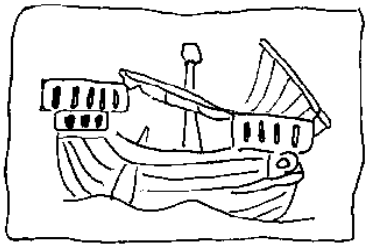
ARCA

Libraries Programme Project LIB-ARCA/2-3039



Title : ARCA Architectural Design Document

Document Reference : ARCA/T12/ADD

Version : Version 2.0

Date : November, 1995

Author(s) : Maria Bruna Baldacci (CNR-IEI)
Donatella Castelli (CNR-IEI)
Alberto Catoni (CNR-CNUCE)
John Favaro (Intecs Sistemi))
Mario Loffredo (CNR-CNUCE)
Giuseppe Romano (CNR-CNUCE)
Oreste Signore (CNR-CNUCE)
Itziar Lopez de Sosoaga (SABINI)

Distribution : CNR-CNUCE/IEI
Fundacion Sancho El Sabio
Intecs Sistemi
Regione Toscana
SABINI
Università di Pisa

Abstract : This report presents the work done during the specification of the ARCA SR Targer system architecture. It improves the requirements for this system and describes the architectural design of the ARCA Target using the Object Modelling Technique approach to software development.

Keywords	:	SR, Z39.50, OMT, OPAC, EXPLAIN, ARCA Target Sytem
Reviewed by	:	
Approved by	:	

Document Status Sheet

Issue	Changes	Date	Reason
0.0	New document	10 August 1995	First draft
0.1	Update	24 August 1995	Rearrangement of previous document by Baldacci, Castelli, Catoni, Favaro, Loffredo, Romano, Signore.
1.0	Update	19 October 1995	Preliminary Draft of the final document
1.1	Update	5 November 1995	After Progress Meeting in Madrid
1.2	Editing	25 November 1995	Preliminary Final version
2.0	Editing	30 November 1995	Final version

Table of Contents

1 Introduction

Task 1.2 of the ARCA Project consists of specifying the architecture of the ARCA SR Target system and the architecture of the interfaces for Sabini and ISIS.

This report presents the work done during the specification of the ARCA SR Target system. In particular it enriches the description of the requirements for this system and describes the architectural design of the ARCA Target using the Object Modelling Technique (OMT) approach to software development [Rumbaugh et al.].

The report is organized in the following sections:

Section 2 enhances both general and functional requirements previously stated.

Section 3 describes the general architecture of the ARCA system.

Section 4 specifies the ARCA system according to OMT models.

Section 5 presents the operations to set up the ARCA system.

Section 6 and 7 describes the interfaces between the ARCA system and, respectively, ISIS and Sabini OPACs.

Appendix A presents the OMT methodology and StP/OMt CASE tool.

Appendix B describes briefly the YAZ toolkit.

2 Enhancing ARCA Target functionalities

Requirements of ARCA Target have been stated in [ARCA/T11/SRD] under two separate headings, i.e. general requirements and functional requirements. General requirements, regarding ARCA Target when communicating with SR Protocol APDUs, can be summarized as follows, with reference to Figure 2.1.

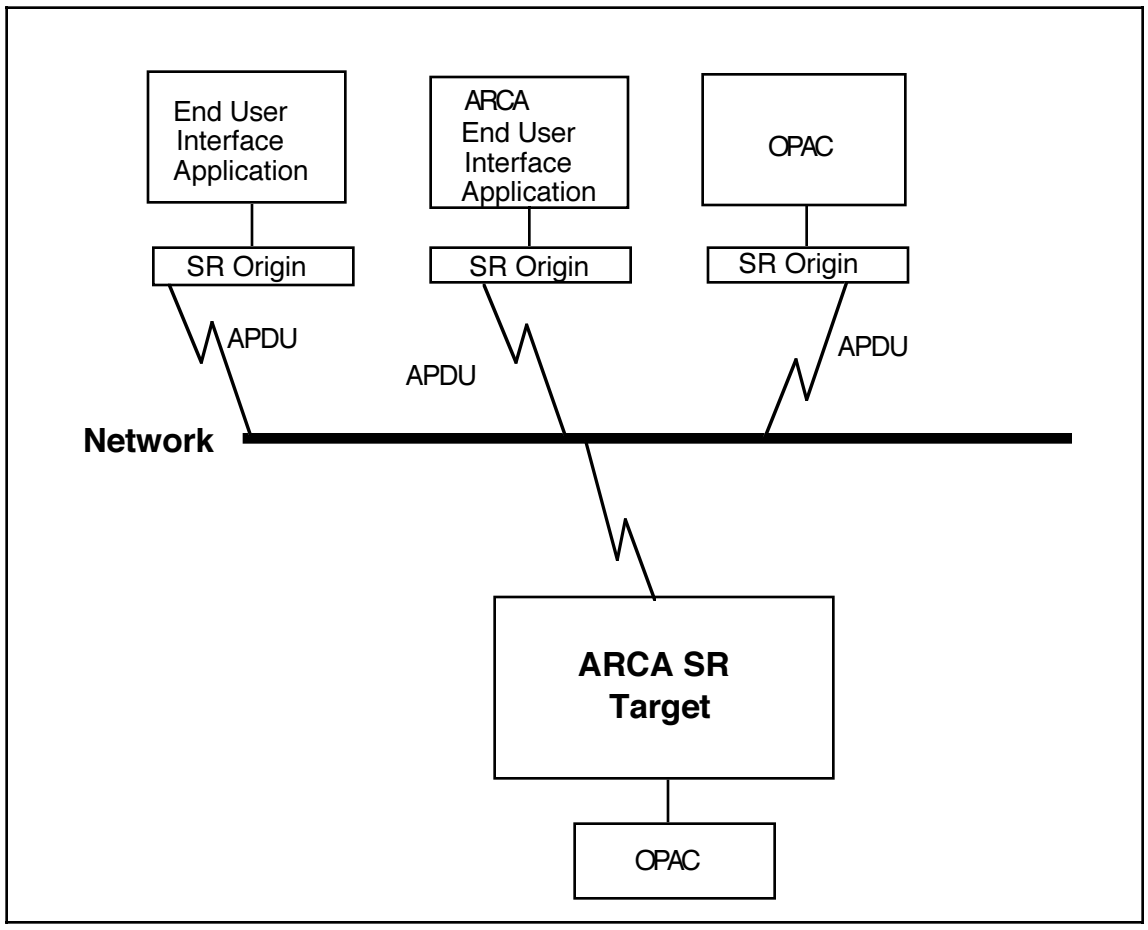


Figure 2.1- An SR Protocol Architecture

The ARCA Target accepts connection from any origin, be the origin functions implemented in an independent end user workstation, or in an OPAC, or activated by the ARCA User Interface Application, described in [ARCA/T22/ADD]. The only constraint is that the incoming request conforms to the ISO 10162/10163 Standard i.e. it must accept the INIT, SEARCH, PRESENT, DELETE, SR-RELEASE and SR-ABORT requests as defined in the SR protocol. To serve these requests the ARCA system must be able to communicate with the faced OPAC.

The requests coming from the ARCA End User Interface should be immediately fulfilled by the ARCA Target as a consistency check should have already been performed at the origin site. This can result in saving of

the processing time, even if some checks can be repeated at the target site as a consequence of some implementation choices.

After discussing the new services defined in the ANSI NISO Z39.50 protocol document, released as preliminary final text in April 1995, ARCA partners agreed upon the immediate implementation of EXPLAIN Service, and possible future implementation of the SCAN Service, being such services already proposed for standardization by the ISO and strongly requested by users.

The decision taken about EXPLAIN forces the updating of both general and functional requirements previously stated. Adjuntive requirements can be grouped under three headings, as follows:

a) openess towards supporting Z39.50 protocol;

b) enhancement of the OPAC basic services

c) communication with an ARCA Origin

2.1. Openess towards supporting Z39.50 protocol

2.1.1 SR protocol extensions and the Z39.50 protocol

As stated in the ARCA/T21/SRD-User Requirements Document, expansions of the SR protocol services are desirable in order to allow users friendly communication. Indeed, the protocol "services to be standardized in the future", as announced in Appendix A of [ISOa] were judged to satisfy most of the requirements stated by the users. In the work done for defining the ARCA system requirements [ARCA T11/SRD] we took into account only the ISO SR documents, waiting for the announced additional functional extensions. Recently, however, the ANSI/NISO Z39.50-1995 is appeared, that is defined as a superset, with bit compatibility, of ISO SR.

Z39.50 -1995 adds several new services and facilities and includes numerous enhancements. New features include:

- The ability to search multiple databases more efficiently by enabling the combination of attributes from different attribute sets within a single query. The revised standard also allows greater flexibility in the definition of attribute sets.

- The ability to request specific portions of a document (such as captions, images, or section headings); to request documents according to specific variants; and to request only the most relevant pertinent portions of a document.

- New services and facilities such as SCAN (used to scan terms in a list or index), sort (for sorting a result set), EXPLAIN (a client can search and retrieve details of the available databases on a server), and

extended services (services that relate to or support information retrieval such as document ordering, request prints of a result set, and defining a periodic query).

- Faster retrieval of a large number of records by allowing a server to respond to a present request with multiple consecutive response messages without intervening requests. Also, better support for large records such as images through new segmentation features.

- Enables a client and server to agree to use a particular language (i.e., English, French, German) and/or character sets during a session.

The services that users are mostly interested in are surely EXPLAIN and SCAN, as also stated in [PARAGON]. The ARCA system architecture has been defined to support the EXPLAIN facility, however it is also open towards the implementation of the other facilities, in particular SCAN.

2.1.2 EXPLAIN

The EXPLAIN facility in the ARCA Target is invoked via a SEARCH operation against a database named "Explain". This kind of search allows an origin to obtain information useful in setting up its SR requests to the target.

The information available in the Explain database can be thought as a *model* of an ARCA Target. This model is logically organized in categories having a well defined structure. In the figure 2.2 a box surrounds a category and the information items associated with it. These items are a subset of the database Explain components as defined within the Z39.50 Protocol with an extra information, "RPN_query_syntax", which specifies the ARCA target query language. The meaning associated with these items can be found in [ANSI].

The queries which appear in a search operation on Explain must satisfy particular restrictions. First, the access points which can be referred in these queries are those items in the figure marked with a star. Second, only particular query structures are allowed. For example, a query on Explain information always must comprise the selection of an appropriate category. So, for example, in order to search for information about a database "A" the query must be the conjunction of "category = DatabaseInfo" and "database_name = A".

The result of a search on Explain is a set of records with the structure defined for the selected category.

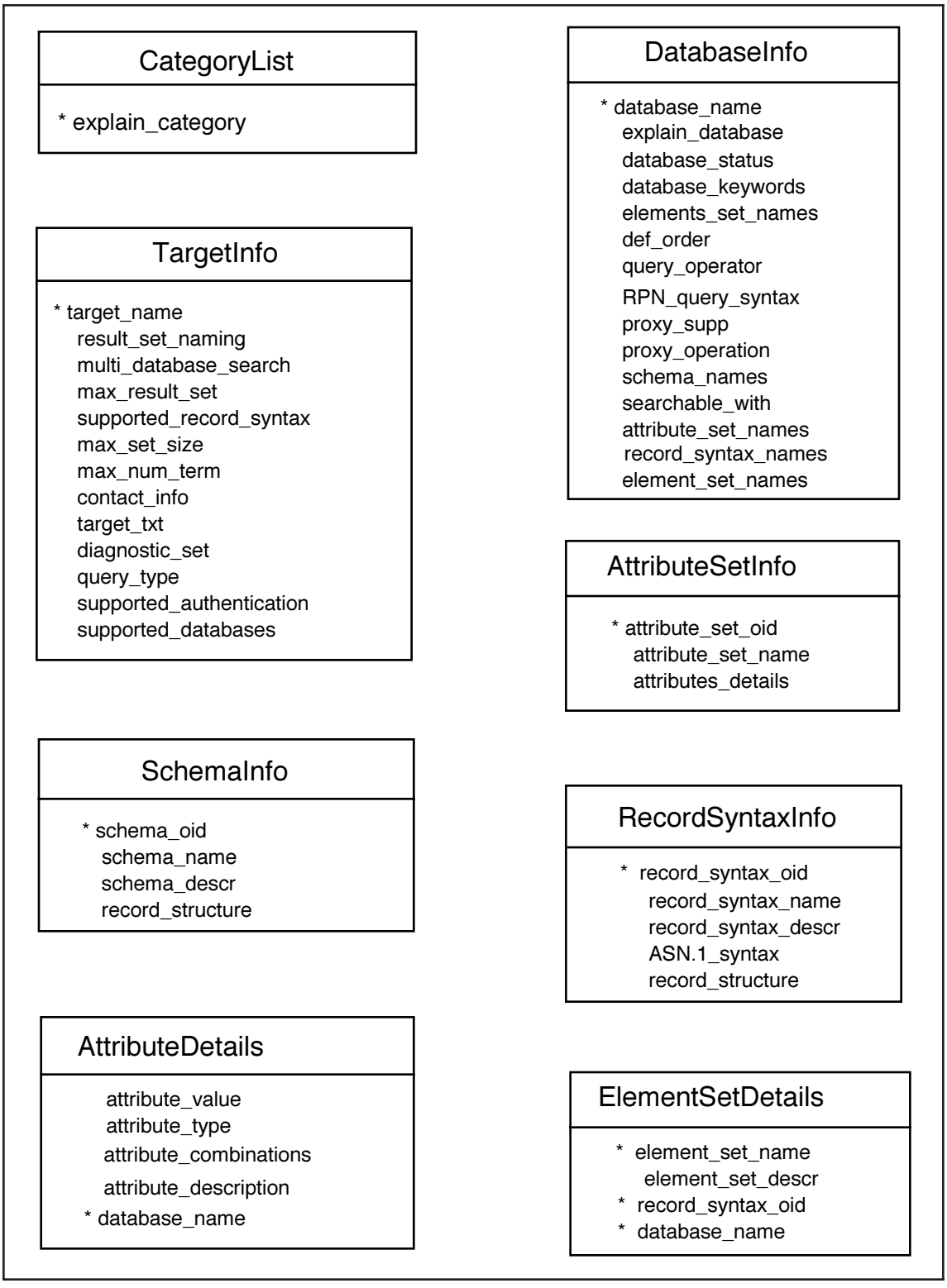


Figure 2.2 - The Explain categories

2.2 Enhancement of the OPAC basic services

The set of functionalities of the ARCA system consists in the union of those provided by the underlying OPAC, which will be called “native”, and those implemented by the ARCA software. In this sense, ARCA aims at enhancing the OPAC basic services.

