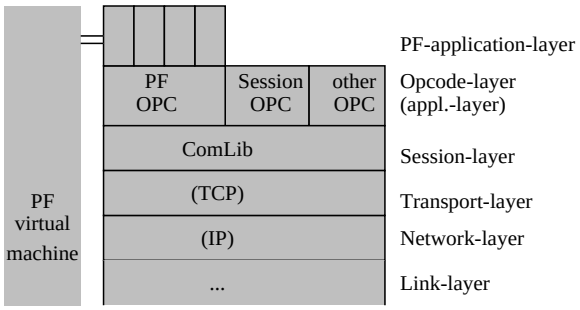


Figure 6: Protocol-layers in SACP (over TCP/IP)

3.2. SACP — Service Agent Control Protocol

The Service Agent Control Protocol (SACP) is used for negotiation between several Service Agents, and between the Service Agents and other parts of the system (as the User Interface Agent, or the applications). Therefore, the SACP has to meet the following characteristics:

- flexibility and simplicity.
- encapsulation of several protocols.
- possibility to express complex rules, and facts.
- robustness.

The SACP consists of several layers in the protocol stack, which are outlined in Figure 6. The SACP protocol is implemented on top of a connection-oriented transport protocol (e.g., TCP).

The session layer has an API that does not have an explicit *open* or *close* call. Functions like *SendToApplication* or incoming connections perform these calls implicitly. Unsent messages are queued, and sent when the connections eventually are set up (again). Even when a connection breaks down (on a lower protocol-layer), the session is still kept. A session is identified by a channel name, which can be resolved to port number and IP address by lookup in an internal database (also called portmapper).

The SACP exchanges messages are marked with an opcode. With this mechanism, SACP can encapsulate other message-oriented protocols. The application can set up the interpreters for the different opcodes. See more in the Appendix.

In SACP the PF interpreter is attached to the application. Other languages than PF (e.g. LISP or Prolog) could be used, when a suitable interpreter is available. PF-code can

be included in messages with application-defined opcodes. This PF-code is processed in an environment that is defined by the opcode.

The SACP protocol needs more than just a mechanism to exchange data and facts on the basis of a simple client-server based protocol. Especially for exchanging policies and procedures (between agents) the exchange of code must be possible. The characteristics of a language that is suitable for our purposes must include the following:

- Possibility for the use of variables, definition of procedures
- Execution of code based on an interpreter (or just-in-time compiler).
- Simple syntax, including the possibility to use a simple subset of the language for messages to applications that can parse the messages with a simple parser.
- Possibility to create run-time environments, where fragmentary code can be executed.
- Availability of interpreter and run-time system, that can be attached to the agents and (possibly) applications. Possibility to integrate the interpreter and run-time system with the message-system.
- Powerful language with run-time constructs, error recovery possibilities, and data structures.
- Human-readable form would be advantageous.
- Text based language makes substitution operations possible in external programs, e.g. using a Perl-script as a client.
- An extensible language is preferable.

Though built for code that is portable and usable for agents, we consider Java not to be suitable for our purposes. Candidates for a suitable programming language would be numerous, including various flavours of BASIC, standard programming languages based on an interpreter, Prolog, LISP, or PostScript. In our implementation we use a subset of PostScript (i.e. we do not need the graphics and typographic operators of the language), that is enriched with a more sophisticated search procedure for the operators and names, in order to be able to write PF-programs in a more object oriented style.

As mentioned above, Java is not more suitable for our purpose than other general purpose languages. Java's distribution model is based on byte code for the Java virtual machine. In order to implement mobile code for exchanging policies, a just-in-time compiler would be necessary, and the agent would have to deal with two different representations of the code. In such an implementation Java code would be exchanged instead of byte code.

Even if we would choose Java, we could not make use of the mechanisms that the Java concept offers with regard to mobile code. Only the byte code for the Java virtual machine is mobile with normal use. However, the byte code is not object oriented, and modifications of the code would be quite difficult to achieve. A solution could be to compile the byte code to Java. However, we consider such a model to be too complicated for a prototype.

3.3. The PF language

PF is an interpreted programming language which is derived from PostScript [26, 18]. However, the graphics and typographic parts of the language definition are not implemented, and there are minor changes in the key words, for practical reasons. PF has an additional feature that changes the search of names within the interpreter. This facility gives PF the possibility for multiple inheritance when the variable `__PARENT__` is set properly in an object. Objects are implemented as dict in PF.

Usually in Postscript the search path for names is to search the dicts in the dict stack.

The PF feature `__PARENT__` defined for each dict can contain a list (array) of dicts, which are searched first recursively. Using this feature, objects can be defined which inherit from parent directories. In the PF language, a dict can be used as an object. A dict is a collection of name-value pairs.

PostScript is stack-based. Therefore, the language is mostly independent from devices and implementations. However, PostScript is also rather unusual for use as a programming language. As PostScript is stack based, the syntax is somewhat "backwards" to a human programmer, but this concept permits a low overhead in processing and storage. Each instruction can be acted upon immediately, without the need to buffer large amounts of the program.

The implementation of the PF-interpreter and run time system was done by M. Linsenmann around 1990, and used in several occasions, e.g., [27]. The implementation is somewhat unfinished, and several operators are missing. Some differences to original PostScript are unintended, and a result of not being prioritised while the system was implemented. However, other differences to PostScript are intended, e.g., the inheritance (search rules) of the interpreter. The main issues for the use of PF and its interpreter are:

- Study language constructs, and use the system as an object of study and experimentation.
- Use the interpreter to build flexible systems for prototyping.

