An Implementation Structure for RM-OSI/ISO Transaction Processing Application Contexts

Flávio Morais de Assis Silva
Edmundo Roberto Mauro Madeira

Relatório Técnico DCC–11/93

Maio de 1993

An Implementation Structure for RM-OSI/ISO Transaction Processing Application Contexts

Flávio Morais de Assis Silva
Edmundo Roberto Mauro Madeira*

*Departamento de Ciência da Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP. Pesquisa desenvolvida com suporte financeiro do CNPq — Conselho Nacional de Desenvolvimento Científico e Tecnológico sob processo 130765/90-2

Abstract

This technical report presents a structure for a modular implementation of application contexts from the RM-OSI/ISO (*Reference Model - Open Systems Interconnection / International Organization for Standardization*) in which the TP (*Transaction Processing*) protocol participates. This protocol provides services to support the execution of distributed atomic transactions in the OSI environment. The structure presented influences the implementation of other application contexts.

In this text it is also shown the way the upper layers protocols are being implemented in a didactic communication system called SISDI-OSI (*Sistema Didático para o Modelo OSI – Didactic System for the OSI Model*) in terms of process configuration and interprocess communication mechanisms. The experiences with protocol implementations for this system are then commented.

1 Introduction

Defined as a group of related operations that are executed so that the ACID (Atomicity, Consistency, Isolation, Durability) properties are guaranteed [16], an atomic transaction constitutes an appropriate abstract concept for the execution of operations on environments subject to faults and where there is resources sharing, since transactions make errors and concurrency aspects transparent for the user. Distributed systems in general are examples of such environments. Atomic transactions executed on such systems are called *Distributed Atomic Transactions*.

TP (*Transaction Processing*) [16, 17, 18] and CCR (*Commitment, Concurrency and Recovery*) [11, 12] are the protocols from the reference model RM-OSI/ISO (*Reference Model – Open Systems Interconnection / International Organization for Standardization*) [5] that provide services at the Application layer of this model for the support of distributed atomic transactions execution. TP and CCR are based on the *Two-Phase Commit with Presumed Abort* protocol [22], that is a distributed transaction commit protocol known from the database area.

The TP and CCR services can be used for an atomic execution of a set of operations provided by other application protocols. For example, these protocols can be used together with RDA (*Remote Database Access*) for the atomic execution of operations to access remote databases [14, 15]. In such case, the TP and CCR services delimit the transaction and control its termination, while the RDA services determine the kind of operations that constitute the transaction.

This technical report presents a logical structure for the implementation of application contexts in which the TP protocol participates. These application contexts are referenced hereafter by *TP Application Contexts*. Then it is described the structure of a didactic communication system called SISDI-OSI (*Sistema Didático para o Modelo OSI – Didactic System for the OSI Model*) that is being developed at UNICAMP [21]. The implementation of the TP protocol for this system is in progress. Later on it is commented on how some components of this system were implemented. The structure for TP application contexts influences the implementation of other application contexts.

2 Application Layer Structure for TP Application Contexts

2.1 Application Layer Structure as defined by ISO

In the RM-OSI/ISO, the active elements that execute functions of a specific layer are called *entities* of this layer. In the Application layer then there are *Application Entities*, refered just by *AEs*. An AE represents a set of Application layer communication capabilities for the realization of some communication objective. There are so AEs for the access to remote files or to access databases in other systems. An AE provides communication services to *Application Processes*, or just *APs*. The APs represent the information processing activities of the users of the communication system.

The services provided by an AE are a composition of the services provided by the *ASEs* (*Application Service Elements*) that take part in

the AE. An ASE represents an element at the Application layer that provides communication services for the realization of a specific communication objective and comprises the protocol necessary for the support of such services. There are, for example, the RDA, FTAM (*File Transfer, Access and Management*), ROSE (*Remote Operations Service Element*) and ACSE (*Association Control Service Element*) ASEs, that provide services, respectively, for access to remote databases, for access to remote files, for the execution of interactive remote operations and for the establishment and release of *application associations* (defined right below). There is also an ASE for the CCR protocol and another for the TP protocol. Two ASEs in different systems exchange data units called *APDUs* (*Application Protocol Data Units*) for the realization of their protocol.

The use of the communication capabilities of an AE and the use of an AP in a specific occasion is modelled respectively by an *AEI* (*Application Entity Invocation*) and an *API* (*Application Process Invocation*). So that two AEIs can communicate they must establish previously an *application association*, or just *association*, that represents a context over which the data exchanges between the AEIs take place.

The Application layer structure as defined by ISO [13] is shown on figure 1. On this figure one can see that an AEI contains objects called *MACF* (*Multiple Association Control Function*) and *SAOs* (*Single Association Objects*). The MACF component represents functions that control related activities executed on more than one association. A SAO models communication functions and state information related to the use of just one association. There is one SAO for each association used by the AEI.

Each SAO is further decomposed in a component called SACF (*Single Association Control Function*) and ASEs. The ASEs in a SAO are the ones whose services are used on that association. Since ASEs provide specific communication functions, the realization of the activities on an association may require the use of the capabilities of more than one ASE. The SACF represents the component of the SAO that controls the interaction among the ASEs of this SAO and their use of the resources

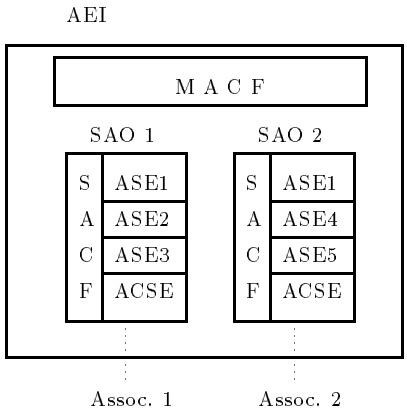


Figure 1: Application layer structure as defined by ISO

provided by the lower layers. Such resources are, for example, tokens and synchronization points from the Session layer or services for data transfer. All the lower layer resources are accessed through Presentation layer services.

The ASEs that provide the services used to form the operations to be carried out atomically are called U-ASEs (*User ASEs*). For an atomic transaction consisting of remote database access operations, for example, the RDA and ROSE ASEs are U-ASEs [14, 15].

So that two AEIs can exchange information through an association, they must be aware and follow a set of rules that will coordinate the communication. This set of rules is called an *application context*. Among the application context rules are, for example, the determination of which ASEs take part in the association, what are the event sequences allowed (these rules extend the rules established by each ASE individually) and which primitives are used to transmit the APDUs to the remote system. The exact application context to be used on an association is negotiated during the establishment of the association.

The application contexts are used, in a general way, to describe the

types of services that are provided through an association. For remote database access, for example, there are two application contexts [14]: the *RDA Basic Application Context* and the *RDA TP Application Context*[1]. The former allows atomic transactions to span over just one system, the database server. The latter allows the definition of atomic transactions that can span over several systems. The TP and CCR protocols participate in the *RDA TP Application Context* and are used exactly to provide the services for the control of this type of transactions.