In addition, the ARCA system gives the chance to implement a set of software modules to fit future requirements. For example, one can implement a module that translates the "native" record format into USMARC, even if the underlying OPAC does not support it.

Once installed, the ARCA system will be able to assure the following functionalities :

- *Naming result set*
  The system will be able to save a result set returned with the specified name. This allows the ARCA target to support the back reference to previously executed queries.

- *Multiple database search*
  A same query could be simultaneously applied to more than one database. If the OPAC does not accept this kind of search, the ARCA target decomposes it in a series of searches on each single database and the returned result sets are seen by the origin as a unique result set.

- *Z39.50 V3 Explain*
  Even if the SR protocol does not yet foresee this functionality, the ARCA system provides it. This enables an ARCA origin to configure itself either to better reflect the target characteristics and to provide a valid databases description to the end-user. ARCA will make available only a part of the Explain informations but the OPAC administrator will be able to easily add further information to fullfil Z39.50 EXPLAIN service.

- *RPN query*
  The OPAC is required to accept at least RPN or infix queries as a string of characters; the ARCA system assures the transformation of the queries formulated in RPN structure into the OPAC proper format.

The whole architecture allows the single OPAC to become an ARCA Target just offering its standard services. Furthermore, it is flexible enough to permit the addition of new features by implementing the appropriate software modules.

2.3 Communication with an ARCA Origin

As noted elsewhere in this document, the possibility is envisioned that there may be a "special relationship" between an ARCA client (origin) and an ARCA server (target). For example, it may be possible for the client to discover even more specific information about the server than is available through the standard EXPLAIN facilities. Or, it may be necessary to communicate in ways that are not specified in the protocol in order to implement some facilities that are available in an ARCA target but not yet

supported by the SR/Z39.50 protocol. This section discusses the options available for communication between an ARCA client and target.

2.3.1 Extensibilty and negotiation in SR/Z39.50 Version 3

In general, we apply the principle of *staying within the standard SR/Z39.50 protocol whenever possible*. We believe that the introduction of proprietary protocol extensions or other nonstandard means of communication represents an outmoded approach that is contrary to the modern movement toward open systems. The SR/Z39.50 protocol now provides numerous facilities to assist clients and targets that have agreements outside of the standard, but nevertheless wish to support these agreements through existing protocol facilities.

Version 3 of the SR/Z39.50 protocol provides powerful mechanisms for extending the protocol. As stated in the standard:

> *Each protocol message includes a field for information whose format is to be defined externally. These externally defined formats will be registered and maintained by the Z39.50 Maintenance Agency, as provisional extensions to the standard, for experimental use and possible consolidation into a subsequent version.*

Specifically, there is a field called "Other-information" or "otherInfo" in each protocol data unit defined in Version 3 of the standard. It is an array of Externals, which may be freely used by implementors to add whatever extensions to the protocol are desired. Communicating client and server partners can browse this list of externals, acting on those extensions it recognizes, ignoring the rest silently.

This otherInfo field will be the major mechanism for exchanging any ARCA-specific data. Since it is present in all protocol data units (e.g. Search, Present, etc.) it will be used in different protocol data units as appropriate to the particular task.

2.3.2 Identification of the ARCA Origin during initialization

An ARCA origin that wishes to identify itself to an ARCA target must find a way to communicate its identity during the initialization process. Although the otherInfo field is also present in the Init procotol data unit, there is a potential problem with using it during initialization, as stated in the standard:

> *Care should be taken by the origin when using this parameter; the origin cannot ascertain that version 3 is in force before sending the Init request.*

Because of this problem, the ARCA client should avoid the use of the otherInfo field during initialization. Fortunately, several other fields are available for communicating information that is outside the standard.

- *Implementation-id*. This field, whose exact contents are not specified by the standard, allows the client to identify a unique implementation. For example, this could identify an ARCA client in a machine-readable form.

- *Implementation-name*. This allows the client to provide a name by which it wants to be known in human readable form. For example, the name "ARCA" could be specified (although an implementation-id might provide a better guarantee against accidental name clashes with other clients).

- *Implementation-version*. This field is also not specified in the standard, and might allow the ARCA client and server to synchronize on which respective versions of the ARCA software they are using. This could also include information about the client implementation platform (e.g. Unix, Windows, Macintosh).

Any of these fields, or some combination of them, can be used by the ARCA client to identify itself to the ARCA target. During identification, the ARCA client could also include information on the most recent version of dictionary data held in the client.

In the Init response APDU, the ARCA target is free to use the otherInfo field in order to transmit ARCA-specific information (such as updated dictionary information).

2.3.3 Dynamic extensions of the protocol by ARCA Target using TCL scripts

The ARCA target is designed to work as a front-end to arbitrary OPACs. Because of this, it is not feasible to plan for special, as-yet unknown features of new OPACs. One approach to dealing with this problem is to provide for *dynamic extensions* of the protocol to support these features.

Dynamic extensions can be supported by using the otherInfo field of the Version 3 APDUs to communicate TCL scripts (which are already used in a similar context for other systems, in order to add functionality in a dynamic fashion). A scenario is as follows:

If a certain OPAC provides a functionality that cannot possibly be mapped onto the current version of the protocol, then the OPAC provider, together with ARCA consortium participants, defines the necessary protocol extension, which is communicated to the client in the otherInfo field. Together with this extension, to handle the extended features the target sends a TCL script to the client . In a Windows environment, it would also be possible to make use of DLLs in a similar fashion.

3 ARCA SR system architecture

3.1 ARCA SR Target subsystems

At the most abstract level the architecture of the ARCA SR Target System can be seen as decomposed in two modules: *Target Core*, which is independent from the interfaced OPAC, and *SR Target OPAC Interface*, which is specific to each library system (see figure 3.1). As the main goal of the ARCA Project is to minimize the effort required to incorporate a new library automation system, the functionalities of the two modules have been chosen appropriately in order to reduce as much as possible the complexity of SR Target OPAC Interface.

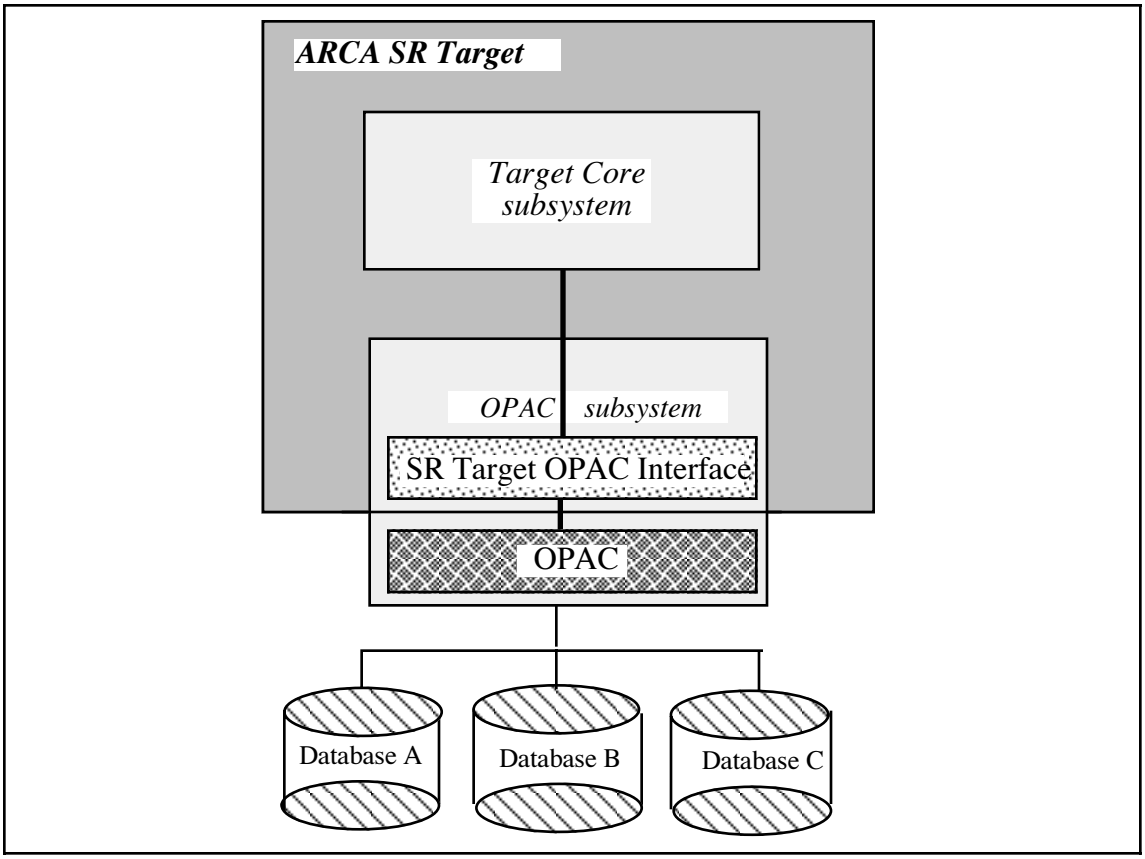


Figure 3.1 - ARCA SR Target Subsystems

The *Target Core* subsystem is a software component which is completely independent from the underlying OPAC. It accepts SR requests and serves them by, possibly, calling appropriate primitives of the SR Target OPAC Interface.

The *SR Target OPAC Interface* is a software component, specific to each OPAC, which renders the library service accessible through a set of primitives.

From the point of view of the Target Core subsystem, an OPAC and its SR Target Interface can be assimilated to a logical system which offers "OPAC-like" search capabilities. This justifies why this subsystem has been called OPAC subsystem.

As a consequence of the above architecture, the functionalities of a target built by interfacing an OPAC with an ARCA SR Target system result from the union of the "native" functionalities provided by the OPAC subsystem and those implemented by the Target Core.

This report focuses on the architecture of the Target Core and the interface among this and the OPAC subsystem. The SR Target OPAC interfaces for the OPACs Sabini and ISIS are presented in a different report.

3.2 OPAC subsystem

This section presents the assumptions on the OPAC subsystem which have been made in defining the Target Core architecture. Any OPAC subsystem which will be interfaced by a Target Core module must satisfy these assumptions.

The OPAC subsystem is characterized in terms of a set of constant parameters, which define its static characteristics, and a set of operations, which define its interface.

3.2.1 Parameters

Services

An OPAC can be considered as being a server application able to support a *subset of* the following *services*: Authentication, RPN-Search, Non-RPN-Search, Present (Export), Delete-result-set and Termination. If Delete-result-set is supported then also Present must be supported.

The Delete-result-set service can be supported for a single result-set, a list of result set, or all result sets held by the OPAC.

Databases

The OPAC provides access to one or more named *bibliographic databases*, each one being a collection of one or more files, with a unique name. The unit of information for retrieval from a database is a record. All the records within a given database have a common structure. An *access point* is a key which can be specified, together with the *access mode*, in a search for records. Databases combination can be specifed in a search, too.

Each database can export its records according to different *record compositions*. A composition is identified by a name and the set of its data elements. The default is *full database record*. The database can export its

records in various *transfer record syntaxes*. OPAC must support UNIMARC, at least.

An *average* and a *maximum size* of records is associated with each database.

Query language

The OPACs we consider are able to process one of the following query types: ISO8777, private, textual infix, textual RPN, RPN structure. Except the first two cases, it is possible to define the set of acceptable queries in terms of allowed *access points* and *selectors*, and *Boolean structures* supported.