3.4. Negotiation in PF

In the following we show a few examples to illustrate the principles negotiations are based upon, including code-passing of PF. It is not intended to give a complete overview, rather than to show the flexibility of the concept. The functions shown are to be considered as examples.

3.4.1. Basic negotiation

Example 1: In an object "player" the value of the variable "framerate" is set to 25. In the next line the value of "framerate" is printed.

```
{ /framerate 25 def } player send
/framerate player send =
```

Example 2: The frame rate for delivering a video can be expressed from the source, the network throughput, and the capabilities on the sink. However, the network is unaware of the term "frame rate". Therefore we could define the source framerate as a function of the frame rates provided by the server and a profile of the throughput of the network. In that case the frame rate of the source is provided as this function, containing variables which value is unknown for the source (server). A simple example for such a definition could be (frame rate 25 when network throughput greater than 5Mbit/s, else 12Mbit/s).

```
\server-framerate {
   network-throughput 5000000 gt { 25 } { 12 } ifelse
} def
```

Example 3: In the service agent several objects are connections, which define an environment in which commands are executed. Therefore all connections are kept in a dict. When executing commands, the operator `ConnSend` is used to execute the operator within the right environment.

```
{ server-framerate = } /player connSend
```

Example 4: Procedures are defined as executable arrays but also used as an array of executable commands, that can be printed, altered, or sent to another recipient. The communication library is added as a separate package into PF. `SendToApplication` is used to send a text string to another application. For our purposes, where procedures / code is to be sent between the service agents, a procedure `CodeSend` is implemented, that converts an executable array to a text string, and sends it to another application (channel) with the specified opcode. The implementation is as follows:

```
/CodeSend {
  512 string dup 0 ssetlength
  3 index CVSA 2 index 2 index SendToApplication pop
  pop pop pop
} def

/ReturnSend {
   @eval@ 123 ev_channel CodeSend
} def
```

Example for use of `CodeSend`, where the procedures `CodeSend` and `ReturnSend` are implemented at the remote agent in the same manner as on the own host:

```
{ /CodeSend /CodeSend load @@ def } @eval@ 123 (sacp) CodeSend
{ /ReturnSend /ReturnSend load @@ def } @eval@ 123 (sacp) CodeSend
```

The example above shows an extension to the evaluation principle of PF for the service agent. Normally the executable arrays are only evaluated when necessary by the operator. A call of `exec` would evaluate the whole executable array. However, in the service agent partial evaluation must be possible, where the operator is substituted with the result of the evaluation. To mark this evaluation the pseudo-operator `@@` is used. A subsequent call of `@eval@` performs the substitution in the current context. Note: `@@` is a pseudo operator, and does not necessarily have an implementation. We show also how the routine `@eval@` is implemented:

```
/@eval@ {
   [ exch
   { dup /@@ cvx eq { pop exec } if
     dup type 68 eq { @eval@ } if
   } aforall
   ] cvx
} def
```

Example 5: The following example shows a simple negotiation: The frame rate that is defined for "player" in the remote agent is queried. The response from the remote agent is an executable definition of the value in the form `/player-frame-rate 25 def` at the SACP opcode 122.:

```
{ { { /player-frame-rate frame-rate @@ def }
    ReturnSend
  } /player ConnSend
} 122 (sacp) CodeSend
```

3.4.2. Values, Functions and Retrieval

In the service agent the information and characteristics on the applications is kept in dicts. For each connection or application one dict is kept that contains information on QoS parameters in the form of:

- values (in PF values are functions that return a constant result);
- functions on how to calculate the value from other known values;
- retrieval functions that describe how to retrieve the functions for getting a value from a remote agent.

Due to the dynamic inheritance structure of PF, an optimisation can be performed that stores retrieved functions, so that they can be used later, and therefore avoid multiple evaluation of the same expression. To achieve this, we use the inheritance mechanism of PF.

For each connection or application four dicts are defined in addition:

FUNC: used to store all functions and values;

CACHE: cached values are stored here;

RETRIEVE: used to store the functions to retrieve the functions or values.

DEFAULT: used store the functions to retrieve functions or values.

The functionality to retrieve the functions is asynchronous. Therefore we cannot wait for the response, for a function to arrive, and be executed. Therefore, it is important to note that retrieve-functions could e.g., include code, that could be used to trigger new calculations.

In order to bind the four dicts into the main dict of the connection by using the inheritence, the following call is used:

```
/__PARENT__ [ CACHE FUNC DEFAULT ] def
```

A cache functionality can be easily achieved. Before using a function, we have to check, whether the function is available in FUNC. If it is not available, we have to retrieve the function from the remote agent. This function is not available directly, and the SA has to decide, whether to use a preliminary value, that might be stored in the DEFAULT dict. An example for retrieving a function might be:

```
/Exec&Retrieve {
   %  name --> value
   dup FUNC exch known {
     Exec&Cache
   } {
     dup RETRIEVE exch send
     cvx exec % found in CACHE or DEFAULT
   } ifelse
} def
```

Storing a value in the cache is not automatic yet, and values have to be put into the stack using a procedure like the following:

```
/Exec&Cache {
   % name --> value
   dup cvx exec
   exch 1 index
   CACHE 2 index known {
      pop pop
   } {
      CACHE begin def end
   } ifelse
} def
```

3.4.3. Stream-Binding in PF

The stream binding mechanism consists of the definition of streams and the constraint function. Instead of the language FIDL defined by Mehus [20], we want to express the stream binding mechanism entirely in PF. Paulsen [17] used PF in the implementation, but the description of the stream binding was not done entirely in PF.[2] In the following, we show a way which constructs of PF can be used for this purpose.