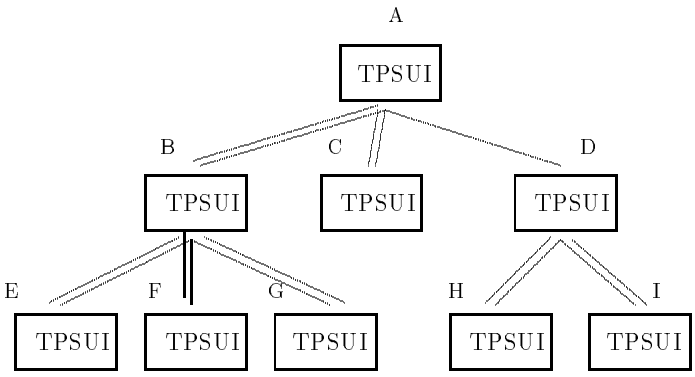
2.2 Application Layer Structure for TP Application Contexts

The *Two-Phase Commit with Presumed Abort* protocol assumes that the systems taking part in a distributed atomic transaction form a tree structure [22]. The nodes of this tree represent the systems and the branches of the tree form the communication topology, that is, the determination of who can send messages to whom. A system can only communicate with another one that is attached to it by a tree branch. Since TP and CCR are based on this protocol, for their operation it is also considered the existence of trees. Two types of trees are defined: *dialogue trees* and *transaction trees* (see figure 2). A dialogue tree contains at each node a *TPSUI* (*Transaction Processing Service User Invocation*). A *TPSU* (*Transaction Processing Service User*) represents a user of TP services. TPSUs are considered as being part of an AP. A TPSUI is an invocation of a TPSU, representing the use of TP services for a specific occasion of information processing. Each branch of the dialogue tree represents a TP *dialogue*. A TP dialogue represents a communication relationship between two TPSUIs, and establishes certain caracteristics of this relationship. This concept is by some means analogous to the *application*

[1] These application contexts, defined in [14], are called *generic application contexts*. In these contexts the parts dependent on the type of database are not specified. To each type of database there will be *specializations* of these generic application contexts that then will generate complete definitions of application contexts. One such specialization appears in [15].

association concept. A dialogue supports the establishment of new relationships between the TPSUIs called *transaction branches*. Two TPSUIs linked by a dialogue are not necessarily participating in a transaction.

A transaction branch represents effectively the relationship between two TPSUIs when they participate in a transaction. Transaction trees are composed of transaction branches. A transaction tree contains also TPSUIs as nodes and this tree indeed represents a transaction. More than one such tree can coexist over the same dialogue tree (see figure 2).



The TPSUIs from A to I and the arcs between them form a dialogue tree. On this dialogue tree there are two transaction trees: one formed by TPSUIs A, B, C, E, F and G and another formed by D, H and I.

Figure 2: Dialogue and transaction trees

The TP services are used for the construction of the trees and, if their user desires, also the subsequent control of the commitment or rollback of the atomic transaction. When a transaction is committed all its operations are executed and the results are made permanent. The rollback of a transaction restores all the data manipulated by it to their initial state, as if none of its operations were executed.

The user can also use just the TP services for establishment and pruning of the dialogue tree and for handshaking and control of a dialogue. This way the own user controls its transactions. CCR services are used by TP only when the TP controls the user transactions.

CCR services are defined to be used on just one branch of the transaction tree, to control the transaction on that branch. TP services are used to control the whole transaction tree. For this control TP uses CCR services on each branch. The TP services user, for example, issues just one TP-COMMIT.req primitive (a TP primitive) to request the commitment of the transaction. When the TP protocol machine receives this primitive, it generates events that cause a C-PREPARE.req primitive (a CCR primitive) to be issued on each association.

Examples of TP services are:

TP-BEGIN-DIALOGUE used to establish a new dialogue;

TP-END-DIALOGUE used to terminate a dialogue;

TP-HANDSHAKE used by the TP service users to synchronize their processing;

TP-COMMIT used by a TP service user to indicate that it has finished all the processing for the current transaction and to request that the transaction be committed;

TP-ROLLBACK used to cause the rollback of the transaction.

The CCR services are:

C-BEGIN used to begin a transaction branch;

C-PREPARE used by a node (superior) to ask one of its children (subordinates) in the transaction tree if it can commit the transaction or not;

C-READY used by a subordinate of a node in the transaction tree to answer its superior that it can commit the transaction;

C-COMMIT used by a node (superior) to request a subordinate to commit its part of the transaction;

C-ROLLBACK used by a node (superior or subordinate) to force the return of the transaction to its initial state;

C-RECOVER used to process recovery from faults. TP uses this service in a transparent way for its user.

Figure 3 shows the information flow in TP application contexts where the CCR is also used (for the contexts where CCR is not used this figure would be modified just by taking out the CCR ASE and its relationships). The full lines indicate the flow of primitives, while the dashed ones indicate the flow of APDUs. The services with names started by the letters TP-, U-, C-, A-, P- and SAF- refer to services provided by, respectively, TP, U-ASE, CCR, ACSE, the Presentation layer and the SACF (see below for SACF services). The interactions between the MACF and the TP-ASE are done through the use of services whose names start with AF-, from *Auxiliary Functions*. These services are used only inside the TP protocol machine.

Note that the APs interact just with the MACF, using only TP and U-ASE service primitives. The services of other ASEs, as the CCR ASE, are used to support these services. The TP services are used to specify the beginning of an atomic transaction and to coordinate its final result. The U-ASE services specify the operations to be carried out atomically.

From service requests done to it, the MACF interacts directly only with the SACF. In the data reception direction also all the primitives generated inside the SAO and from the Presentation layer are passed to the MACF by the SACF. This is done so that the MACF can control the events on the various associations and the SACF can control the events on one association, inside a SAO, performing activities as, for example, verifying the correct sequence of events according to the application context.

On the data sending direction, the SACF passes the primitives to the ASEs to which they correspond. The TP-ASE can send its APDUs either

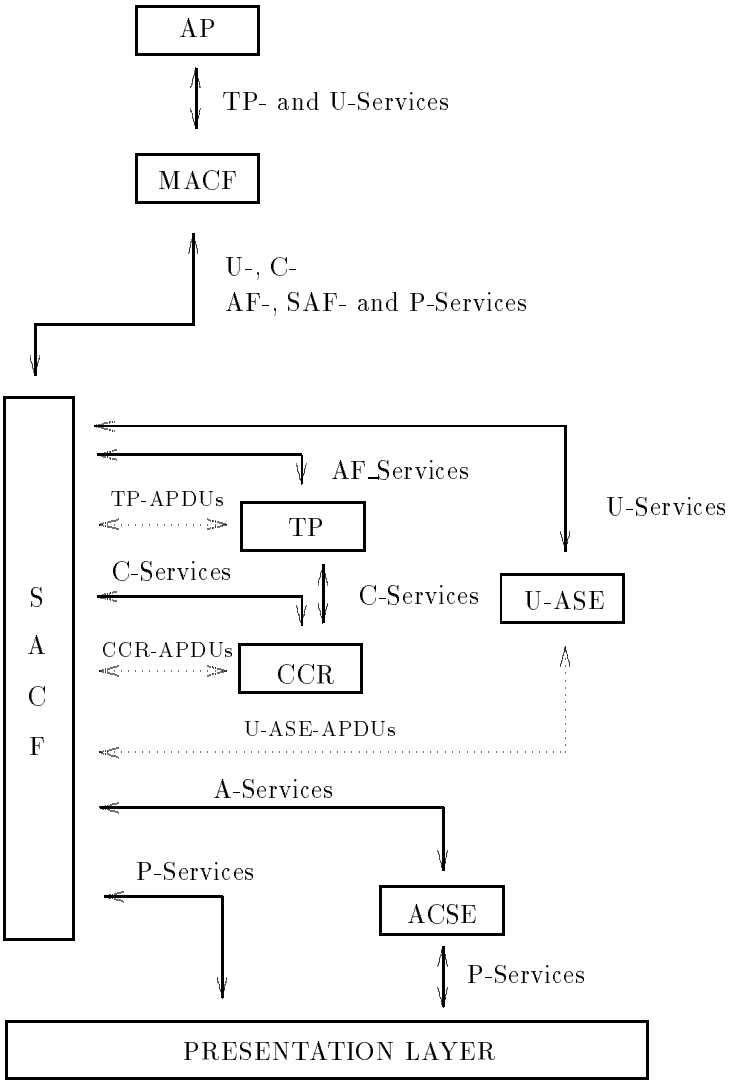


Figure 3: Information flow for TP application contexts

using CCR primitives or passing them to the SACF. The SACF receives, beyond APDUs from TP, also APDUs from CCR and from the U-ASE and send them to the remote system as values of user data parameters of Presentation layer service primitives. The CCR services can also be requested by the TP-ASE or by the SACF just for the control of the transaction, without sending any APDU as user data.