Result sets

The OPAC is able to support *result sets* which are local data structures identifying records (i.e. composed by records identifiers, e.g. pointers). The logical structure of the result set is that of a named (by default) ordered list of triples consisting of:

a) an ordinal number corresponding to the position of the triple in the list;

b) a database name;

c) a unique identifier (of local significance only) of a record within the database named in b).

A result set item is referenced by its position within the result set, that is, by a).

The *number of results sets* which can be handled concurrently cannot exceed a maximum: the OPAC, however, must support at least the result set whose name is "default".

3.2.2 Operations

Each of the operations listed below assumes to be invoked in the right order. For example, if an authentication operation occurs, it must be the first one.

Each operation returns FALSE or TRUE according to whether it has been executed successfully or not; it also access a storage area, called CommArea, containing the error messages (MsgArea) and information of general use for the OPAC.

The CommArea includes:

- the MsgArea, which holds:
  - a bib-1 diagnostic as defined in ISO 10163 (mandatory)

- a bib-1 diagnostic as defined in ANSI/NISO Z39.50-1995 (optional)
- an OPAC error message (optional).

- an identifier for the Origin requesting the service that is used by the OPAC to check the privileges.

- a set of information identifying the origin.

OPAC.Authentication

This operation takes as input an authentication information and checks its validity. The form and content of this information are a matter of prior agreement between the origin and the target. If the information supplied as input is valid, then the OPAC stores the pertinent information into a local object and returns an appropriate code (TRUE) and a list of the available services. If not all the services are available on all databases, the OPAC can send an explanatory message, too. If it is invalid returns a code (FALSE) and a message explaining the kind of information required.

OPAC.RPN_Search

This operation may be applied to an OPAC specifying a list of *database names*, a *result set name* (optional) and a *query*. This operation assumes that:

a) all the databases indicated are associated with the OPAC and can be searched in combination;

b) the specified query is one of those which the OPAC can serve and it is an RPN query formulated in terms of an RPN structure ;

c) the creation of a new result set does not exceed the OPAC result set capacity.

The OPAC must verify that the requested service is compatible with the authentication information.

The result of a successful search is the association of a *result set* to the given result set name. As a consequence of this association the result set can be referenced in a subsequent search query statement and manipulated to form a new result set.

OPAC.Non_RPN_Search

The parameters of this search operation are: a list of *database names*, a *result set name* (optional) and a *query*. This search assumes that:

a) all the databases indicated are associated with the OPAC and can be searched in combination;

b) the type of query is one among {infix, ISO8777, private, RPN string};

c) the specified query is formulated in textual form;

d) the creation of a new result set does not exceed the OPAC result set capacity.

The OPAC must verify that the requested service is compatible with the authentication information.

As with the previous search operation, the result of a successful search is the association of a *result set* to the given result set name.

OPAC.Present

A present operation exports a subset of the records from a result set. These records are referenced by their relative position within the result set. This operation takes as input the *result set name*, the *start point* within the result set, the *number of records requested*, the *record syntax*, and the *record composition*. The last parameter specifies the desired composition of retrieval records. The present operation assumes that result set name, record syntax and record composition are known to the OPAC.

The OPAC must verify that the requested service is compatible with the authentication information.

The present operation returns a *list of references to documents* which satisfy the given specification. If (start point + number of records requested) is greater than (number of records within the result set + 1) then the documents selected are those from the start point to the last document in the result set. Of course, if the start point is greater than the last position in the result set then the subset returned is empty.

OPAC.Delete

The delete result set operation takes in input a *list of result set names*. The opearation cancels the association between a result set name, given as input, and the corresponding result set. This operation assumes that the result set name is known to the OPAC.

OPAC.Termination

This operation fires when the origin communicates his intention to terminate the session, or the ARCA target decides to kill it. This operation takes as input a *list of result set names*, which are assumed to be known to the OPAC, a deletes all the association between them and the corresponding result sets.

4 Description of the architecture

The definition of the architecture of the Target Core subsystem follows the stage of understanding the ARCA System requirements. In this section these requirements are made precise by building a model according to the OMT methodology, which is described in Appendix A. Let us remind that it covers three aspects: the static structure (object model), the sequencing of interactions (dynamic model) and data transformations (functional model). The architecture has been designed using the automated CASE tool StP/OMT ([StP/OMT]), whose characteristics are briefly described in Appendix A.

4.1 The Object Model

The OMT methodology requires to specify first the context of the system. In our case, it consists of three interacting subsystems (Figure 4.1):

- *Origin Subsystem*
  It describes how the interaction between the Target Core and the origin occurs through the network.

- *Target Core Subsystem*
  It consists in all the components that are involved in the implementation of the SR functionalities.

- *OPAC Subsystem*
  It represents the API interface level between ARCA and the underlying OPAC.

As regard to the properties of each class, we must outline that we will consider only those who are needful to understand the general functionalities of the ARCA system. In particular, we will provide a description of the methods showing only the most relevant input parameters. Among the input parameters of each method, we won't include the type of the object the method is acting on because it is implicit in the object-oriented design style.

Each class will present two methods that are for the creation and the deletion of the instances (objects) of that class. In respect with the C++ notation, these methods will be called *constructors* and *destructors* and will be identified respectively by <class-name> and by ~<class-name>.

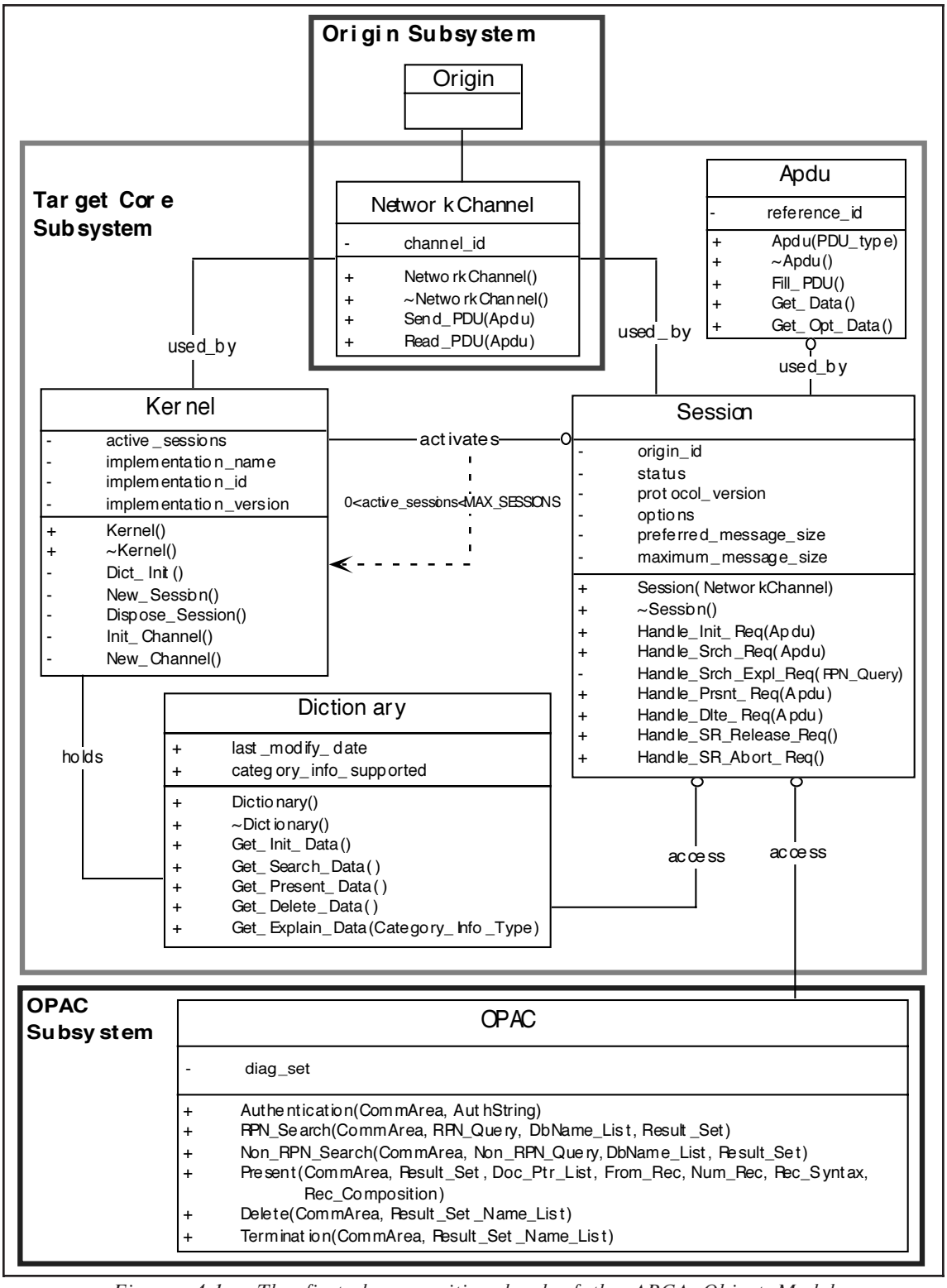


Figure. 4.1 - The first decomposition level of the ARCA Object Model

4.1.1 Class Apdu

The instances of this class represent the data structures exchanged between the origin and the target. This class abstracts the common structure among the APDUs (Table 4.1) used by each Session object during its existence. The

APDUs can be divided according to their type each one representing a message exchanged between the origin and the target (Figure 4.2).

Attribute	Description
reference_id	Field common to all the APDU types (Presently, it has meaning only for the origin).
Method	
Apdu(PDU_type)	In this case, the constructor needs also the APDU type to create the right APDU object (i.e. InitPDU, SearchPDU and so on.)
Fill_PDU	Fills the specified APDU with the proper data.
Get_Data	Returns data about the APDU.
Get_Opt_Data	Returns optional data about the APDU.

Table. 4.1 - The properties of the Apdu class

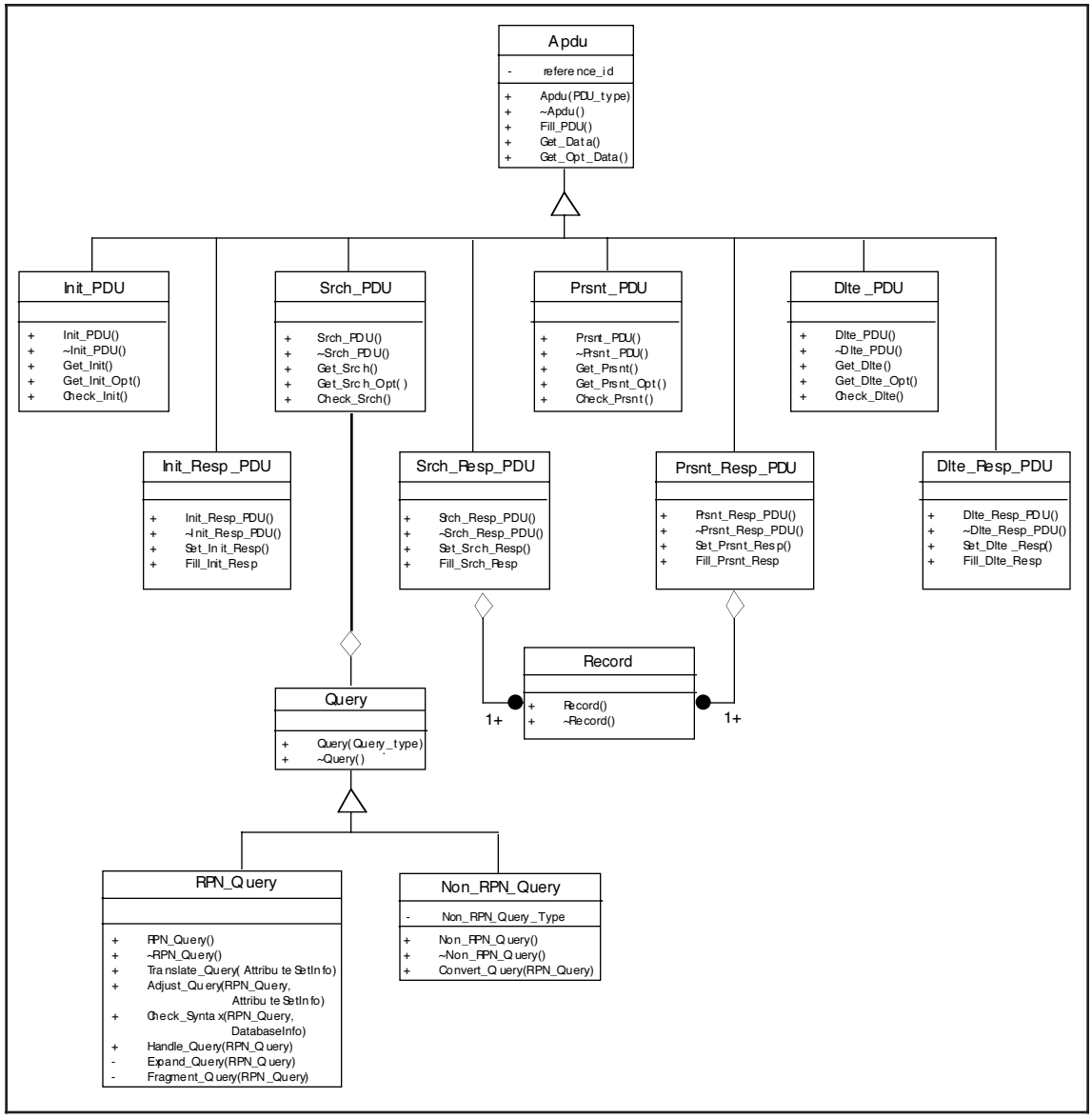


Figure. 4.2 - The first decomposition level of the ARCA Object Model

The methods acting upon an APDU are specialized for each subtype (Table 4.2 - 4.5).