Stream descriptions in general can be defined as objects or nested objects. In PF we could use a dict (*dict*) for this purpose. Dicts are a collection of name-content pairs. Each stream definition could be done in separate objects. As an example, the following FIDL definition (left) could be described as shown on the right:

```
stream StreamDocument {
  source flow FlowDocument {
     text htm {
        encoding = "HTML";
        version  = "2.0";
     }
     image gif {
        encoding = "GIF";
        version  = "87a";
        height   = 200;
        width    = 100;
     }
     constraint htm & gif;
  }
}
```

```
/StreamDocument <<
  /__PARENT__ [ Stream ]
  /FlowDocument <<
     /__PARENT__ [ Source_Flow ]
     /htm <<
        /__PARENT__ [ Text ]
        /encoding (HTML)
        /version  (2.0)
     >>
     /gif <<
        /__PARENT__ [ Image ]
        /encoding (GIF)
        /version  (87a)
        /height   200
        /width    100
     >>
     /constraint { htm gif @and@ }
  >>
>> def
```

Using this method, each stream definition is one object. The constraint definition is defined as an executable array in PF, which is resolved when the media configuration is evaluated. The mechanisms behind would be similar as in the FIDL definition. The special operators `@and@`, `@or@`, etc. are distinct from the logical operators `and`, `or`, etc.; these can only be used in connection with the constraint function. Besides the constraint definition the object can also define other functions like the appraisal function.

Alternatives could be expressed in PF like this:

[2] Therefore, an extra parser for some data structures had been necessary to implement.

```
/size { 17 19 @alternative@ } def
/size { [ 17 19 20 21 ] @alternative@ } def
```

Other possibilities could include preferences in a definition like

```
/size {
   [ 17 19 21 ] @preferred@
   [ 19 23 ]    @possible@
   @alternative@
} def
```

The semantics of these functions is implementation dependent. Note, that the definition of these functions can be exchanged between the participants of the negotiation.

The purpose of the descriptions above was to show the principles and possibilities of negotiation in SACP in some examples. While parts of the system are implemented, an implementation still is to be performed for the entire system.

4. Discussion

In this section we discuss a few open points, and give a conclusion.

4.1. Security

The use of agents involve always an additional security concern in an application or system. The security concerns include:

- A login procedure should be employed. This is possible to integrate into the session protocol. However this is not yet implemented.
- It is possible to introduce an encryption mechanism into the session protocol. Other possibilities would include the use of Open SSH and port forwarding.
- Malicious or malformed code could crash the service agent. The interpreter has a try-catch mechanism that can recover from these problems. However, there is still some work to be done for this feature being fully implemented.
- PF includes the possibility to log messages in PF, which could be included at a later run, e.g., for error recovery purposes. A possibility would be to write the current state in PF to a file, perform a recovery with the same state that is written to the file.

4.2. Screen shots from the implementation

Figure 7 shows an screen shot from the Implementation by Paulsen. The system uses the well-known `vic` application, which was adapted to the SACP protocol. This was done in order to show that a real application can be connected with SACP.

The two upper windows in the screen shot belong to the User Agent. The "smiley face" constitutes the main window of the User Agent, which is used to control other child windows, such as the configuration selection dialogue, which is labelled "gap/vic" in the example. Here the user chooses between available media encodings and parameter qualities. The resulting configuration is communicated to the Service Agent at the remote host, and if accepted, used as the sending application's source configuration.

The two windows below belong to the SACP-enhanced vic application. On the right side is the main window of vic, showing a thumbnail image of the received video stream. The "menu" button is generally not used when using vic in the SACP framework, because transmission

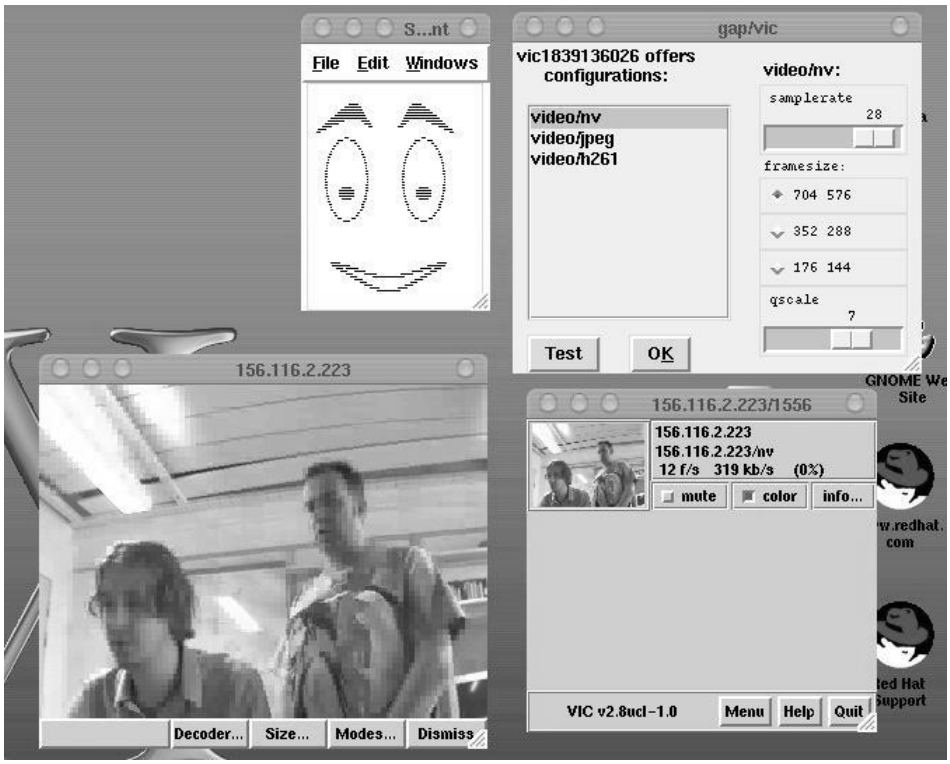


Figure 7: Screen shot from vic used with SACP

is controlled by the receiving side. On the left side is an enlarged version of the thumbnail image.