On figure 3 more than one U-ASE could exist. For example, in the *RDA TP Application Context*, mentioned previously, there are two U-ASEs: the RDA ASE and the ROSE ASE. The AP issues just RDA services. The RDA protocol uses ROSE services to convey RDA APDUs. In this case, the ROSE APDUs, containing embedded RDA APDUs, are passed to the SACF, that can concatenate them with TP or CCR APDUs.

The ACSE services are used by the SACF in TP application contexts and only these two components can issue Presentation layer service primitives directly. The ACSE uses the services for the establishment and release of presentation connections. An application association is always supported by a presentation connection. The SACF uses the other Presentation services. When the SACF receives primitives coming from the Presentation layer, it extracts the APDUs in the user data parameter (if any) and sends them to the adequate ASE.

The TP protocol specification defines the actions to be carried out by the TP-ASE, the MACF and the SACF components. The TP-ASE has as functions just to creat APDUs, as a result of the issue of a TP primitive by the TPSUI (that causes an issue of an AF- primitive by the MACF), and AF- primitives, generated as a result of the receipt of an APDU coming from the remote system. The SACF controls the events that take place in the SAO while the association is being used to support a dialogue (or *channel*, as it will be defined later) as also while it is not being used. The SACF functions will be better described later. The MACF controls the events related to the atomic transaction control for the realization of the *Two-Phase Commit with Presumed Abort* protocol. Note that this is the main part of the TP protocol. It is done by the MACF and not by the TP-ASE because the function of the *Two-Phase*

Commit with Presumed Abort protocol is indeed the control over events resulting from the communication with various systems, that in the RM-OSI/ISO is represented by events that happen on various associations. The MACF also serves as interface between the AP and the SAOs.

In the following sections, 2.3 to 2.7, it will be presented a logical structure for the implementation of TP application contexts. The structure is defined by a refinement of each component of the Application layer. Aspects of implementations already done are commented on section 3.

2.3 Structure for MACFs

Conceptually the MACF functions can be considered as being structured into the components shown on figure 4. The *Info* component represents the informations that are necessary for the control of the related activities executed on the associations or arisen from this control. Such kind of informations can be, for example, informations about the global state of the atomic transaction and informations about primitives or fields of previously sent or received primitives.

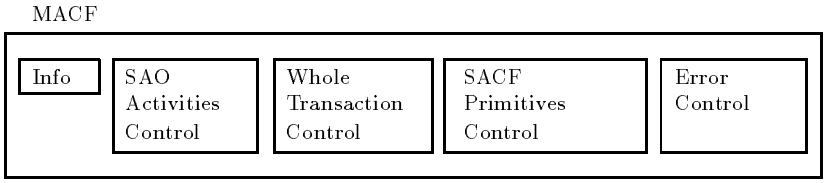


Figure 4: Structure for MACFs

The *Whole Transaction Control* component represents the handling of events related to the global activity executed on the several associations, that is, the control of events that are concerned with the whole transaction. For example, as was previously commented, the issue of a TP-COMMIT.req primitive by a TPSUI on an intermediate node causes the TP protocol to send informations to each of its subordinates,

i.e., causes activities to be done on all the associations controled by the MACF. This primitive then relates to the whole transaction.

The *SAO Activities Control* represents the handling of events related to just one branch (association) but that depends on or influences the global state of the activity. For example a TP-BEGIN-DIALOGUE.req primitive can not be issued during the termination phase of a transaction (depending on certain conditions). Although this primitive relates to just one association, the sequencing control for its issue depends on the global state of the transaction (the state on which the transaction is). Conceptually one can consider that there is one instance of the *SAO Activities Control* component to each SAO.

The *SACF Primitives Control* represents the treatment of primitives that refer to a SAO as a whole. The TP protocol defines two such primitives:

1. SAF-DETACH-ASSOCIATION.req: used by the MACF to tell the SAO that this SAO is no longer needed to be controled by this MACF; and

2. SAF-ASSOCIATION-LOST.ind: used by the SAO to tell the MACF that the association is no longer controled by this MACF. This primitive is issued when a problem is detected with the association, as, for example, the abrupt release of the association caused by a communication error (for instance the detection of a protocol error in any of the lower layers).

The *SACF Primitives Control* component handles then the issue of a SAF-DETACH-ASSOCIATION.req primitive and the receipt of a SAF-ASSOCIATION-LOST.ind primitive. Conceptually one can also consider that there is one instance of such component for each SAO controled by a MACF.

The *Error Control* component controls events indicating error on an association. These events can cause actions to be carried out on other associations. For example, when it is detected an association abort,

depending on the global state of the atomic transaction, the whole transaction must be rolled back. This means that informations must be transfered on several associations to coordinate a consistent termination of the transaction on all the systems that participated in it.

This MACF decomposition intends to clarify the understanding of the activities specified in [18] for the MACF.

2.4 Structure for SAOs

By considering the functionalities present on the association establishment process, it is convenient to decompose a SAO in two components: the *Association Establisher* and the *Application Context Specific Part* (figure 5).

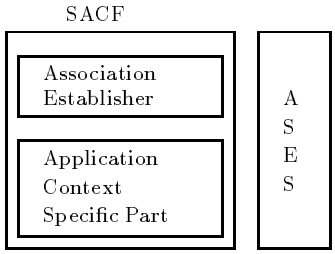


Figure 5: Structure for a SAO

The *Association Establisher* component is the responsible, as its own name indicates, for the establishment of associations. This component is considered to be part of the SACF, because the association establishment procedure can require interactions among ASEs and the control over resources provided by the lower layers, what are SACF functions. At association establishment phase, the specific application context to be used on the association is negotiated. One of the functions of the *Association Establisher* component is to execute this negotiation, and therefore it must know which contexts the AEI supports.

The *Application Context Specific Part* represents the functions to control the interactions among the ASEs and the use of the resources provided by the lower layers, according to the specific application context negotiated.

In an implementation it is very useful to really consider this function of SACFs. As it represents the realization of the rules of an application context one can consider to have separate implementations of *Application Context Specific Parts*, one for each application context supported by the system. An implementation of one of these components would contain the code to carry out the functions associated with this component that are related to the specific application context it represents. After having negotiated the application context to be used on an association, the *Association Establisher* activates the appropriate *Application Context Specific Part*.

This implementation structure suggests therefore that the implementation of a given ASE should be done considering the activities of the protocol that are dependent on application contexts and those that are not. By letting the SACF do all the application context dependent activities (in its *Application Context Specific Part*) the ASEs implementations become very modular, being able to be used on many application contexts in a transparent way.

For the CCR protocol implementation, for example, the CCR ASE does just APDU and primitive generation, caused respectively by the issue of primitives by the user of the ASE and by the arrival of APDUs from the remote system, and the control of events sequencing rules. The control over the sequential numbers and token possession is done by the SACFs, because it depends on the use of these resources by other ASEs that can take part in the application context. This control is then dependent on the application context used.