Method	Description	
	Init_PDU	**Init_Resp_PDU**
Set_Init_Resp		Sets particular data of the initialize response PDU.
Fill_Init_Resp		Fills the initialize response PDU.
Get_Init	Returns the mandatory fields of the initialize request PDU.	
Get_Init_Opt	Returns the optional fields of the initialize request PDU.	
Check_Init	Performs a check of the incominig request against the data in the dictionary.	

Table 4.2 - The properties of the Init_PDU and Init_Resp_PDU classes

Attribute	Description	
	Srch_PDU	**Srch_Resp_PDU**
query	Query.	
records		Returned records.
Method		
Set_Srch_Resp		Sets particular data of the search response PDU.
Fill_Srch_Resp		Fills the search response PDU.
Get_Srch	Returns the PDU mandatory fields.	
Get_Srch_Opt	Returns the PDU optional fields.	
Check_Srch	Checks the current request against the dictionary.	

Table 4.3 - The properties of the Srch_PDU and Srch_Resp_PDU classes

Attribute	Description	
	Prsnt_PDU	**Prsnt_Resp_PDU**
records		Returned records.
Method		
Set_Prsnt_Resp		Sets particular data of the present response PDU.
Fill_Prsnt_Resp		Fills the present response PDU.
Get_Prsnt	Returns the PDU mandatory fields.	
Get_Prsnt_Opt	Returns the PDU optional fields.	
Ceck_Prsnt	Checks the current request against the dictionary.	

Table 4.4 - The properties of the Prsnt_PDU and Prsnt_Resp_PDU classes

Method	Description	
	Dlte_PDU	**Dlte_Resp_PDU**
Set_Dlte_Resp		Sets particular data of the delete response PDU.
Fill_Dlte_Resp		Fills the delete response PDU.
Get_Dlte	Returns the PDU mandatory fields.	
Get_Dlte_Opt	Returns the PDU optional fields.	
Check_Dlte	Checks the current request against the dictionary.	

Table 4.5 - The properties of the Dlte_PDU and Dlte_Resp_PDU classes

Among the items composing the search APDUs, an important role is played by the query (Table 4.6). Such objects represent the type of queries that can be processed by the ARCA system. If the query is a RPN query (Table 4.7), ARCA will perform several checks on its admissibility in respect with the characteristics of the underlying OPAC, and, as much as possible, the query will be transformed to be successfully submitted to the OPAC.

Method	Description
Query(Query_type)	In this case, the constructor needs also the query type to activate the constructor of the corresponding class.

Table 4.6 - The properties of the Query class

Method	Description
Translate_Query	Trasforms the query by translating the attributes of the BIB-1 set into the related attribute of the OPAC set.
Adjust_Query	Adjusts the RPN query to produce a new one which presents the selectors in the rigth positions.
Check_Syntax	Checks whether the RPN query is formulated according to the allowed syntax.
Handle_Query	Transforms the query according to the values of "result_set_naming" and "max_num_term" attributes of the TargetInfo object.
Expand_Query	Expands the query by replacing each reference to a result set with the corresponding subquery.
Fragment_Query	Fragments the query in a set of simple (two operands and one operator) subqueries.

Table 4.7 - The properties of the RPN_Query class

The selectors will be placed in the right position and, to overcome the OPAC limitations about backreferencing and number of composing terms, the query will be opportunely managed. Moreover, the access points of bib-1 will be translated according to the OPAC native value and, if the OPAC doesn't support RPN query processing, the query will be converted to an infix or a RPN textual notation (Table 4.8). No check will take place if the incoming query type will be ISO8777 or private textual whilst a partial check occur on a ARCA Origin query. The Search and Present Response APDUs shall return a collection of records representing a part of or the whole result of the submitted queries (Table 4.9).

Attribute	Description
Non_RPN_Query_Type	A query type among infix string, RPN string, ISO8777, textual private.
Method	
Convert_Query	Produces the infix or RPN textual version of a RPN query.

Table 4.8 - The properties of the Non_RPN_Query class

Attribute	Description
Record	A database or diagnostic record.
Method	

Table 4.9 - The properties of the Record class

4.1.2 Class NetworkChannel

As it is shown in Figure 4.1, the Origin Subsystem and the ARCA Subsystem share the NetworkChannel class. This class allows to abstract the usual functions of sending and receiving data over a network from the implementation of the OSI transport layer. Therefore, we will be able to change the implementation of such a layer according to the chosen protocol (for example TCP/IP or ISODE) without affecting the rest of the system. For such reasons, we decided to abstract the connection between the target and an origin (for example a TCP/IP or ISODE address) and generally refer to it as a network channel. Among the functionalities demanded to NetworkChannel methods (Table 4.10), there are also the encoding decoding of the outcoming incoming data as stated in the requirements of ISO 10163 [ISOb]. This enables us to deal, in the rest of the ARCA system, only with APDUs without regarding how they are encoded or decoded. At execution time, N+1 NetworkChannels will be instantiated (N for the currently active connections with the origins and one for the connection which the Kernel object waits on).

Attribute	Description
channel_id	Unique identifier for a network channel.
Method	
Send_PDU	Takes as input an APDU, encodes it and sends it down the network channel.
Read_PDU	Reads an octet string from a network channel and decodes it building an APDU to be used in the ARCA system.

Table. 4.10 - The properties of the NetworkChannel class

4.1.3 Class Dictionary

The Dictionary holds data that describe the entire ARCA system. For each service (Table 4.11), a check against those data takes place, in order to make the OPAC serve only correct requests. The Dictionary also holds data to support the EXPLAIN Service.

Attribute	Description
last_modify_date	Date of last update of the Dictionary
category_info_supported	Supported Explain category info
Method	
Get_Init_Data	Returns the information about the Initialize service.
Get_Search_Data	Returns the information about the Search service.
Get_Explain_Data	It takes as input the Explain category to which data refer. Returns the information about the EXPLAIN service.
Get_Present_Data	Returns the information about the Present service.
Get_Delete_Data	Returns the information about the Delete service.

Table 4.11 - The properties of the Dictionary class

As it is shown in Figure 4.3, the unique Dictionary object can be seen as composed of items belonging to eight subclasses. Each subclass represents an Explain category as stated in the Z39.50 V3 protocol.

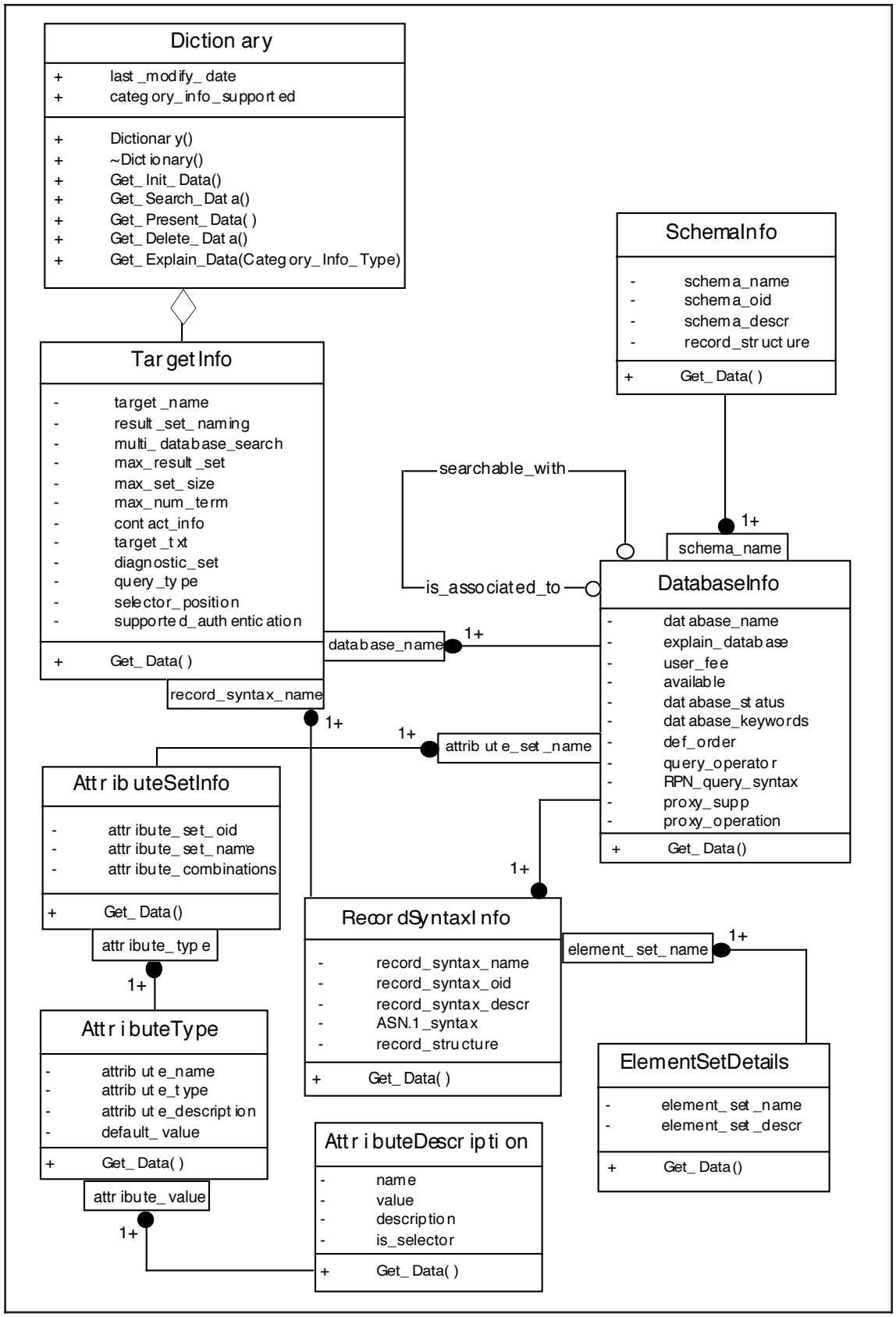


Figure 4.3 - The Dictionary context

The class TargetInfo (Table 4.12) is concerned with information related to the whole ARCA system. Such information corresponds to facilities

provided by the underlying OPAC natively or by means of ARCA procedures.

Attribute	Description
target_name	Target name.
result_set_naming	Indication about the possibility of result set naming.
multi_database_search	Indication of whether or not ARCA supports the search on more than one database.
max_result_set	Maximum number of result sets ARCA handles.
max_set_size	Maximum number of records in a result set.
max_num_term	Maximum number of terms allowed in a query.
contact_info	Information about the person or institution to call.
target_txt	Description of the target system.
diagnostic_set	Diagnostic sets supported by ARCA.
query_type	The types of the queries ARCA can satisfy.
selector_position	The allowed selector position in a query.
supported_authentication	Indication of whether or not ARCA supports access control.
Method	
Get_Data	Returns the data about the target.

Table 4.12 - The properties of the TargetInfo class

The target satisfies the requests directed to one or more databases of the underlying OPAC. Each database can be searched in combination with others and has proper indications regarding the way a query should be formulated to be processable. Information about this is given by the attributes regarding operators and RPN query syntax (Table 4.13) and attribute combinations (Table 4.15).

A database can have one or more schemas i.e. common understandings about the information contained in the database records (Table 4.14).

A database recognizes one or more sets of attributes (bib-1 is mandatory) each one composed by a specific collection of attributes which may be only a part of the entire attribute set. The associations between the allowable attribute combinations in the search query and the OPAC access points is described (Table 4.15). The information about each single attribute is maintained (Table 4.16-4.17).

Obviously, database records are stored according to a specific record syntax (i.e. USMARC, UNIMARC) that can be associated with a collection of element sets (Table 4.18 - 4.19).

Attribute	Description
database_name	Full database name.
explain_database	Whether this is an Explain database.
user_fee	Whether there is a fee to pay for searching.
available	Whether the database is available.
database_status	Whether this database is currently searchable.
database_keywords	List of keywords for the database.
def_order	Default order in which records are presented.
query_operator	Supported query operators.
RPN_query_syntax	Allowed RPN query syntax.
proxy_supp	Whether proximity operations are supported.
proxy_operator	Proximity parameters supported.
Method	
Get_Data	Returns the data about the requested database.