4.3. Conclusion

In this document we have described the ENNCE service agent framework. We have shown the flexibility of the framework in some examples, and the conformance to other, more general frameworks, like the RFC 2703. Parts of this system have been implemented within two master theses, and the work funded by the Norwegian Research Council.

The work by Michelsen [8] looked at the user aspects within the framework. Paulsen [17] implemented a prototype of the negotiation and agents. However, in both attempts the use of PF has not been implemented so profound as desirable, because some technical necessities have directed the implementors other ways. Therefore, much work has been left for later implementations, that would include the implementation of the entire system.

In our work we did not implement all parts. Especially the important part of setting up the stream, communicating with RSVP, etc. have not been performed as intended. A reason for this is the late implementation of the network, which was delayed by unexpected technical reasons (cf. [14]).

4.4. Future work

The work of WP1 of the ENNCE project is relevant for mobile applications. Especially for multimedia-online-applications with synchronised media content it is important to have the possibility for QoS negotiation techniques. The ENNCE architecture was considered as a basis

for several projects, e.g., for implementing a client-server based multimedia application used on mobile terminals with varying network and terminal QoS properties, where also renegotiation issues were of interest.

References

[1] W. Leister and P. Spilling. A service Agent for Connection and QoS Management in Multimedia Systems. notat IMEDIA/05/98, Norsk Regnesentral, Oslo, 1998.

[2] H.O. Rafaelsen and F. Eliassen. Trading and Negotiating Stream Bindings. In *Proc. IFIP/ACM Middleware'2000*, april 2000.

[3] Ø. Hanssen and F. Eliassen. Policy Trading. In *Int. Symp. Distributed Objects and Applications, Antwerp*, pages 219–227, 2000.

[4] F. Eliassen and H.O. Rafaelsen. A Conformance Relationship supporting Selection of Explicit Stream Bindings. In *Proc. NIK'99, Trondheim*, pages 69–80, 1999.

[5] F. Eliassen and H.O. Rafaelsen. A Trading Model of Stream Binding Selection. In *Proc. Smartnet'99, Bangkok*, pages 251–264. Kluwer, 1999.

[6] T. Plagemann, F. Eliassen, V. Goebel, T. Kristensen, and H.O. Rafaelsen. Adaptive QoS Aware Binding of Persistent Multimedia Objects. In *Proc. DOA'99*, 1999.

[7] F. Eliassen and J.R. Nicol. Supporting interoperation of continuous media objects. *Theory and Practice of Object Systems: Special Issue on Distributed Object Management*, 2(2):95–117, 1996.

[8] T. Michelsen. *Bruk av agent-teknologi ved brukerintegrert QoS-forhandling.* Cand.Scient Thesis in Computer Science, Institutt for Informatikk, University of Oslo, 1999.

[9] T. Kristensen and T. Plagemann. Enabling Flexible QoS Support in the Object Request Broker COOL. In *Proc. International Workshop on Distributed Real-Time Systems (IWDRS 2000)*, april 2000.

[10] F. Eliassen, T. Kristensen, T. Plagemann, and H.O. Rafaelsen. MULTE-ORB: Adaptive QoS Aware Binding", (position paper). In *Workshop on Reflective Middleware (RM 2000), Proc. IFIP/ACM Middleware'2000*, april 2000.

[11] T. Plagemann, F. Eliassen, B. Hafskjold, T. Kristensen, R. Macdonald, and H.O. Rafaelsen. Managing Cross-Cutting QoS Issues in MULTE Middleware, (extended abstract). In *Proc. Workshop on Quality of Service in Distributed Object Systems (QoSDOS), in association with 14th European Conference on Object-Oriented Programming (ECOOP 2000)*, june 2000.

[12] W. Leister and P. Holmes. Characterization and selection of applications for ENNCE WP1. notat IMEDIA/03/99, Norsk Regnesentral, Oslo, 1999.

[13] L. Aarhus, J. Riisnæs, and T. Karlsen. An experimental network infrastructure supporting QoS. notat IMEDIA/02/99, Norsk Regnesentral, Oslo, 1999.

[14] L. Aarhus, E. Fjellheim, and J.-O. Eide. ENNCE - Final QoS-controllable Network Infrastructure. notat IMEDIA/02/01, Norsk Regnesentral, Oslo, 2001.

[15] Draft Internation Standard. *ISO-10746-1: Basic reference model of Open Distributed Processing.* ISO/IEC JTC1/SC21/WG7, 1995.

[16] Y. Ding and R. Malaka. An Agent-based Architecture for Resource-Aware Mobile Computing. In *Proc. IMC2000, Rostock-Warnemünde*, 2000.

[17] G. Paulsen. *Hovedfagsoppgave, to appear.* Cand.Scient Thesis in Computer Science, Institutt for Informatikk, University of Oslo, 1999.

[18] Adobe Inc. *Postscript Language Reference.* Addison-Wesley, 1999.

[19] G. Klyne. A syntax for describing media feature sets. *IETF*, RFC 2533, 1999. Standards Track.

[20] S. Mehus. *Type Checking and Binding of Stream Interfaces in a Multimedia Database System.* Cand.Scient Thesis in Computer Science, Department of Computer Science, University of Tromsø, 1997.