2.5 Structure for the Application Context Specific Part of SACFs

To provide a scheme of the functions realized by the *Application Context Specific Part* of SACFs, the decomposition showed on figure 6 was done. The *Info* component represents informations necessary for the control of the SAO or arisen from this control. Informations of such kind may be, for example:

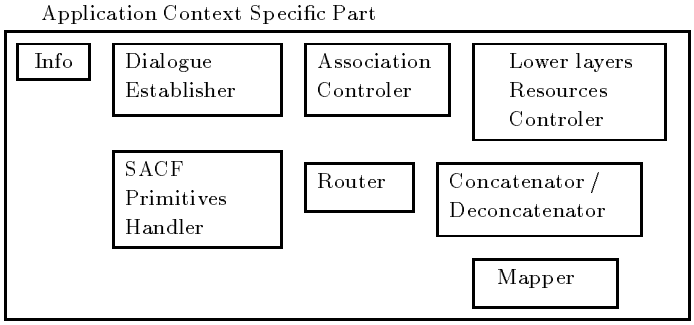


Figure 6: Structure for the Application Context Specific Part of SACFs

- state information about the global activity executed in the SAO. The TP specifies states for the SAO in order to detect protocol errors. The SAO is, for example, in the *Free* state when the association is just established or when it is not in use; the *Busy* state represents the period between the dialogue establishment and the liberation of the SAO by the MACF (use of the SAF-DETACH-ASSOCIATION.req primitive); and there are other states that define more specific phases as *Bidding*, that is the state of a SAO when it is processing the *Bid Mechanism* (see below) or the *Stray* state, that defines the period during dialogue establishment when there is no *Bidding*;

- information about token possession, parameters of primitives previously sent or received, the current sequential number for the synchronization numbers, among others, that influence the SACF operation or are used to fill in parameters of Presentation services primitives.

The *Association Controler* represents functions that analyse each event that takes place in the SAO and control the necessary actions to be performed. For example, controls the sequencing rules of events considering all the ASEs present in the SAO.

The *Dialogue Establisher* represents functions for the establishment of a *dialogue*. A dialogue is a relationship that is created over an association. TP specifies two such dialogues: dialogues for the execution of a transaction branch, called simply *dialogues*, and *channels*, that are dialogues for doing recovery from communication and application faults. The *Dialogue Establisher* component controls the *Bid Mechanism* when it is used to establish a dialogue. This mechanism is used to prevent the occurence of collision at dialogue establishment time.

The *SACF Primitives Handler* treats the primitive SAF-DETACH-ASSOCIATION.req, defined above. This primitive has an argument whose value determines certain events that can still happen on the association (arrival of APDUs) but that are not to be considered as errors by the SACF. The SACF controls the association while not in use by a dialogue or channel.

By the interaction scheme among the components of the SAO shown on figure 3, the APDUs from the ASEs (except ACSE APDUs) are sent to the SACF for transmission. During data receptions, conversely, the APDUs are sent by the SACF to the ASEs. This scheme results from the fact that the TP application contexts allow for concatenation of APDUs, so that these APDUs can be sent together as user data of just one Presentation service primitive. A concatenation can involve APDUs from just one ASE but also APDUs from diferent ones. The rules specifying the allowed concatenations are specified individually by each ASE, but there are also rules specified by the application context. These

application context rules extend the ones from each ASE. The *Concatenator/Deconcatenator* component represents these functions. All the ASEs send their APDUs to the SACF. The SACF can then concatenate the APDUs according to the rules of the application context being used.

The mapping of the APDUs, concatenated or not, into the user data parameter of Presentation layer service primitives to be sent to the remote system also depends on the specific application context being used. The specification of ASEs determines the primitives that must be used to transfer their APDUs, but the concatenation process can alter the choice of the Presentation primitive. The *Mapper* component represents the functions to choose the appropriate primitive, according to the concatenations done (if any) and the application context used. This component also fills in the other parameters of the Presentation primitive. The control over lower layers resources are necessary to it.

The *Router* component represents the functions of determining the right component that must treat a CCR or ACSE primitive generated by the receipt of an APDU from the remote system. This component sends the primitive to the TP-ASE or to the MACF, depending respectively on if the primitive contains TP APDUs as user data or not.

The *Lower Layers Resources Controler* represents the functions to control aspects of use of lower layer resources as tokens and synchronization points. Such resources are provided by the lower layers through Presentation layer services but the semantics attributed to them and the control of their use are determined by the Application layer. For example the Session layer provides mechanisms for setting synchronization points in the dialogue but is not concerned with the state information associated with each point. It is Application layer responsibility to maintain the necessary state information for doing resynchronization. For these points the Session layer services just provide means for setting them and to indicate resynchronizations.

Because more than one ASE can make use of such resources, the SACF must control the use of the primitives that affect them. Therefore, for example, the control over token possession and the sequential number for the synchronization points are done by the SACF, determining then

the changes on the values of the variables specified in [6] (variable V(A), V(M), V(R) and Vsc) for the control of these sequential numbers and the issue of service primitives related to these synchronization points. Doing the implementation this way results in the ASEs doing nothing about this control.

2.6 Structure for AEIs

An AEI for atomic transactions support has the structure depicted on figure 7. On this figure a set of MACFs is depicted, each one controling a set of SAOs. There is also another set of SAOs under the control of a component called *Free Associations Controler*.

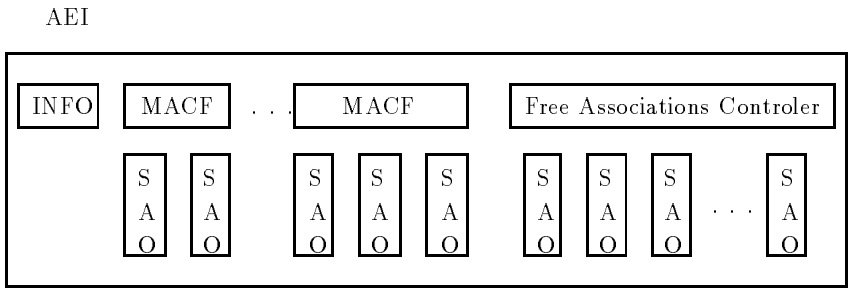


Figure 7: Structure for AEIs

The latter set of SAOs represents associations that are established, but that are not being used at the moment. It forms a pool of available associations. The existence of such associations is useful for efficiency reasons, because associations already established can be reused. Instead of releasing an association, the AEIs can keep this association established for use in the future. For example, it would be useful to maintain associations with a system if this system participates frequently in transactions that use the same application context. This happens, for example, in a bank environment, where customers make many similar transactions

(to take money out from accounts at the same bank agency, for example) at a bank terminal daily. The associations would not have to be reestablished at each time of use.

The *Free Associations Controler* represents functions to grant solicitations of associations to support new dialogues made by the MACFs and to control the set of associations when they are not being used. It represents also the functions to attach a SAO from this set to a MACF, when the dialogue establishment is requested by the remote system. The *Free Associations Controler* is not considered here to be a MACF, because the associations under its control are inactive, and, therefore, can not have activities related that must be controled. It represents just a local management function.

In the document that defines the TP protocol [18] a *Channel Protocol Machine* (*CPM*) is specified. This protocol machine is considered as being a MACF that controls the establishment and release of *channels*, used to recover dialogues in the event of faults. The functionalities of this machine are however modelled here by the *Error Control* component from the MACF, by components of the *Application Context Specific Part* from SACFs and by the *Free Associations Controler*.