Table 4.13 - The properties of the DatabaseInfo class

Attribute	Description
schema_name	Name of the schema.
schema_oid	Object identifier of the schema definition.
schema_descr	Description of the schema.
record_structure	Abstract record structure defined by this schema.
Method	
Get_Data	Returns the data about the requested schema.

Table 4.14 - The properties of the SchemaInfo class

Attribute	Description
attribute_set_oid	Object identifier of the attribute set.
attribute_set_name	Name of the attribute set.
attribute_combinations	Table which associates the allowable attribute combinations of the search query with the corresponding OPAC access points.
Method	
Get_Data	Returns the data about the requested attribute set.

Table 4.15 - The properties of the AttributeSetInfo class

Attribute	Description
attribute_type	Attribute type.
attribute_name	Attribute type name.
attribute_description	Attribute type description.
default_value	Attribute default value.
Method	
Get_Data	Returns the data about the requested attribute.

Table 4.16 - The properties of the AttributeType class

Attribute	Description
name	Attribute name.
value	Attribute value.
description	Attribute description
is_selector	Whether this attribute is a selector.
Method	
Get_Data	Returns the data about the requested attribute.

Table 4.17 - The properties of the AttributeDescription class

Attribute	Description
record_syntax_name	Name by which this syntax is known.
record_syntax_oid	Object identifier of the abstract record syntax.
record_syntax_descr	Description of the abstract record syntax.
ASN.1_syntax	ASN.1 module describing the syntax.
record_structure	Record structure defined by the syntax.
Method	
Get_Data	Returns the data about the requested record syntax.

Table 4.18 - The properties of the RecordSyntaxInfo class

Attribute	Description
element_set_name	Element set name.
element_set_description	Description of the element set.
Method	
Get_Data	Returns the data about the requested element set.

Table 4.19 - The properties of the ElementSetDetails class

4.1.4 Class Kernel

At the run time, there will be a unique instance of this class (Table 4.20). Such an instance will accomplish the following main tasks:

- At the system start up, it will load and initialize the dictionary and create a network endpoint which it will wait on for new connection requests.

- When a request is detected, such Kernel object will instantiate a new network channel that will be used by a new Session object. Such an object will take care of the requests from the origin allowing the kernel to keep on waiting for new requests.

- When a Session will notify to terminate a connection, the Kernel will make the resources free.

The Kernel object activates zero or more Session instances without exceed the MAX_SESSIONS limit. It holds the Dictionary.

Attribute	Description
active_sessions	The number of the sessions currently active.
implementation_name	The implementation name of the ARCA system.
implementation_version	The implementation version of the ARCA system.
implementation_id	The implementation identifier of the ARCA system.
Method	
Dict_Init	Loads and initializes the dictionary.
Init_Channel	Initializes an instance of NetworkChannel for waiting new connection requests.
New_Channel	Takes as input a channel and returns a new instance where the origin will continue sending requests.
New_Session	Creates a new instance of the Session class.
Dispose_Session	Terminates an instance of a Session object and releases all the resources held from it.

Table 4.20 - The properties of the Kernel class

4.1.5 Class *Session*

Session objects (Table 4.21) provide functions for each of the services defined in the SR protocol. They constitute the real interface between the origins and the OPAC Subsystems. At this level of decomposition, we only present the methods corresponding to the services. We will consider as attributes all the parameters that characterize a user working session. These parameters are those a Session object negotiates with the origin in the initialization phase and, once fixed, they will not be modified until the end of the working session.

The Sessions access the Dictionary to retrieve the information necessary to send appropriate request to the underlying OPAC or to respond to searches on Explain database. It exchanges APDUs with the associated origin through a network channel.

Attribute	Description
origin_id	Identifier of the origin system in the target environment.
status	Current status of the work session.
protocol_version	Protocol version negotiated in the initialization phase.
options	The services that the origin could request.
preferred_message_size	The size of the large messages.
maximum_message_size	The absolute maximum size.
Method	
Session(NetworkChannel)	In this case, the constructor needs also the network channel given by the Kernel object to the session object to wait for the requests from the origin.
Handle_Init_Req	Provides the Initialize service.
Handle_Srch_Req	Provides the Search service.
Handle_Srch_Expl_Req	Provides the Search Explain Service. It is called within Handle_Srch_Req.
Handle_Prsnt_Req	Provides the Present service.
Handle_Dlte_Req	Provides the Delete service.
Handle_SR_Release_Req	Provides the SR-Release service.
Handle_SR_Abort_Req	Provides the SR-Abort service.

Table 4.21 - The properties of the Session class

4.1.6 Class OPAC

Since we decided to process requests only for one OPAC, only a single object of this class will exist during the ARCA system execution. The Opac object is assumed to be a permanent object; this is why the Opac class does not present either the constructor and the destructor. This class abstracts the API interface level between the OPAC and the ARCA system (Table 4.22). It provides the basic functions to retrieve and manage collections of records. If a failure occurs, all functions return diagnostic information in the following format:

- a bib-1 diagnostic as defined in ISO 10163 (mandatory);
- a bib-1 diagnostic as defined in ANSI/NISO Z39.50-1995 (optional);
- an OPAC error message (optional).

Attribute	Description
diag_set	Diagnostic set used by the OPAC.
Method	
Authentication	Identifies the user of the origin who makes a request.
RPN_Search	Performs a search taking as input a RPN query with the OPAC native terms and builds the result set.
Non_RPN_Search	Performs a search taking as input a query in the (RPN, infix, ISO8777, private) textual mode and builds the result set.
Present	Formats the retrieved record according to the specified record syntax and composition and returns the reference to those records.
Delete	Deletes the specified result sets.
Termination	Releases the resources allocated into the OPAC for a particular user (e.g.: temporary result sets).

Table 4.22 - The properties of the Opac class

4.2 The Dynamic Model

The dynamic model describes the dynamic behaviour of the objects that can exist. In this section, we will give only the dynamic model of the Kernel and the Session objects. In fact, we want to put more emphasis upon the sequencing of interactions concerning the Target Core subsystem.

4.2.1 Dynamic model of the Kernel object

The figure 4.4 illustrates the dynamic behaviour of the Kernel object. At the system start up, it initializes the Dictionary and the NetworkChannel objects by calling the proper methods. Then, it entries into the *waiting_request* state and waits for a new connection and abort requests. Connection requests arise from the NetworkChannel and require the Kernel to check the MAX_SESSIONS parameter before instantiating a new Session object. Such object will take care about the requests of the SR service. The abort requests come from Session objects who notify the end of an association, and require the disposal of a Session object.

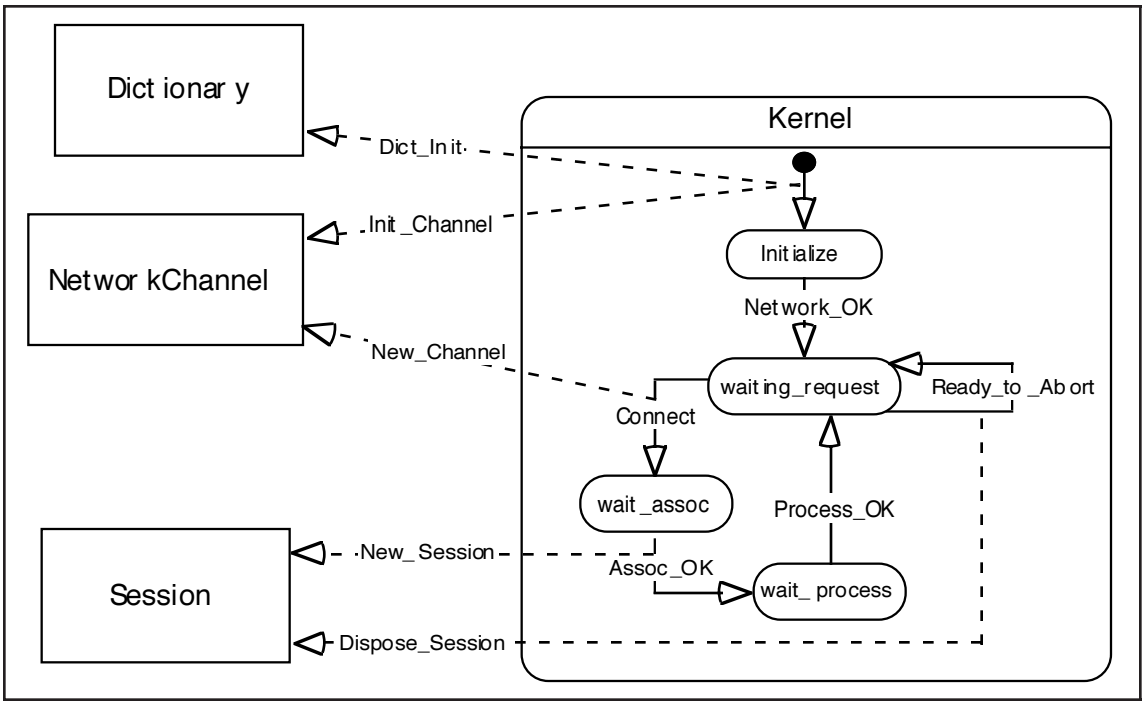


Figure 4.4 - The dynamic model of the Kernel object

4.2.2 Dynamic model of the Session objects

A representation of a Session object behaviour can be seen in figure 4.5. A Session object waits for a request coming from the network, sends the related event to the OPAC and waits for the answer. Once the service is completed, the Session object returns to the "Waiting_Request" state.

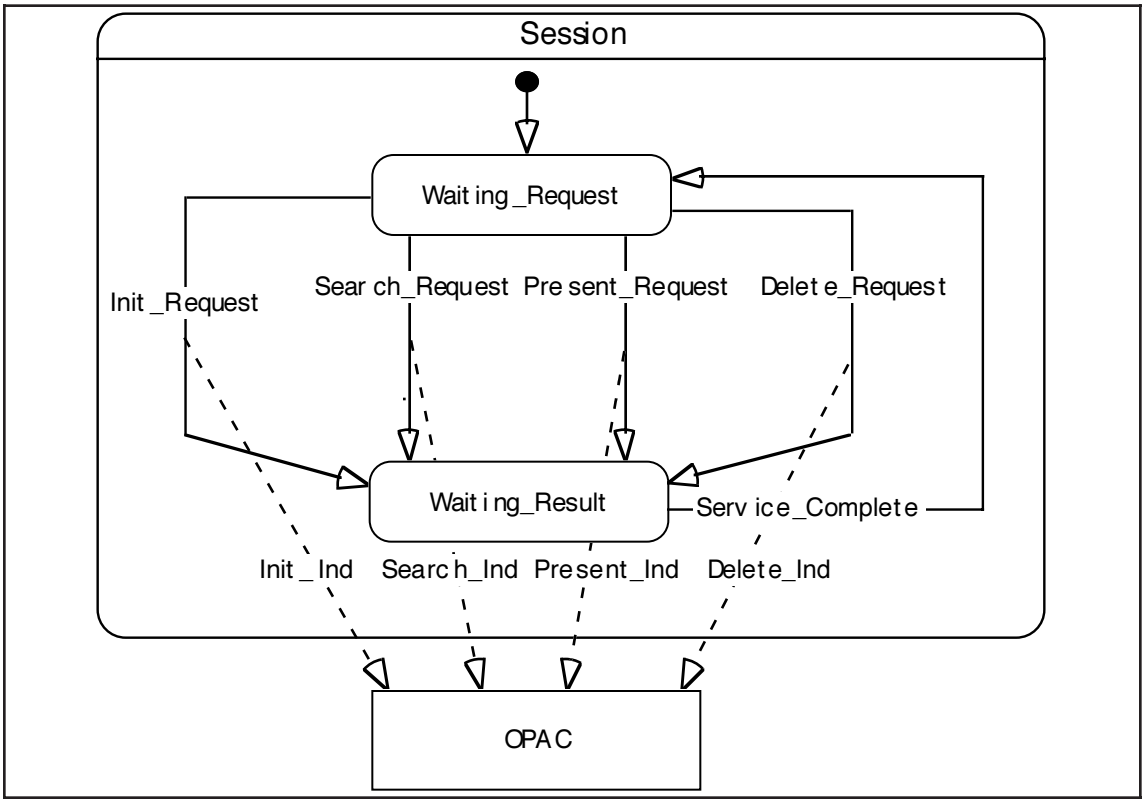


Figure 4.5 - The dynamic model of the Kernel object

The Session objects (Figure 4.6), once instantiated, wait for an Initialize APDU in the *Closed* state. When such APDU is received two cases can occur. If the Init APDU will come from a generic SR origin, the parameters will be compared with the related ones in the Dictionary in order to determine if the connection can be accepted. In case of rejection, an APDU containing the reject indication will be sent and the Session object will wait on the *Reject* state for a subsequent release request coming from the origin. If the connection will be accepted, the Session object will go to the *Open* state where it will wait for the requests about the other SR services.