[21] G. Klyne. Protocol-independent content negotiation framework. *IETF*, RFC 2703, 1999. Informational.

[22] K Holtman and A. Mutz. Transparent content negotiation in http. *IETF*, RFC 2295, 1998. Experimental.

[23] A. Danthine and O. Bonaventure. From Best Effort to Enhanced QoS. In O. Spaniol, A. Danthine, and W. Effelsberg, editors, *Architecture and Protocols for High-Speed Networks.* Kluwer Academic Publishers, 1994.

[24] Burkhard Stiller, George Fankhauser, Bernhard Plattner, and Nathalie Weiler. Pre-study on customer care, accounting, charging, billing, and pricing. Pre-stydy, ETH Zürich, 1998.

[25] Burkhard Stiller. Overview of billing systems for internet service providers. Pre-stydy, ETH Zürich, 1998.

[26] G. Reid. *Thinking in Postscript.* Addison-Wesley, 1990.

[27] W. Leister. *Geometrisches Modellieren durch interaktive Rekonstruktionsmethoden.* Dissertation, Fakultät für Informatik, Universität Karlsruhe, 1991.

A. The SACP

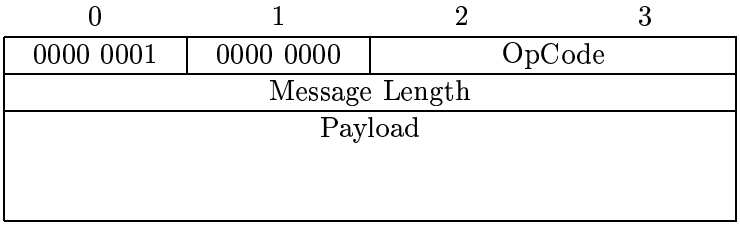
A.1. Format

The SACP protocol comes with two formats: The *Network Format* is for use on networks. where it is not necessary for humans to read the contents. The second format is textual. It is intended for use when saving messages on files or when messages are supposed to be entered by a human operator (e.g. for debugging purposes). The protocol-library detects the format automatically.

The SACP protocol is based on a connection oriented network protocol on the application level. When a connection is opened, a sequence of messages is transferred. Some state information is kept, e.g. the names of the connections.

All messages have an opcode and a payload. The contents of the payload is characterised by the opcode. Opcodes are application defined. However, some opcodes are system-defined, and used to maintain connections.

A.1.1. Network Format



A.1.2. Text Format

`*opcode*payload#`	message with payload
`*opcode#`	empty message
`#`	end of message sequence

A.2. Opcodes

The reserved opcodes in SACP are defined in Table 1. Other opcodes are defined by the application. Opcodes can be used to distinguish different types of messages. Usually the SENDADDR message is sent first, followed by Protocol version numbers, authentification, password check, and protocol switch follow. Thereafter the payload information starts.

A.2.1. SENDADDR opcode

After opening a connection for the SACP protocol, the sender identifies itself using opcode 0. In file-format this might look like

```
*0*myname#
```

A.2.2. Portmapper opcode

The portmapper opcode is used to locate other servers within the SACP domain. A participiant can be notified with the portmapper opcode where an application with a specific name

Code	Name	Payload	Explanation
0	SENDADDR	`name`	identification (senders address)
1	NOP	—	empty command
2	VERSION	`version`	protocol version number
3	PORTMAP	`connection host:port`	portmapper message
3	PORTMAP	`connection 0`	clear portmapper message
3	PORTMAP	`connection`	portmapper query
4	BROADCAST	`connection opcodes...`	specify opcodes for broadcast
5	PSWITCH	—	switch to text format
6	AUTH	`username`	specify username
7	PASSWD	`password`	specify password
8	—		
9	PF	`pf-code`	pf-message

Table 1: Reserved Opcodes in SACP

can be found with respect to hostname and port number. (Note: This makes only sense for servers!). By sending the value 0 instead of `hostname:portnumber` the entry for this name is deleted from the database. Requests are sent by sending the name of the application only. (Note: The requested application may or may not answer!).

Example: An application `myname` tells its own address, deletes the entry for `thisiswrong` and requests the data for the name `wanttoknow`. (Note: The receiver does not have to answer to any request. The communication is asynchronous.

```
*3*myname myhost:1234#
*3*thisiswrong 0#
*3*wanttoknow#
```

A.2.3. Broadcast Opcode

The library permits to send a message to multiple recipients. All recipients that want to receive a message that is broadcast register with the sender by giving the opcodes they are interested in. When using the communication library broadcast call these messages are sent to all interested recipients. Note: This is not a real broadcast as for IP, but a sequential message passed to several recipients. The syntax is as follows:

```
*4*connection br-opcode
opcode-list#
```

The following `br-opcodes` are defined:

- 0: delete all broadcast opcodes for the specified connection.
- 1: add the opcodes in opcode-list for the specified connection.
- 2: delete the opcodes in opcode-list for the specified connection.