2.7 Structure for AEs

In order to consider addressing aspects, an AE was structured in the way shown on figure 8. Associated with the invocations of an AE appears a module called *AEI Selector*. This module carries out functions to determine an AEI to treat an association establishment request made by a remote system. During an association establishment, the presentation address of the AE with which the association is to be established must be necessarily provided by the system that requests the establishment. This address does not determine, however, an AE invocation. The association establishment requestor can, optionally, provide informations about a specific AEI or API with which the association is to be established. If so the *AEI Selector* module directs the establishment request for this AEI. If they are not provided, this module is responsible for determining an

adequate AEI to treat the request (or to request the creation of a new AEI). This determination is done based on local system decisions. If the request can not be accepted at this level of AEI selection this module answers negatively to the remote system. Note that this module interacts with the ACSE, because the informations necessary for the selection of an AEI (informations about APs, AEs, AEIs and APIs) are transmited on APDUs of this protocol. The ACSE ASE receives these APDUs and issues primitives to the *AEI Selector* module.

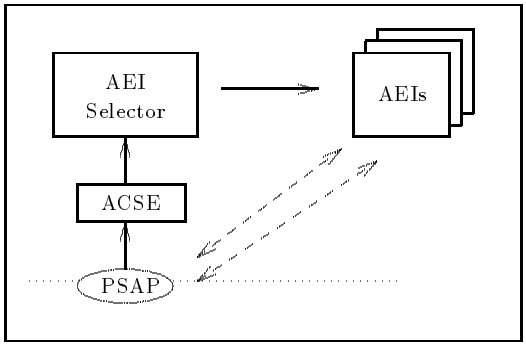


Figure 8: Structure for AEs

Since potentially many application protocols can participate in TP application contexts, this implementation structure should be followed by all application protocols. It is modular, because the protocol activities dependent on application contexts are separated from the ASE implementation, resulting in the ASEs being implemented in a way that they can be used on many application contexts without modifications. All the application context dependent aspects are isolated in SACFs. This structure is not directly derived from implementations of other application contexts, but it is useful to structure even them according to the ideas above.

3 Implementation Aspects: the SISDI-OSI

An RM-OSI upper layers implementation is being developed at UNICAMP (Universidade Estadual de Campinas). The objectives of this implementation is to have a didactic communication environment to be used in computer network courses, to serve as a basis to gain experience in protocol implementation issues and to build a platform for the development of certain types of applications as, for example, *ODP* (*Open Distributed Processing*) [27] systems.

This system is called SISDI-OSI (*Sistema Didático para o Modelo OSI – Didactic System for OSI Model*) [21] (figure 9) and its project includes the implementation of all the Presentation and Session layers functional units, an interface that maps the Transport Class 4 services onto TCP/IP, and, at the Application layer, the following protocols: ACSE, CCR, ROSE, TP, RDA, DS (*Directory Services*), NM (*Network Management*) and MMS (*Manufacturing Message Specification*). Although it does not appear in the RM-OSI/ISO, SISDI-OSI includes also an implementation of the MMS *Application Program Interface*, as defined by the MAP/TOP project [20]. So that the user of SISDI-OSI can access the system, it is defined an interface that allows the user to submit APs or individual services and to observe the effects on the system in all the layers. An ASN.1 compiler [8] [24] was developed that, from ASN.1 descriptions of PDUs[2], generates data structures in the C language that can be used in the protocols implementations. This compiler also generates coding and decoding rotines to be used at the Presentation layer. SISDI-OSI is a second version of a system called SISDI-MAP (*Sistema Didático do Protocolo e da Interface de Aplicação MMS do MAP – Didactic System of the MMS Protocol and Application Interface from MAP*) [23]. SISDI-MAP had been implemented in IBM-PC compatible systems, under DOS operating system, using a real time kernel to provide a multitasking environment. SISDI-OSI is being reprojected, now for the UNIX environ-

[2] A *PDU* (*Protocolo Data Unit*) is a data unit exchanged between entities from the Presentation layer or from the lower layers. This term can also be used, as it is the case, to refer to PDUs or APDUs generically

ment and it is being developed at SUN workstations. SISDI-MAP had just a very small part of SISDI-OSI.

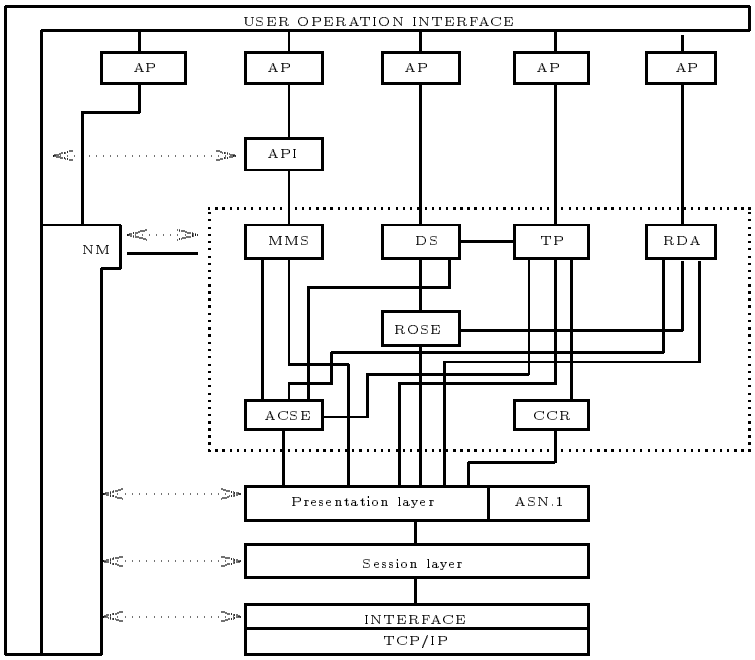


Figure 9: SISDI-OSI

3.1 General Implementation Scheme

Because it is a didactic system, SISDI-OSI is structured more based on simplicity aspects than on efficiency issues. The scheme for the implementation of protocols (ASEs, MACFs, SACFs and Presentation and Session protocols) for this system is shown on figure 10. Each protocol is implemented as an independent UNIX process, that simulates its instances of use internally (for example, by using tables). To receive

messages, that can be service primitives, APDUs or control primitives, each process has just one queue, whose address is known by the other processes which must send messages to it. The messages can be selected by type, although this functionality is currently not used. Each process is also attached to a memory area that is shared by all the protocols. This area contains a buffer used to store the contents of primitives and APDUs.

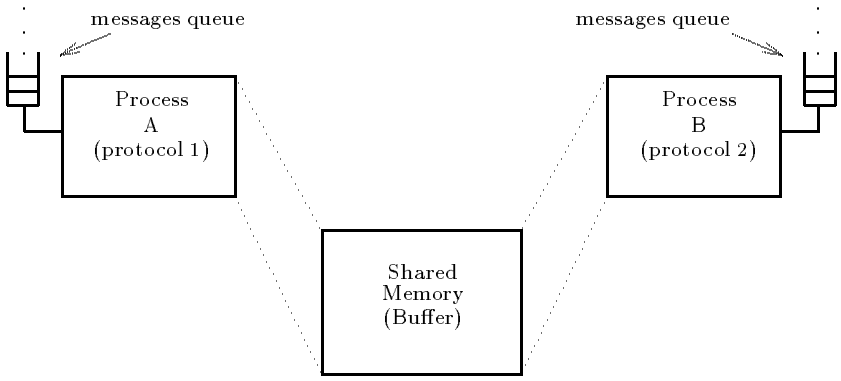


Figure 10: Structure for protocol implementation

The queues and shared memory area were implemented using mechanisms for interprocess communication from the UNIX System V, respectively *Messages* and *Shared Memory* [2, 26]. When a process must send a message to another process, it allocates an area in the buffer, writes the contents of the message in this area and puts a message in the target process queue. The structure of this message, exemplified by the case of a CCR C-COMMIT.req primitive issue, is shown on figure 11. The message contains a field to identify the protocol instance to which the message is related (*invoc_id* field); a field to indicate the protocol to which the message relates (*component* field – in this example, because it is a CCR primitive, this field has the value *CCR*); a field to identify if the

message contains PDU or primitive data (*info_type* field); and a pointer to the area at the buffer where the data are really stored (*info_ptr* field). The primitive parameters are stored in a region of the buffer and the first field of this region is an integer value that determines the type of the primitive (or APDU).