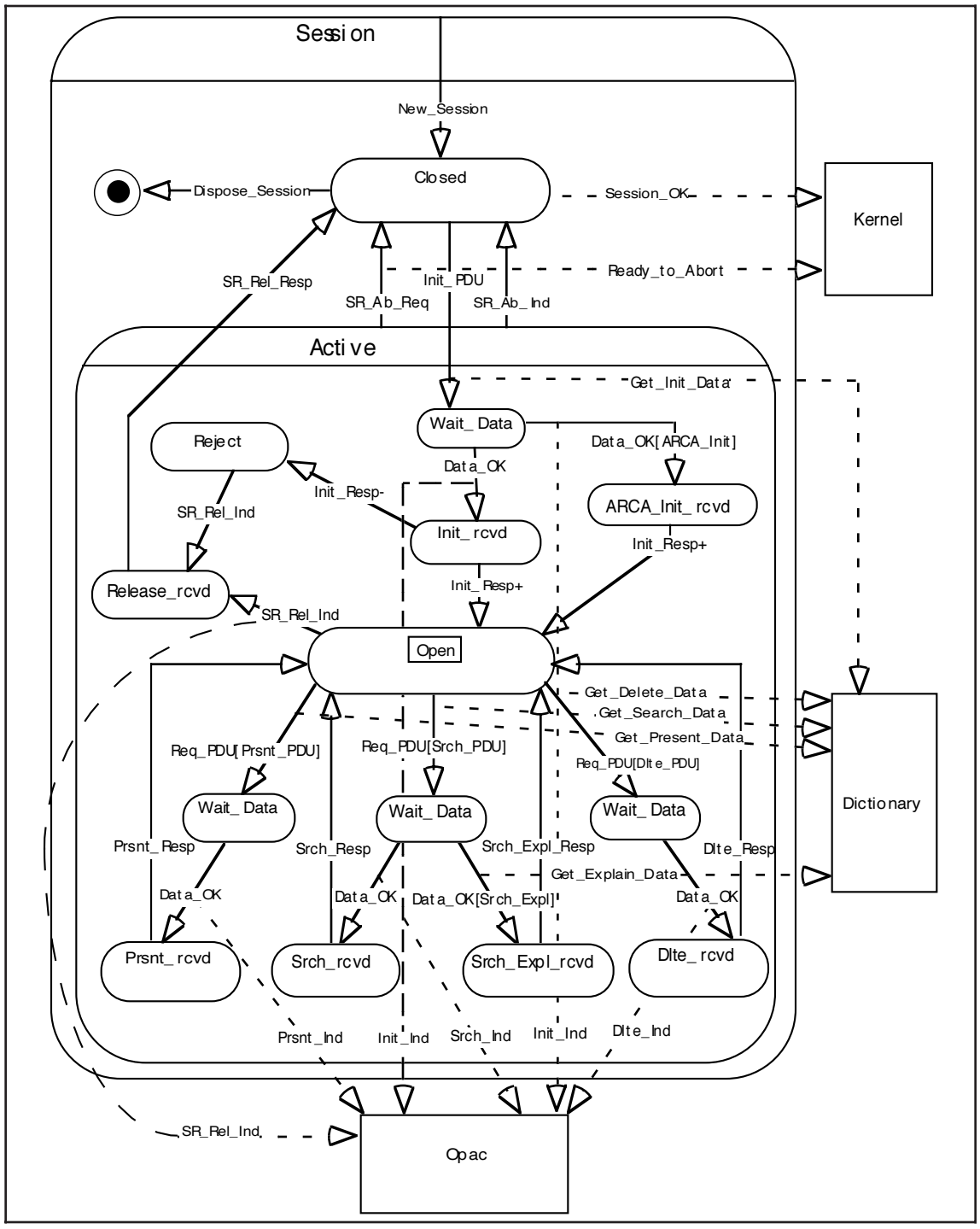


Figure 4.6 - The dynamic model of the Session objects

The arrival of a SR Search, Present and Delete services will cause a comparision between the APDU parameters and the related ones stored in the Dictionary in order to verify the request correctness. After this check, the appropriate OPAC API will be called and the interaction will continue as described in the *Open* state. However, we must stress that if the database to be searched will be the Explain database, the query will be submitted to the Dictionary rather than the OPAC.

In every active state, a SR_Abort can be accepted. In this case, the OPAC will be recovered in a consistent status by calling the Termination API and the environment concerning the current working session will be cleared. At the termination of each connection, the related Session object will ask the Kernel object to be killed.

4.3 The Functional Model

As it was previously mentioned, the functional model represents the data transformations across the various methods of the classes involved in the project. We will limit the description of the functional model to the methods concerning the receiving, the processing and the sending of the various APDUs since they constitute the most relevant part of the ARCA system.

In Figure 4.7, we show a first level functional model representing the APDUs transformations. Such transformations involve four methods belonging to the Session class (Handle_Init_Req, Handle_Srch_Req, Handle_Prsnt_Req, Handle_Dlte_Req) and two belonging to the NetworkChannel class (Read_PDU, Send_PDU). After the reception and the recognition of an incoming APDU from a network channel, the method associated with its processing will be called. Subsequently, the outcoming APDU will be passed to the Send_PDU method that will send it through the network channel. Since the Read_PDU and Send_PDU methods are implemented using the functionalities offered by the YAZ toolkit, we will describe, in the following, only the submodels of the Session methods.

If a Init_PDU will be received (Figure 4.8), it will be checked against the information stored in the Dictionary to verify if the connection parameters proposed by the origin can be accepted by the target. At this point, the target will decide to accept or deny the proposed connection. However, if the connection will be accepted, the target will be able to overwrite some of the submitted parameters. In any case, an Init_Resp_PDU will be created, filled and sent.

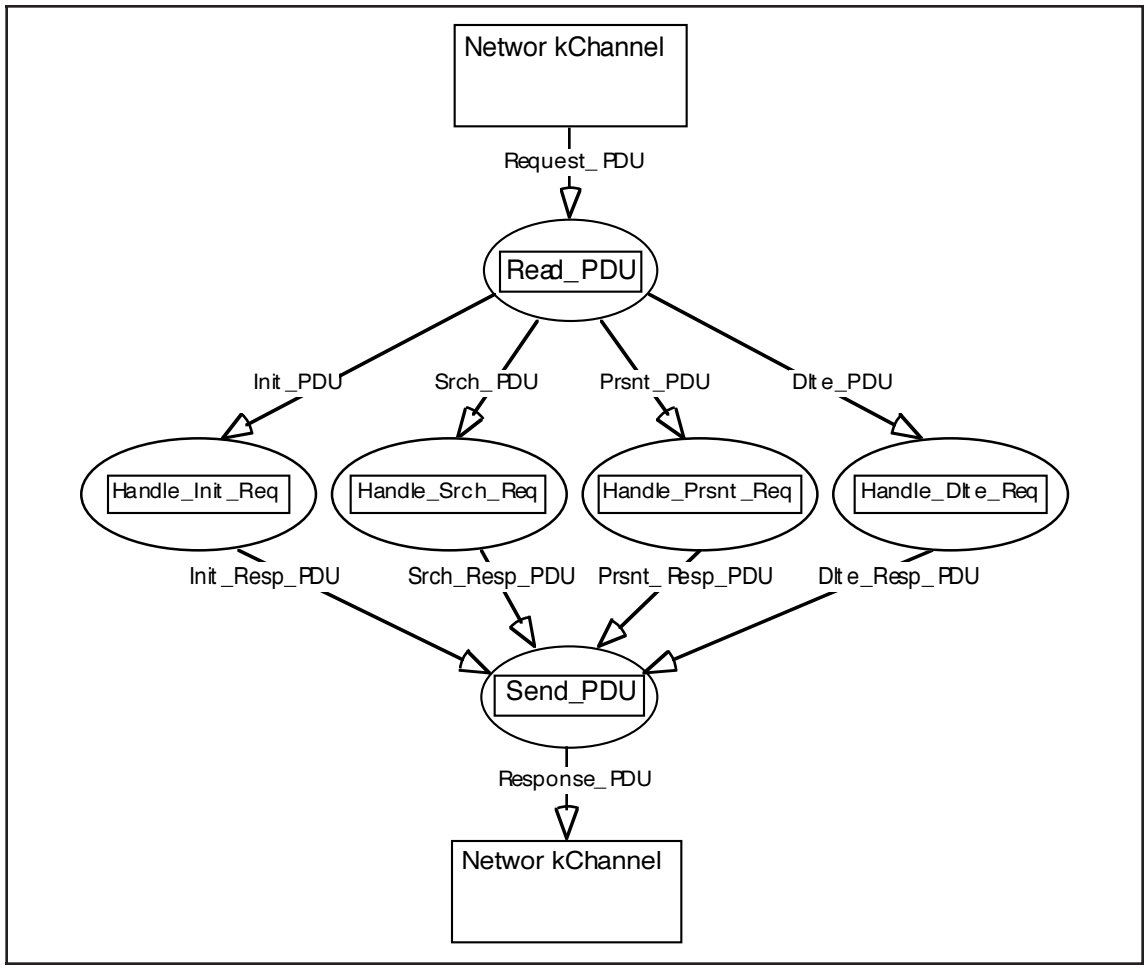


Figure 4.7 - The functional model about APDU processing

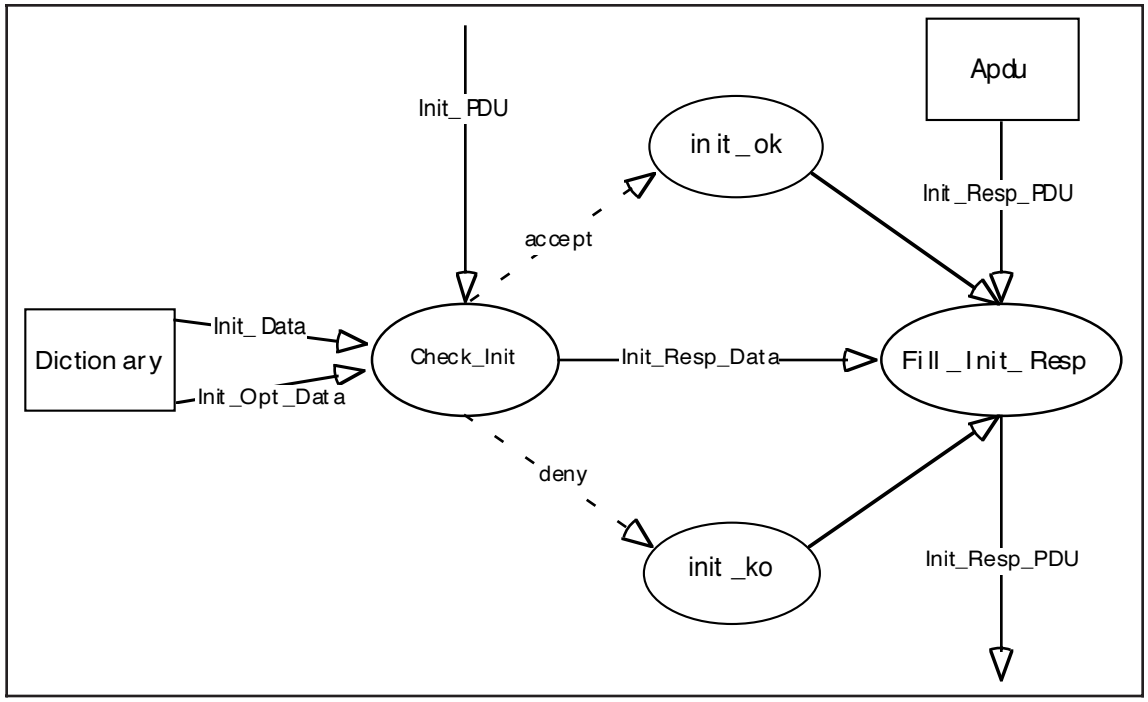


Figure 4.8 - The functional submodel about the processing of an Init_PDU

In case of reception of a Srch_PDU (Figure 4.9), a first check on the query type will take place. If the query is an ISO8777 or private query, it will be submitted to the OPAC without performing any check. Otherwise, if it is a RPN queryit will be adjusted to put the possible selectors in the right position and, subsequently, it will be checked against the allowed query syntax.

In both cases, the query will be transformed according to the value of "max_num_term" and "result_set_naming" attributes of TargetInfo class. If the OPAC doesn't support result set naming, the references to old result sets will be exploded in the corresponding query. If there is a limitation on the maximum number of terms, the query will be fragmented in a set of simpler queries (Fig. 4.10). The various cases and the consequent actions are summarized in table 4.23.

Unlimited # of terms	**Result set supported**	**Action**
Y	Y	The query is unchanged.
Y	N	For each result set referred in the incoming query, the corresponding query is substituted in the resulting query.
N	Y	The query is subdivided in several subqueries (one operator and two operands). The Session process will invoke repeatedly the appropriate API function, supplying the result set name when needed. At the end of the process, the temporary result sets created will be deleted invoking the delete result set API.
N	N	If the incoming query can be subdiveded into several subqueries where at least one operand is a simple term, the Session will invoke repeatedly the API search, using the default result set to store the temporary results. The OPAC must accept the last created result set as operand.