Example: Register opcodes 17, 23, and 44 for broadcast and unregister opcodes 24 and 99:

```
*4*myconnection 1 17 23 44#
*4*myconnection 2 24 99#
```

A.2.4. Often used sequences at startup

Client applications should use the following startup sequence:

```
*0*myname#
*5#
```

Server programs, that listen to a port that can receive the SACP protocol should have the following startup sequence:

```
*0*myname#
*5#
*3*myname myhost:myport#
```

B. The communication library

A communication library in the programming language C is used. It consists of the following modules:

- addutil (additional utilities)
- comlib (main communication library)
- libihix (binding for use with X11)
- libihim (binding for use without X11)

B.1. The comlib interface

```
extern void     InstallRecvSocketHandler(void (*CB)(), char *CBD);

extern void     InstallConnectCallbackRoutine(void      (*CB)(char*));
extern void     InstallDisconnectCallbackRoutine(void   (*CB)(char*));

extern int      SendToApplication(char *connection, int jobno, char *messptr);
extern int      SendToApplicationBinary(char *connection, int jobno,
                                        char *messptr,    int messlen);

extern int      OpenClient(char *connection);
extern int      OpenServer(char *connection, int portnumber);

extern int      ConnectionAvail(char *connection);
extern int      ConnectionForce(char *connection);

extern int      ConvertAddressToPortnumber(char *portname);

extern int      SendPortmapMessage(char *connection, char *conninfo);
extern int      InterpPortmapMessage(char *connection, char *message);
extern int      DistributePortmapInformation(char *connection);

extern void     socketcomClose();
```

B.2. The IHINTERFACE

The IHINTERFACE is the interface towards the operating system, and implements methods used within the comlib. There are two bindigs, one for use with the X-Windows system, and one for standalone use. The following interface is common for both bindings:

```
extern int      InstallComInputHandler(CommunicationDescriptor*, void (*IH)());
extern int      RemoveComInputHandler(CommunicationDescriptor*);
```

The standalone binding has these calls in addition:

```
extern void IHI_MainLoop();
extern void IHI_Dispatch();
extern int  InstallTickerHandler(int duration, void (*IH)());
extern int  RemoveTickerHandler();
```

B.3. The broadcast interface

```
extern int BroadcastBinary(int,char*,int);
extern int Broadcast(int,char*);

extern int OrderBroadcast(char*,int);
extern int CancelBroadcast(char*,int);
extern int ClearBroadcast(char*);
```

B.4. The fileinterp interface

```
int FileReadCommand(FILE*,int*,char*);
void FileInterprete(FILE*);
int BufferReadCommand(char*,int*,char*);
```

B.5. The pipecom interface

```
int initPipeCommand(char*);
int startPipeCommand(char*,int,int);
int stopPipeCommand();
int InstallRecvPipeHandler(void (*CB)(),void*);
```

B.6. The portmap interface

```
int  AddPortmapEntry   (char*, char*, char*);
int  RemovePortmapEntry(char*);

/* Note: The return pointers for QueryPortmapEntry
 * must not be used after a new call of a routine of portmap
 * if these values should be needed: make a copy with strdup
 */
int  QueryPortmapEntry (char*, char**, char**);

int  KeepPortmapEntry(char*,int);
```

B.7. Example program

The following example shows how to use the communication library.

```
/*-------------------------------------------------------------
 * Copyright (c) 1998 Wolfgang Leister
 *               Norsk Regnesentral
 *
 * This program published under the conditions of the GPL
 * The copyright remains with the author
 * This program comes WITHOUT ANY WARRANTY
 * -------------------------------------------------------------*/

#include <stdio.h>
#include <mcall.h>
#include <addutil.h>
#include <appaddr.h>
#include <ihim.h>
#include <socketcom.h>
#include <cominterp.h>
#include <stdlib.h>
#include <sys/types.h>
#include <signal.h>
#include <sys/wait.h>

/*****************************************************/
/* Global Variables */
char       *OWN_ADDR   = NULL;
AppAddress *Self       = NULL;

void connectCB(char *a)
{
   if (a) {
      fprintf(stderr,"ConnectCB connected: %s\n",a);
   }
} /* connectCB */

/*****************************************************/
void comint_NOINTERP_CB(char *vmeD, char *clientD)
{
  extern int comintLastOpcode;
  fprintf(stderr,"NOINTERP: OPC=%d\n",comintLastOpcode);
} /* comint_NOINTERP_CB */

/*****************************************************/
void comint_17_CB(char *vmeD, char *clientD)
{
  fprintf(stderr,"Received message: %s %s\n",vmeD,clientD);
} /* comint_17_CB */

/*****************************************************/
void tickerCB()
{
  fprintf(stderr,"TICKER\n");
  SendToApplication("sacp2",18,"Ticker");
```

```
} /*tickerCB*/
/*****************************************************/

main(int argc, char *argv[])
{
  extern int socketcom_debug;
  socketcom_debug = 1;
  Self = sscanAppAddress("sacp1","sacp1","localhost");
  MCALL(Self,SetHost)(Self,OWN_ADDR);

  InstallConnectCallbackRoutine(connectCB);
  InstallRecvSocketHandler(comintInterpreter,NULL);
  InstallTickerHandler(50000,tickerCB);
  comintInterpInit();
  comintInterpSetDefault (    comint_NOINTERP_CB, NULL);
  comintInterpSetCallback(17, comint_17_CB,       NULL);

  if (OpenServer(Self->connection,
                 ConvertAddressToPortnumber(Self->portname))<0) {
     fprintf(stderr,"Cannot open Socket Connection");
  }
  SendToApplication("sacp2",17,"Put this in queue");

  IHI_MainLoop();
} /* main */
```

C. Binding to pf-Interpreter

The binding between the communication library and the pf-Interpreter is based on a previous implementation between X Windows and the pf-Interpreter.