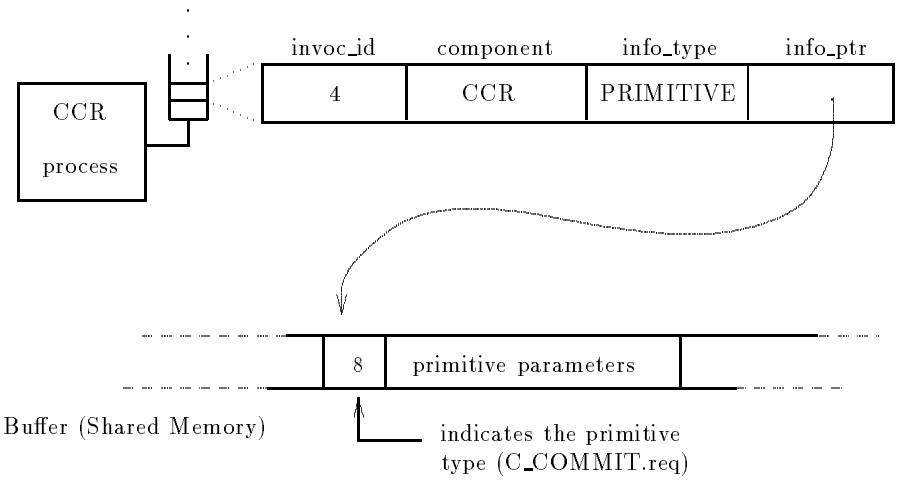


Figure 11: Structure of the messages that are put into process queues

The simulation of instances of use of each protocol is done according to the language used for the implementation. The ACSE ASE, for example, was implemented in C. The simulation of the instances was done then using internal tables, that maintain the state for each individual instance of use. The CCR implementation, by its turn, was done using ESTELLE [10], that is a Formal Description Technique defined by ISO, and a compiler that translates an ESTELLE specification into C code [1]. The simulation of the instances was done then by using ESTELLE module instances. In every case, however, the instances are simulated internally in an UNIX process and the messages that arrive at the queue are treated according to the *invoc_id* field, shown on figure 11.

The *Semaphore* mechanism from UNIX System V [2] is also used, to control concurrency in accessing the buffer area. The pattern of use of the buffer by the protocols from the RM-OSI model implies that concurrency control is only necessary during allocation and deallocation of buffer areas, because these procedures change buffer management data. After having been allocated in the buffer an area will not be read or written concurrently, but by one protocol at a time. This means that it is not necessary to control concurrency during reading or writing of allocated data.

When a message is sent to a protocol the sending process must specify the invocation identifier of the recipient process that the message relates to. Inside a SAO a given invocation identifier is used on all the SAO components to identify the same association. By the same way the presentation and the session connection that supports an association have all the same invocation identifier. Inside a SAO and at the Presentation and Session layers therefore a process that receives a message with a given invocation identifier just send any APDU or message with the same invocation identifier.

The MACF however must control the invocation identifiers relating to the associations it controls. An appropriate project for an *Application Program Interface* [19] for TP application contexts should consider this control. Note that the notion of *Application Program Interfaces* does not exist at the RM-OSI, but it is encouraged here. The system must contain a mechanism to allow the reutilization of invocation identifiers.

3.2 Message Structures

The message structures for the Application and Presentation layers were developed in such a way to be according to the data structures generated by the ASN.1 compiler used in the system [24]. The C structures generated by the compiler define the way the data must be structured so that the rotines are able to code the outgoing data into *BER* (*Basic Encoding Rules*) [9]. BER are the codification rules to be used for all APDUs and PDUs from the Presentation layer in SISDI-OSI. For the

outgoing data the coding rotines encode the data from the C data structures into BER. For incoming data the decoding rotines process the data in BER and transforms them into the C data structures.

The C data structures generated by the processing of the definitions of an application protocol APDUs can be included in the code for this protocol, giving already the declaration of such APDUs. As many primitive parameters are directly mapped on APDU fields and vice-versa, the C structures generated by the ASN.1 compiler are also used to structure primitive parameters. The primitive parameters have the same type declaration as the APDU fields into which they are mapped.

Primitive parameters that are mapped into presentation PDU fields are processed in the same way, because the Presentation layer PDUs are also processed by the ASN.1 compiler, since the encoding rules defined for this layer is also BER [3]. Primitive parameters that are mapped into Session PDU fields or that are not mapped into any PDU field are defined by the protocol implementor.

The structure for primitives and PDUs was done in such a way to follow exactly the data structures generated by the ASN.1 compiler and to avoid data copying. The informations that go in the messages that are put into the queues of the processes are just data to identify the type of message and the address where the contents of the message are (see figure 11). The major part of the message, that is its contents, is put into the buffer in the shared memory. The protocols that must access them, do it just by manipulating pointers.

The use of pointers provides a very flexible way to perform APDU concatenations. The SACF identifies the ASE of each APDU and fills in fields at the user data parameter of Presentation service primitives. These fields allow the SACF to specify the presentation context to be used to codify each APDU. This constitutes one more SACF function that hides this control from the ASEs. A detailed explanation of the data structures used appear in [25].

The coding rotines at the Presentation layer receives all the APDUs to be coded in a tree structure together with a Presentation layer PDU. The compiler then allocates an appropriate area in the buffer to put the

result of the codification. If the area is not big enough another one is allocated and both are linked by using a pointer. The coded data at the lower layes are treated as a linked list of the buffer areas that contain them.

An interesting aspect of the data structure used is that when the primitives, APDUs and Presentation layer PDUs are stored in an area at the buffer, the first field of this area is an integer that identifies the type of the primitive or PDU (see figure 11). The remainder part contains the primitive parameters or the PDU fields. By using pointers, when a primitive and PDU have correspondent parameters and fields, the transformation of one into the other is made just by changing the value of this integer field. This avoids data copying.

The definition of types for parameters of primitives from the Session layer and for the transport primitives implemented (as mentioned above, just an interface to the TCP protocol was done) is also done in a way to facilitate the conversion into PDU fields, but the ASN.1 compiler can not be used, since the codification rules used in these layers are not BER.

3.3 Synchronization Assumption

A useful assumption that was made for the implementation of application contexts is to synchronize the processing of all the components of a SAO and the Presentation and Session layers. This synchronization is intended to not permit parallel execution of these components for the processing of events related to an association and the supporting presentation and session connections. Parallel execution of activities related to different associations and presentation and session connections is however allowed.

If synchronization is not supposed the implementations get much more difficult due to state inconsistencies that can happen among the components. One such situation occurs for example when sending the CCR APDU C-ROLLBACK-RI [12]. This APDU is mapped in the user data parameter of P-RESYNCHRONIZE primitives. The following scenario can happen during a collision of P-RESYNCHRONIZE

primitives (figure 12). On figure 12a the C-ROLLBACK-RI is sent to the SACF for transmission and the Presentation layer sends a primitive P-RESYNCHRONIZE.ind to the SACF, containing, as user data, another C-ROLLBACK-RI. The SACF treats first the APDU that comes from the CCR ASE and sends this APDU as user data of a P-RESYNCHRONIZE.req primitive (figure 12b). A collision then occurs at the Presentation layer (figure 12c). This collision is resolved by Session layer rules and depends on the value of the synchronization point number, that is a parameter of the P-RESYNCHRONIZE service primitives, and on which AEI has requested the establishment of the association [7]. The SACF then would have also to determine the result of the collision, in order to consider the primitive P-RESYNCHRONIZE.ind, and to decide either to send the APDU to the CCR, or to descard it. The implementation of the SACF then gets more complicated and there is duplication of activities. With the assumption of synchronicity the outgoing and the incoming events would happen serially, keeping then state consistency among the SACF and the Presentation and Session layers.