Table 4.23 - The possible actions to perform on the incoming query

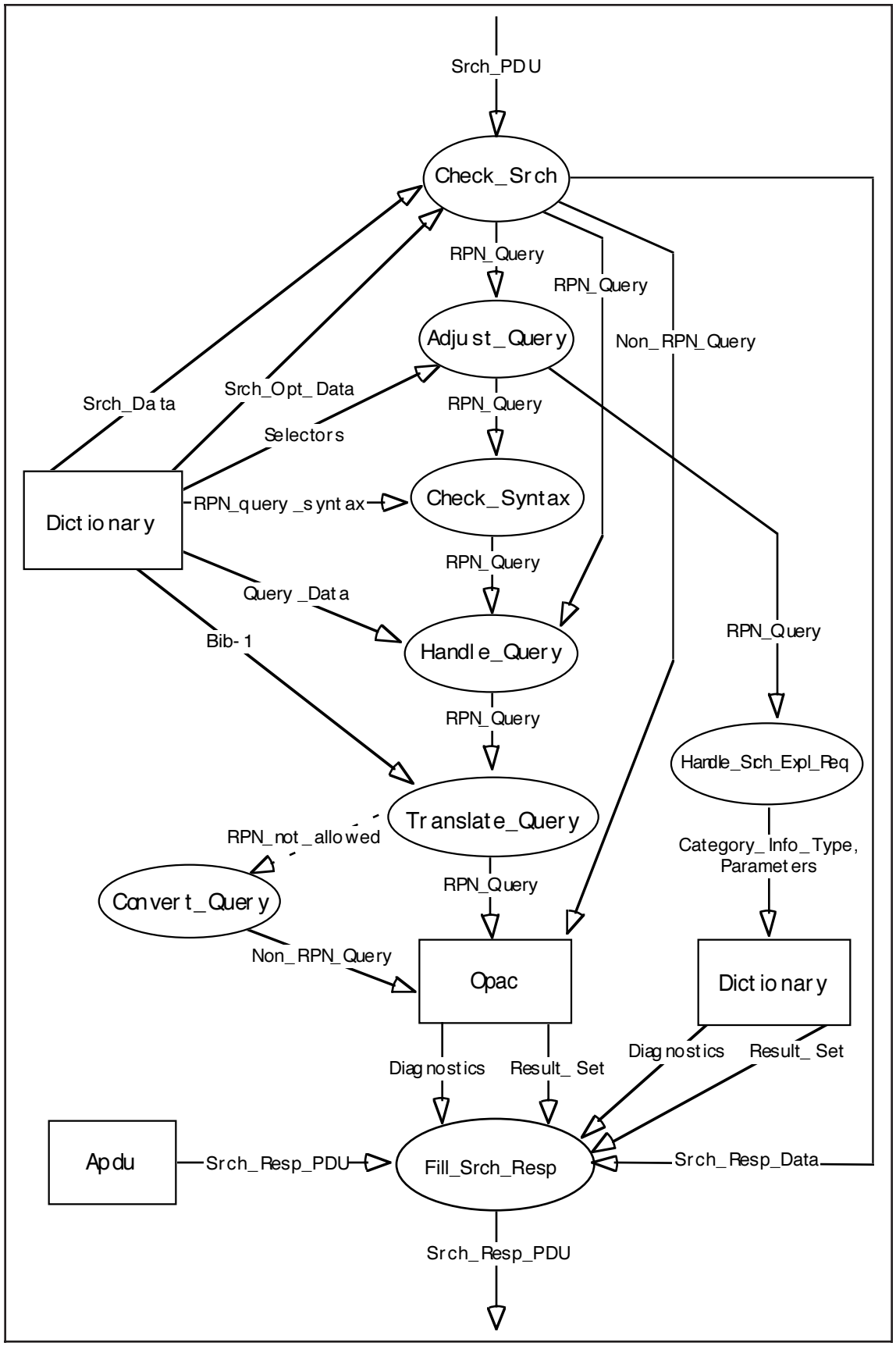


Figure 4.9 - The functional submodel about the processing of an Srch_PDU

Before submitting the query to the OPAC, the bib-1 access points value will be translated to the corresponding OPAC native value ([ARCA/T12/ABD]).

After adjusting the query, we will be aware of the databases the search considers. In particular, in an EXPLAIN service request, the search acts on the Dictionary. If the search will not consider the Explain database and the OPAC is not able to process RPN queries, the query will be converted in a (infix or RPN) textual form.

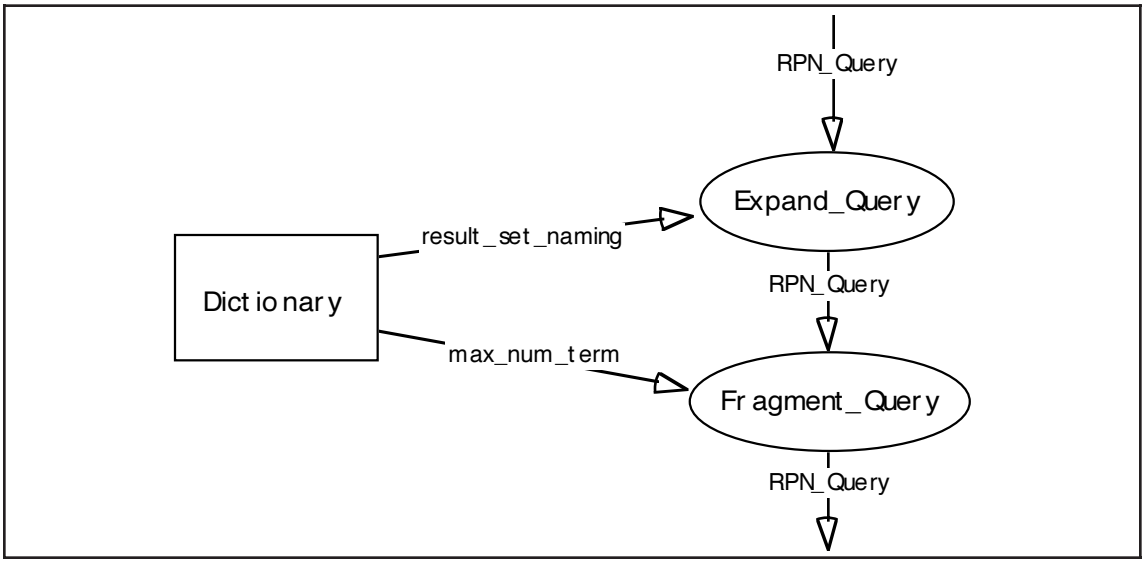


Figure 4.10 - The functional submodel of the Handle_Query process

Whatever the search scenario will be, a Srch_Resp_PDU will be generated and sent. Records will be included in the Srch_Resp_PDU in respect with the negotiated result set bounds.If a Prsnt_PDU will be received (Figure 4.11), a process will check it and ask the dictionary for the information regarding the record composition structure. Afterward, the formatted records and the further information needed for the response will be used to create and fill a Prsnt_Resp_PDU.

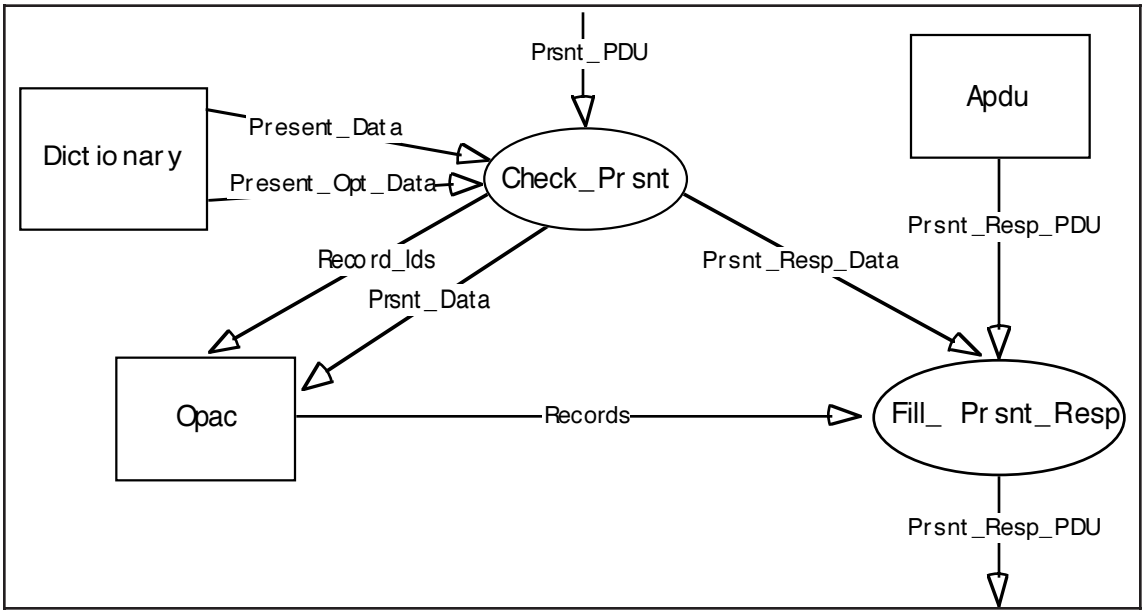


Figure 4.11 - The functional submodel about the processing of an Prsnt_PDU

Finally, when a Dlte_PDU will be received (Figure 4.12), a process will check the APDU against the information extracted by the Dictionary and pass the result set to be cancelled to the Delete Opac method. Such process will translate the logical name of the result set into the physical one and request the delete operation of the underlying OPAC. The OPAC response will be used to complete the Dlte_Resp_PDU.

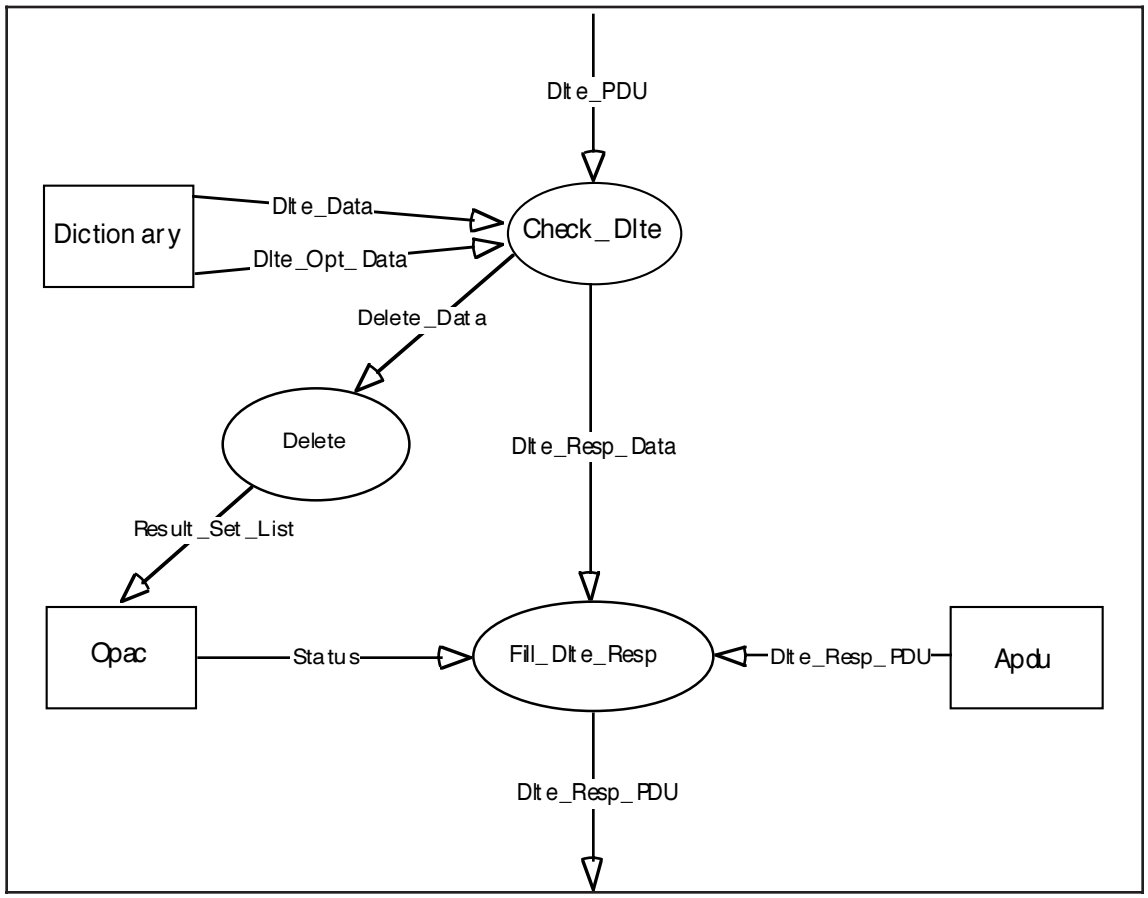


Figure. 4.12 - The functional submodel about the processing of an Dlte_PDU

5 Setting up the system

As it has been outlined in the previous sections, the behavior of the Target Core is parametric with respect to the static characteristics of the interfaced OPAC (see section 3.2). These characteristics regarding the information stored in the Dictionary and the OPAC syntax for processable RPN queries are made known to the Target Core when the system is set up. In addition to a simple keyword based language, a graphical and interactive interface will be provided to input these parameters and to change their values when necessary.