C.1. Events from communication library to pf

```
#define MAX_EVENTS 20

#define PF_GETEVENT 17
#define PF_POLL      0
#define PF_EXIT     99

// Events 1-8 define for other purposes
#define ComCode          9
#define ComTimer        10
#define ComConnect      11
#define ComDisconnect   12
#define ComMessage      13

typedef struct {
  short ev_type;
  char *ev_contents;
  char *ev_channel;
  int   ev_jobtype;
} PfEv_Event;
```

```
PfEv_Event *PfEv_PutEvent();
PfEv_Event *PfEv_GetEvent();
int PfEv_ProcessEvents();
void PfEv_DispatchInput();
int SetEvent();
int InitEvents();
```

C.2. A program-example

This program example starts the pf-interpreter, and the communication library. Thereafter, events from the communication library are sent to the PF interpreter in an endless loop.

```
#include <stdio.h>
#include <mcall.h>
#include <addutil.h>
#include <appaddr.h>
#include <ihim.h>
#include <socketcom.h>
#include <cominterp.h>
#include <stdlib.h>
#include <sys/types.h>
#include <signal.h>
#include <sys/wait.h>
#include <time.h>
#include <global.h>
#include <pf.h>

#include "events.h"
#include "comintcb.h"

/*************************************************/
/* Global Variables */
char *OWN_ADDR    = NULL;
AppAddress *self  = NULL;
/*************************************************/

char    *reqbuf  = NULL;
char   **environPtr;
char    *ProgFileName;

main(argc,argv,envp)
int  argc;
char  *argv[];
char     *envp[];
{
  /* from pf - begin */
   extern int _pf_status;
#if LINUX
   extern char **environPtr;
   environPtr = envp;
#endif
   /* from pf - end */
  extern int socketcom_debug;
```

```
  socketcom_debug = 1;
  ProgFileName = argv[0];
  pf_init();
  InitEvents();

  self   = sscanAppAddress("sacp","sacp","localhost");
  MCALL(self,SetHost)(self,OWN_ADDR);
  reqbuf = (char*) malloc(100*sizeof(char));
  sprintAppAddress(reqbuf,self);
  fprintf(stderr,"Server Connection %s is set to %s\n",
  self->connection,reqbuf);

  InstallConnectCallbackRoutine(connectCB);
  InstallDisconnectCallbackRoutine(disconnectCB);
  InstallRecvSocketHandler(comintInterpreter,NULL);
  InstallTickerHandler(500000,tickerCB);
  comintInterpInit();
  comintInterpSetDefault(comint_NOINTERP_CB,NULL);
  comintInterpSetCallback( 3 ,comint_OPCM_PORTMAP_CB ,NULL);
  comintInterpSetCallback( 9 ,comint_Code9_CB        ,NULL);

  if (OpenServer(self->connection,
     ConvertAddressToPortnumber(self->portname))<0) {
     fprintf(stderr,"Cannot open Socket Connection");
  }

  pf_boot();
  pf_argv(argv);
  while (!(_pf_status=pf_run()));
  PfEv_DispatchInput();
  pf_end();
  exit(0);
} /* main */
```

C.3. events.pf

The file events.pf is needed for the pf-part of the binding:

```
% file events.pf

/Event 11 dict dup begin
   /ev_type   0 def
   /ev_window 0 def
   /ev_state  0 def
   /ev_detail 0 def
   /ev_mx     0 def
   /ev_my     0 def
   /cnt1 0 def
   /cnt2 0 def
end def

/EventLoop {
 {
      Event
```

```
      17 halt             % fill information in event
      dup begin /cnt2 dup cvx exec 1 add store end
      (Event:\n) print
      dup display_event
      pop                 % remove event from stack
  } loop
} bind def
```

C.4. Communication Library Calls in PF

The following definitions are used for the calls from PF to the communication library.

```
/SendToApplication   1 comopdef      % message opcode channel -> result
/OpenClient          2 comopdef      % chan                   -> result
/OpenServer          3 comopdef      % chan portnumber(int)   -> result
/CVPortnumber        4 comopdef      % portname               -> portnumber(int)
/ConnectionAvail     5 comopdef      % chan                   -> bool
/ConnectionForce     6 comopdef      % chan                   -> bool
/SendPortmapMessage  7 comopdef      % coninfo channel        -> result
/InterpPortmapMessage 8 comopdef     % message channel        -> result
/DistributePortmapInformation 9 comopdef % chan                 -> result
/SocketcomClose     10 comopdef      % ->
/Broadcast          11 comopdef      % msg opc         -> res
/OrderBroadcast     12 comopdef      % opc chan        -> res
/CancelBroadcast    13 comopdef      % opc chan        -> res
/ClearBroadcast     14 comopdef      % chan            -> res
/AddPortmapEntry    15 comopdef      % host port chan -> result
/RemovePortmapEntry 16 comopdef      % chan -> result
/QueryPortmapEntry  17 comopdef      % chan -> host port result
/KeepPortmapEntry   18 comopdef      % bool chan -> bool
/SocketcomDebug     19 comopdef      % int -> int
```

C.5. Callbacks for Opcodes in PF

Opcodes that are not mentioned in Table1 are visible in the pf-code program. The c-pf interface generates an event with the values for `ev_type`, `ev_contents`, `ev_channel` and `ev_jobtype` respectively.

The functionality for opcodes is programmed in pf within a dict **Callbacks**.

The functionality for mapping special events to opcodes is programmed in `comlibcb.c`. The hitherto reserved opcodes for SACP are used as follows:

Code	Name	Payload	Explanation
17	TICKER	date	ticker event
997	OPENCHAN	channel	generated instead of opcode 0
998	CLOSECHAN	channel	generated when channel is closed
999	EXIT	—	exit program
122	EXEC	pf code	exec code in channel dict environment
123	EXEC	pf code	exec code in event dict environment

D. PF-libraries for service agent

The following section shows libraries and convenience functions that are used to support the Service Agent.