This synchronization assumption can be implemented using a queue to store messages to the SACF and to the session entity. This queue should be managed by a component that would control the sending of messages to these components depending on the use of the association and its supporting presentation and session connections. Although this synchronization assumption implies mechanisms to indicate when a new message from this queue can be sent to the SACF or session entity, it is being considered that this mechanism is easier to be done than to control problems analogous to the one described above. The synchorization assumption is also useful because the implementations become clearer and then more probable of being correct.

The lack of synchronization between an ASE and its user may cause alterations to be done in the protocol state table. The ASE can, for example, treat an user primitive just after treating an APDU received from the remote system. If the issue of the primitive was correct if the APDU was not arrived but incorrect after it, the user would not have a way to know if it can or not issue the primitive. The synchronization

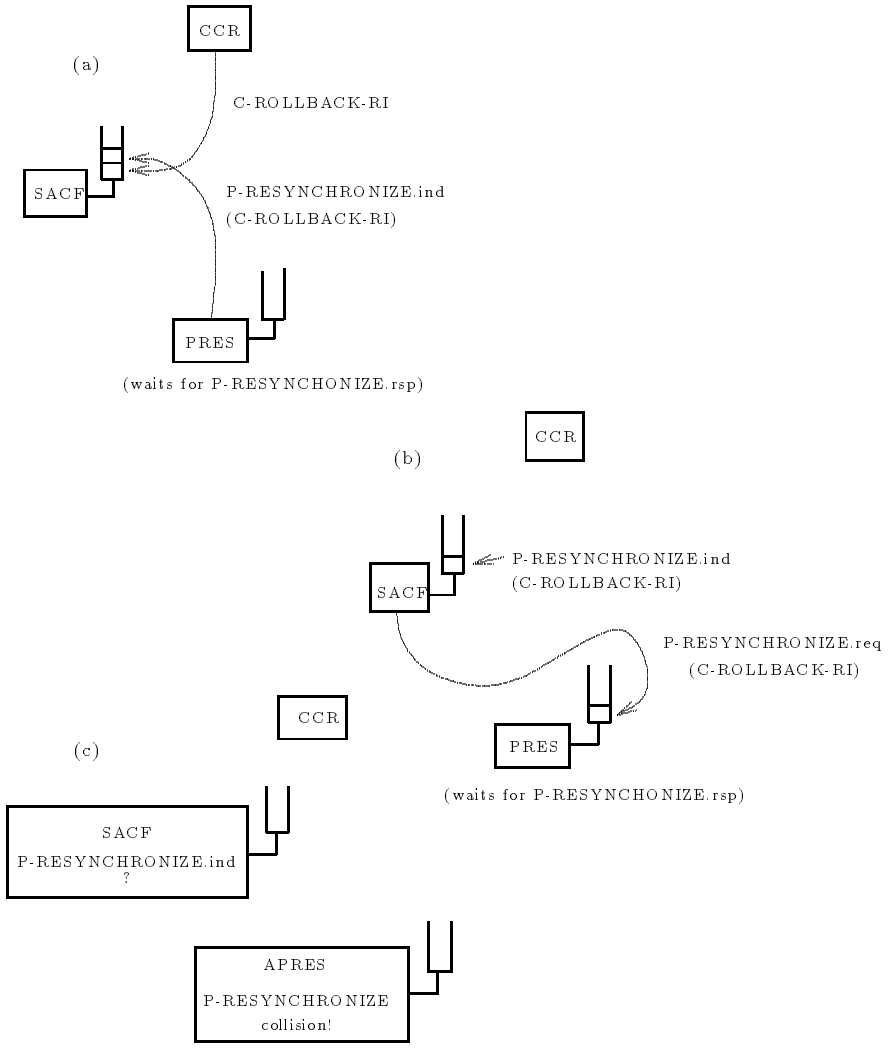


Figure 12: Example of a collision caused by lack of synchronization between components

assumption eliminates this problem for the ASEs whose service user is inside a SAO. If the user can not (or should not) be synchronized with the ASE then the problem continues and perhaps modifications to the state table must be done. [25] describes the modifications done for the CCR state table for the cases where a synchronization could not be assumed.

3.4 SAO Mounting

The implementation structure suggested on this technical report tries to make the implementation of the Application layer components as much modular as it is possible. After the *Association Establisher* component establishes the association and determines the application context to be used on the association, it activates an appropriate *Application Context Specific Part*. This component contains all the informations about the application context negotiated and processes the SAO mounting.

In this process the *Application Context Specific Part* component sends to each ASE that takes part in the application context the informations necessary to "configure" the ASE for this application context. The informations are, for example, the queue identifier of the process to which the ASE must send its primitives, the queue identifier of the *Application Context Specific Part* in order to the ASE to send its APDUs and some specific informations dependent on each ASE. For the CCR, for example, it is sent the title of the remote AE. This information is necessary for the issue of one of its primitives (C-BEGIN.ind) [11, 12].

4 Conclusion

The objectives of this technical report are to try to give a structure for application contexts that contain the TP protocol so that they can be better understood and to show the general structure for the implementation of these application contexts in a didactic communication system called SISDI-OSI.

The definition of functionalities to each component was done considering their precise definitions in order to obtain a very modular structure

for the implementations. The presented structure suggests to isolate the application context dependent activities of application protocols in SACFs (in the *Application Context Specific Part*). In this way the implementation of the ASEs can be just "configured" and then can be used on any application context that defines this ASE as a participant. By using these configurations, the structure then is also useful for the implementation of other application contexts. This would make a generic structure for the implementation of a very complex Application layer and seems to bring great flexibility to the implementation.

The use of the C data structures generated by the ASN.1 compiler has been very practical in the implementation of the protocols and gives a very flexible way to treat concatenations of APDUs. The fact that the coding rotines receives the APDUs and Presentation PDUs to codify at once tends to make the data structures easier to be handled by the protocols than schemes in which each APDU is codified isolately, by the own ASE calling coding rotines, as the scheme described in [4].

The synchronization assumption for the execution of SAO components and Presentation and Session layer entities makes the implementations easier to be done and then more probable of being right. This assumption however requires new functionalities to control the use of an association and its supporting presentation and session connections. In any case it seems that these new functionalities are easier to be realized than having to cope with the problems generated by the lack of synchronism.

The implementation of the CCR protocol was done using the TDF ESTELLE. This generated a specification of the relationship of this protocol with other Application layer components that is according to the implementation structure here defined. This specification can also be used to complement the TP ESTELLE specification present in [18] and so allows for the study of a semi-automatic implementation of these protocols.

The SISDI-OSI is however under development and much work is still necessary to be done. Particularly the TP implementation is not complete. Other implementations for sure will bring more informations about

this implementation structure and its applicability to other application contexts, not only TP application contexts.

References

[1] *EWS User's Manual – Esprit Project 265 – SEDOS Estelle Demonstrator*, June 1989.

[2] Maurice J. Bach. *The Design of the UNIX Operating System.* Prentice-Hall Software Series. Prentice-Hall, Inc., EUA, 1986.

[3] CCITT. Presentation protocol specification for open systems interconnection for CCITT applications – recommendation X.226, 1988.