D.1. Connections.pf

```
(Class connections\n) print
/Connections 30 dict def
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Callback redefinition of callbacks related to Connections

Callbacks begin
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % 997 = Open channel  (redefinition)
  997 {
(997: Open Channel ) print
ev_channel print (\n) print
Connections ev_channel    % cvn not necessary; ev_channel is NAME_TYPE
                                % in new implementation cvn can be used additionally.
1 index 1 index known {
   (channel ) print ev_channel print ( already known\n) print
           pop pop
} {
   exch begin
              /conn-dictsize Connections send dict def
   end
   (channel ) print ev_channel print ( generated\n) print
} ifelse
  } def % 997

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % 122 = execute in proper Connection environment
  122 {
(122: Exec ...\n) print ev_contents print (\n) print
Connections ev_channel
1 index 1 index known {
   dget begin
      ev_contents cvx exec
   end
} {
   (channel ) print ev_channel print ( unknown\n) print
   pop pop
} ifelse
  } def % 122
end

 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 % ... method conn-name -->
 /ConnBegin {
    Connections exch
    1 index 1 index known {
      dget begin
    } {
      pop /uchan dget begin
    } ifelse
 } def

 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 /ConnEnd {
```

```
    end
 } def

 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 /ConnSend { ConnBegin cvx exec ConnEnd } def

 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 Connections begin
    /conn-dictsize 10 def
    /uchan conn-dictsize dict def
 end
```

D.2. Callbacks.pf

```
(Class callbacks\n) print
/Callbacks 30 dict def

Callbacks begin
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % 999 =  stop and exit programme
  999 {
(999: Exiting ...\n) print
SocketcomClose
99 halt
  } def % 999

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % 998 = Close channel
  998 {
(998: Close Channel ) print
ev_channel print (\n) print
  } def % 998

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % 997 = Open channel
  997 {
(997: Open Channel ) print
ev_channel print (\n) print
  } def % 997

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % 123 = execute in current (i.e. event dict) environment
  123 {
(123: Exec ...\n) print ev_contents print (\n) print
ev_contents cvx exec
  } def % 123

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % 17 = ticker
  17 {
(17: Ticker ...\n) print ev_contents print
  } def % 123
end
```

D.3. Remproc.pf

```
(Install remproc.pf\n) print

% @@ is a pseudo-operator
%/@@ { } def

% cvs&sappend: string simple --> substring
/cvs&sappend { 0 cvs sappend } dup 0 512 string aput def

/@eval@ {
   [ exch
   { dup /@@ cvx eq { pop exec } if
     dup type 68 eq { @eval@ } if
   } aforall
   ] cvx
} def

/CVSA {
   % append object to string
   % string any -> substring
   { dup type 68 eq {
        dup xcheck {
  1 index ( {) sappend
          CVSA
          0 index ( }) sappend
        } {
  1 index ( [) sappend
          CVSA
          0 index ( ]) sappend
        } ifelse
     } {
        % dup (#) print type = (#) print
        dup type 6 eq {
          % names
          dup xcheck {
             1 index ( ) sappend
             1 index exch cvs&sappend
          } {
             1 index ( /) sappend
             1 index exch cvs&sappend
          } ifelse
        } {
  dup type 73 eq {
            % string type
            1 index ( \() sappend
            1 index exch cvs&sappend
            0 index (\)) sappend
          } {
            dup type 69 eq {
               % dict type
               1 index ( << ) sappend
               {
                 exch CVSA CVSA
               } dforall
```

```
               1 index ( >> ) sappend
             } {
               % all the other cases
               1 index ( ) sappend
               1 index exch cvs&sappend
             } ifelse
           } ifelse
         } ifelse
       } ifelse
    } aforall
} def
```

D.4. Program example test2.pf

```
(Dict  test2\n) print
/stacksize { (stack size: ) print count = ( \n) print } def
/pause { (---pause---\n) print input pop } def

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
/test2 300 dict def
test2 begin
/comopdef {  512 add opdef } bind def
(comlibop.pf) run

/nullproc { { } } def

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(callbacks.pf) run
(events.pf) run
(connections.pf) run
(remproc.pf) run

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Execute all files in argv (use of suffix is mandatory).
1 1 argv alength 1 sub {
  argv exch aget dup (Load ) print = (\n) print run
} for

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Open Client
3 SocketcomDebug =
1 SleepRetry pop
0 BindRetry pop
(ttt) OpenClient

(naos) (1360) (sacp) AddPortmapEntry pop

(Opened Socket Connection ) print = (\n) print

(hello) 56 (sacp) SendToApplication pop
/TheString 512 string dup 0 ssetlength def

TheString { /frame-rate 2977 def } @eval@ CVSA 122 (sacp) SendToApplication pop
```

```
TheString dup 0 ssetlength
{ { frame-rate 20 string cvs 88 ev_channel SendToApplication pop } /spiller ConnSend }
 @eval@ CVSA 122 (sacp) SendToApplication pop

/CodeSend {
% code opcode channel -->
% without @eval@
  512 string dup 0 ssetlength
  3 index CVSA 2 index 2 index SendToApplication pop
  pop pop pop
} def

/ReturnSend {
   @eval@ 123 ev_channel CodeSend
} def

{ /CodeSend /CodeSend load @@ def } @eval@ 123 (sacp) CodeSend
{ /ReturnSend /ReturnSend load @@ def } @eval@ 123 (sacp) CodeSend

{ { { /spiller-frame-rate frame-rate @@ def }
  ReturnSend
  } /spiller ConnSend
} 122 (sacp) CodeSend

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

EventLoop
quit
end
```