[4] Sanjay B. Chikarmane. Upper layer architecture for HP MAP 3.0 OSI services. *Hewlett-Packard Journal*, pages 11–14, August 1990.

[5] ISO. Information processing systems – open systems interconnection – basic reference model – ISO 7498, 1984.

[6] ISO. Information processing systems – open systems interconnection – basic connection oriented session service definition – ISO 8326, August 1987.

[7] ISO. Information processing systems – open systems interconnection – basic connection oriented session protocol specification – ISO 8327, August 1987.

[8] ISO. Information processing systems – open systems interconnection – specification of abstract syntax notation one (ASN.1) – ISO 8824, December 1987.

[9] ISO. Information processing systems – open systems interconnection – specification of basic encoding rules for abstract syntax notation one (ASN.1) – ISO 8825, November 1987.

[10] ISO. Information processing systems – open systems interconnection – estelle - a formal description technique based on an extended state transition model – ISO/IEC ISO 9074, November 1988.

[11] ISO. CCR service definition – ISO 9804 – final text, February 1990.

[12] ISO. Information technology – open systems interconnection – protocol specification for the commitment, concurrency and recovery service element – ISO/IEC 9805, April 1990.

[13] ISO. Information processing systems – open systems interconnection – application layer structure – ISO/IEC DIS 9545, 1988.

[14] ISO. Information processing systems – open systems interconnection – remote database access – part 1 : Generic model, service and protocol, May 1991.

[15] ISO. IPS – OSI – remote database access – part 2: SQL specialization, February 1990.

[16] ISO. Information technology – open systems interconnection – distributed transaction processing – part 1: OSI TP model – ISO/IEC 10026–1, May 1991.

[17] ISO. Information technology – open systems interconnection – distributed transaction processing – part 2: OSI TP service – ISO/IEC DIS 10026–2, May 1991.

[18] ISO. Information technology – open systems interconnection – distributed transaction processing – part 3: Protocol specification – ISO/IEC 10026–3.2, October 1991.

[19] Edmundo Roberto Mauro Madeira and Manuel de Jesus Mendes. An application interface model for communication software. In *IEEE Global Telecommunications Conference – GLOBECOM'90*, San Diego, EUA, December 1990.

[20] Manuel J. Mendes. *Redes Locais Industriais*. UNICAMP/CONSAI, 1990.

[21] Manuel de Jesus Mendes, Edmundo Roberto Mauro Madeira, Flávio Morais de Assis Silva, Elza Kiyomi Shimabukuro Garcia, Lisiane Maria Bannwart Ambiel, Luiz Otávio Merlin Miranda, María Inés Valderrama Restovíc, Milton T. Sakamoto, and Célio Toshihiro Fujito. SISDI-OSI: Sistema didático para o modelo OSI. In *11o. Simpósio Brasileiro de Redes de Computadores*, UNICAMP, Campinas, SP, May 1993.

[22] C. Mohan and B. Lindsay. Efficient commit protocols for the tree of processes model os distributed transactions. Technical report, IBM Research Laboratory, San Jose, EUA, 1983.

[23] Antenor Paglioni Junior, Durval Carvalho Ávila Jacintho, Edmundo Roberto Mauro Madeira, Ivo Alexandre Fernandes, Jaime Nicolato Correa, José Mario Souza Lima, Maria Cristina Zabeu, Verônica Lima Pimentel de Sousa, and Manuel de Jesus Mendes. SISDI-MAP: Sistema didático do protocolo e da interface de aplicação MMS do MAP. In *Seminário Franco-Brasileiro em Sistemas Informáticos Distribuídos*, Florianópolis, SC, September 1989.

[24] María Inés Valderrama Restovíc. *Compilador ASN.1 e Codificador/Decodificador BER*. Master thesis, UNICAMP, September 1992.

[25] Flávio Morais de Assis Silva. *Um Refinamento da Estrutura da Camada de Aplicação do RM-OSI/ISO e Aspectos de sua Implementação em um Sistema Didático de Comunicação*. Master thesis, UNICAMP, May 1993.

[26] SUN Microsystems, USA. *Programming Utilities and Libraries*, 1990. Revision A of 27 March.

[27] Volker Tschammer, Manuel de Jesus Mendes, Wanderley Lopes de Souza, Edmundo Roberto Mauro Madeira, and Waldomiro P.

de Loyolla. Processamento distribuído aberto e o modelo rm-odp/iso. In *11o. Simpósio Brasileiro de Redes de Computadores*, UNICAMP, Campinas, SP, May 1993.

Relatórios Técnicos – 1992

01/92 **Applications of Finite Automata Representing Large Vocabularies,** *C. L. Lucchesi, T. Kowaltowski*

02/92 **Point Set Pattern Matching in d-Dimensions,** *P. J. de Rezende, D. T. Lee*

03/92 **On the Irrelevance of Edge Orientations on the Acyclic Directed Two Disjoint Paths Problem,** *C. L. Lucchesi, M. C. M. T. Giglio*

04/92 **A Note on Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams,** *W. Jacometti*

05/92 **An (l, u)-Transversal Theorem for Bipartite Graphs,** *C. L. Lucchesi, D. H. Younger*

06/92 **Implementing Integrity Control in Active Databases,** *C. B. Medeiros, M. J. Andrade*

07/92 **New Experimental Results For Bipartite Matching,** *J. C. Setubal*

08/92 **Maintaining Integrity Constraints across Versions in a Database,** *C. B. Medeiros, G. Jomier, W. Cellary*

09/92 **On Clique-Complete Graphs,** *C. L. Lucchesi, C. P. Mello, J. L. Szwarcfiter*

10/92 **Examples of Informal but Rigorous Correctness Proofs for Tree Traversing Algorithms,** *T. Kowaltowski*

11/92 **Debugging Aids for Statechart-Based Systems,** *V. G. S. Elias, H. Liesenberg*

12/92 **Browsing and Querying in Object-Oriented Databases,** *J. L. de Oliveira, R. de O. Anido*

Relatórios Técnicos – 1993

01/93 **Transforming Statecharts into Reactive Systems,** *Antonio G. Figueiredo Filho, Hans K. E. Liesenberg*

02/93 **The Hierarchical Ring Protocol: An Efficient Scheme for Reading Replicated Data,** *Nabor das C. Mendonça, Ricardo de O. Anido*

03/93 **Matching Algorithms for Bipartite Graphs,** *Herbert A. Baier Saip, Cláudio L. Lucchesi*

04/93 **A lexBFS Algorithm for Proper Interval Graph Recognition,** *Celina M. H. de Figueiredo, João Meidanis, Célia P. de Mello*

05/93 **Sistema Gerenciador de Processamento Cooperativo,** *Ivonne. M. Carrazana, Nelson. C. Machado, Célio. C. Guimarães*

06/93 **Implementação de um Banco de Dados Relacional Dotado de uma Interface Cooperativa,** *Nascif A. Abousalh Neto, Ariadne M. B. R. Carvalho*

07/93 **Estadogramas no Desenvolvimento de Interfaces,** *Fábio N. de Lucena, Hans K. E. Liesenberg*

08/93 **Introspection and Projection in Reasoning about Other Agents,** *Jacques Wainer*

09/93 **Codificação de Seqüências de Imagens com Quantização Vetorial,** *Carlos Antonio Reinaldo Costa, Paulo Lício de Geus*

10/93 **Minimização do Consumo de Energia em um Sistema para Aquisição de Dados Controlado por Microcomputador,** *Paulo Cesar Centoducatte, Nelson Castro Machado*

Departamento de Ciência da Computação — IMECC
Caixa Postal 6065
Universidade Estadual de Campinas
13081-970 – Campinas – SP
BRASIL

reltec@dcc.unicamp.br