The ARCA Target system will run on UNIX environments thus enabling a subsequent easy porting on LINUX. It will be implemented under Sun OS 4.0. The code will be written in C++ and compiled and linked using the GNU public software. To syntactically parse the incoming RPN query in a search request, i.e. to implement the "Check_Syntax" method of the "RPN_Query" class, ARCA needs the LEX and YACC free libraries. In particular, GNU FLEX and BISON could be used.

The ARCA system will use the YAZ toolkit library for what concerns decoding/encoding the APDUs, and handling the communication with a connection endpoint (see Appendix B) that will be implemented by adopting TCP/IP sockets.

As regards to the source code language of the interface to set up the information being loaded in the Dictionary, we are thinking of Tcl/Tk.

The OPAC administrator is required, at least, to provide all information needed by the Target Core to perform the basic services, i.e. init, search, present and delete. For example, the administrator has to specify how bib-1 attributes' combinations are mapped into OPAC native language, the limitations on the result sets, the supported record syntaxes, the maximum record size, and so on. Some of these parameters will be proposed to the OPAC administrator with a default value. The installation of the ARCA system involves the creation of a set of work directories, therefore, the OPAC administrator will have to assign the appropriate permissions to ARCA.

To assist the implementation of the API functions, the OPAC administrator will be provided with the skeletons of the methods of the "OPAC" class. After completing these dummy functions, he/she will have to compile and link them to the rest of the system.

For some specific needs, like translation from a record syntax to another, the OPAC administrator will have to provide in the Dictionary the names of the relevant modules. These modules must be compiled and linked with the whole system, too.

As the OPAC can make use of its workareas, it will be up to the OPAC administrator to set up the appropriate shell script to clear and/or allocate all the needed workareas when the ARCA system will start up.

6 SR Target ISIS Interface

This section describes the unterface between the ARCA system and the ISIS OPAC. Firstly, the four main functions to access a CDS/ISIS database are presented. Secondly, the supporting structures and the APIs of the interface are specified.

6.1 ISIS OPAC

The CDS/ISIS is an Information Retrieval System developed and distributed by UNESCO in different versions running on SCO UNIX, DOS/WINDOWS and VAX/VMS.

The available software is able to access a single CDS/ISIS database at time, therefore access to multiple databases will be done by multiple calls to the required API functions. Such software involves four main functions:

- *ISIS_Open*
  It performs the database open functions and makes all the workareas available.

- *ISIS_Search*
  It executes the document selection using a linear infix query and stores the selected documents into a temporary storage disk file as an occurrence list. It means that all the pointers to the selected keywords, belonging to the selected documents, are stored into the temporary storage.

- *ISIS_Get_Rec*
  It is able to get from the document file (CDS/ISIS Master File) the selected document. Any document is identified by an ordinal number (Master File Record Number) starting from 1. We will name this number as the CDS/ISIS Document Identification (ISIS_Doc_Id).

- *ISIS_Format*
  It performs the printout of the selected document just available via the ISIS_Get_Rec function. The printout of the document is done via a print formatting language which is able to define the fields to be printed and the way they must be printed. The print formatting language commands are stored in a disk file named ISIS_Print_Format. In the same installation more than one ISIS_Print_Format for any database may exist.

Since the CDS/ISIS functions work as a command driven system a private set, for any database and for the same database for any query, must be defined and used to store the CDS/ISIS Document List and the related

CDS/ISIS Query. The private set is identified by a name assigned by the ARCA target. The private set is known as ISIS_Result_Set.

The ISIS_Result_Set is composed by two different types of informations:

- The Document List: which contains the ISIS_Doc_Id of any document matching the query.
- The Query: which includes the related query, the real CDS/ISIS database name and the ARCA_ISIS_Database_Name. The real database name is the name of the database known by CDS/ISIS and is mapped from the ARCA-ISIS database name via a gateway interface.

6.2 ARCA-ISIS Interface

6.2.1 The supporting structures

The ARCA-ISIS database gateway interface is stored into a file known by the ARCA-ISIS-APIs. It contains informations like:

a) the real CDS/ISIS database name;

b) the CDS/ISIS database working directory;

c) the CDS/ISIS database working directory minimum disk-space required;

d) the maximum number of documents to be retrieved;

e) the default print format name to be used;

f) the list of the available formats.

The default format and the list of available formats are used to map the SR record syntax and the SR record composition into the corresponding ISIS_Print_Formats.

6.2.2 The ISIS API Functions

- *OPAC.Authentication/Termination*
  CDS/ISIS Information Retrieval system does not yet support authentication, therefore, such APIs receives the user info data and always returns TRUE.

- *OPAC.Non_RPN_Search*
  After loading the database gateway interface, it executes the database selection (ISIS_Open); then, it parses the query in order to find the referenced ISIS_Result_Set and replaces them with the corresponding query. The resulting query is submitted to the ISIS search function (ISIS_Search) which returns in the CDS/ISIS temporary storage the

occurences list needed to compute the relative ISIS_Doc_Ids. The ISIS_Result_Set is then generated by storing the document ids in the ISIS_Document_List and the query in the ISIS_Query. The ISIS-Non-Rpn-search also cleans up the temporary storage disk file required to perform the search and finally returns the number of document selected in ad-hoc parameters.

- *OPAC.Present*
  It extracts the real database name from the ISIS_Result_Set, and after loading the database gateway interface, it loads the document identifier (ISIS_Doc_Ids) stored in the ISIS_Document_List belonging to the ISIS_Result_Set. Then, it opens the database (ISIS_Open) and for every requested document identifier loads the corresponding document (ISIS_Get_Rec) and formats it according to the specified syntax (ISIS_Format).

- *OPAC.Delete*
  For each of the ISIS_Result_Set indicated it locates the set and deletes it.

7 SR Target SABINI Interface

This section describes the interface between ARCA and the SABINI OPAC. The first part of the section deals with the principal functions of the SABINI OPAC.

7.1 SABINI OPAC

SABINI is a library automation system developed in Spain. SABINI works under the UNIX operating system, together with additional applications development software (UNIVERSE, UNIDATA, etc.) . Standard UNIX files and facilities are used, allowing full and easy access to all UNIX comands, to communications products supported by the manufacturers, and to all other applications based on UNIX. The SABINI system involves the following main functions:

- *SABINI_Open*
  Performs the initialization functions, taking into account the database selected, inquiry language and working file assigned. It should be remembered that SABINI may be used to consult one or several databases, and that it is a multilingual system.

- *SABINI_Select*
  Selects the bibliographic references in the database which contains the term indicated in the query. This set of references, identified by a number, is stored in the working file; only the references' identifiers are kept.

- *SABINI_Combine*
  Executes the Boolean operation indicated in the query between the selected sets, generating a new set of references which is also stored in the working file, as in the previous function.

- *SABINI_Limit*
  Limits the set selected, keeping only those bibliographic references which fulfil the conditions for edition dates, country and language indicated in the query. It also generates a set of references which is stored in the working file.

- *SABINI_Format*
  Locates the documents selected in the database, and converts them to the edition format requested. Two types of format may be requested in the SABINI system: ISBD and MARC. Full or partial information may be included in both cases.

- *SABINI_Browse*
  Displays a list of the terms in the database in alphabetical order, starting from a given term. It is possible to select one of these terms.

- *SABINI_Consult*
  Displays the terms related to a given term, according to the structure of the Thesaurus in the database. It is possible to select one of these terms, thus facilitating navigation among the Thesaurus trees.

In short, a query in SABINI consists of various fundamental queries (Select, Combine, Limit) which are called "commands", and whose result is a set of references whose identifiers are stored in a working file. This file is designated SABINI_Result_Set, and also includes the query itself.

The Browse and Consult functions are used as an aid to locate the query term in the function SABINI_Select.

7.2 ARCA_SABINI Interface

7.2.1 The supporting structures

a) List of available databases
b) Default database name to be used
c) List of available languages
d) Default language to be used
e) Maximun number of Result_set names
f) Maximun number of documents to be retrieved
g) List of available print format

7.2.2 The SABINI API Functions

- *OPAC.Authentication*
  The SABINI system does not yet support authentication, therefore, such APIs receives the user info data and always returns TRUE.

- *OPAC.RPN_Search*
  After loading the database gateway interface, it executes the initialization function (SABINI_Open), then analyses the query and

breaks it down into commands which are executed with the SABINI functions (SABINI_Select, SABINI_Combine, SABINI_Limit). The result is kept in the specified SABINI_Result_Set.

- *OPAC.Present*
  Locates the records belonging to the SABINI_Result_Set in the database, taking into account the starting point and number of records requested, and executes the SABINI function (SABINI_Format). The records are formatted in accordance with the specified syntax.

- *OPAC.Delete*
  Given a Result_set_name SABINI deletes the corresponding SABINI_Result_Set

- *OPAC.Termination*
  For each of the Result_set_names indicated in a list, SABINI deletes all the associated SABINI_Result_Set.

8 References

[ANSI] *Information Retrieval: Application Service Definition and Protocol Specification*, ANSI/NISO Z39.50-1995, April 1995

[ARCA/T11/SRD] *ARCA Target System Requirements*, Report ARCA/T11/SRD, July 1995

[ARCA/T12/ABD] *Analysis of BIB-1 Mapping*, Report ARCA/T12/ABD, September 1995

[ARCA/T22/ADD] *User Interface Application Design Document*, Report ARCA/T22/ADD (in progress).

[CDS/ISIS] *CDS-ISIS Versione 3.0 per Mini e Micro Computer. Manuale d'uso*. Firenze: Titivillus Edizioni, 1992

[ISOa] International Organization for Standardization (ISO): *Information and Documentation — Open Systems Interconnection —Search and Retrieve Application Service Definition*, International Standard ISO 10162 (1993)

[ISOb] International Organization for Standardization (ISO): *Information and Documentation — Open Systems Interconnection — Search and Retrieve Application Protocol Specification*, International Standard ISO 10163-1 (1993)

[PARAGON] PARAGON, Project LIB-SR-TARGET/2-3034. WP 200-Task 230: Target Requirements \Analysis. 20 May 1995

[Rumbaugh et al.] Rumbaugh J., Blaha M., Premerlani W., Eddy F. and Lorensen W.: *Object-Oriented Modelling and Design*, Prentice-Hall International, 1991

[SABINI] *Automatizacion de Bibliotecas. Version 3.0. Manual de Uso*. Madrid, Sabini, 1994

[StP/OMT] *Object Modelling Technique. Creating OMT Models*, Software through Pictures Release 2, IDE (Interactive Development Environment)

[YAZ] *YAZ User's Guide and Reference*, Index Data

9 Definitions and acronyms

ANSI	American National Standards Institute
APDU	Application Protocol Data Unit
API	Application Program Interface
ARCA	Access to Remore Catalogues
ASN.1	Abstract Syntax Notation One
CASE	Computer Aided Software Engineering
DLL	Data Link Library
GUI	Graphical User Interface
ISO	International Organization for Standardization
ISODE	ISO Development Environment
MARC	Machine Readable Cataloguing
OMT	Object Modelling Technique
OPAC	Online Public Access Catalogue
OSI	Open Systems Interconnection
RPN	Reverse Polish Notation
SR	Search and Retrieve
StP/OMT	Software through Pictures/OMT
TCL	Tool Command Language
TCP/IP	Transport Control Protocol/Internet Protocol
UNIMARC	Universal MARC
USMARC	United States MARC
YAZ	Yet Another Z39.50 Toolkit

Appendix A: OMT design methodology

Object Modelling Technique (OMT) [Rumbaugh et al.] is an object-oriented methodology which supports the entire software life cycle. It requires to build a starting model of the application domain and, subsequently, refine it during the design phase. This methodology comprises the following stages:

- *Analysis*. This stage is concerned with understanding and modelling the application domain. The output from analysis is a formal model that captures the essential aspects of the system: the objects and their relationships, the dynamic behaviour of each object and the functional transformations of the data.

- *System Design.* The overall architecture of the system is established during this stage. Using the object model as a guide, the system is organized into subsystems.

- *Object Design*. During this stage, there is a shift in emphasis from application concepts towards computer concepts. The analysis models are elaborated, refined, and then optimized to produce a practical design. First the basic algorithms are chosen to implement each major function (method) of the system. Based on these algorithms, the structure of the object model is then optimized for efficient implementation.

- *Implementation*. The object classes and relationships developed during object design are finally translated into a particular programming language.

During all these stages modelling is the main activity. To support appropriately this task three models are provided; each model represents related but different viewpoints, each capturing important system aspects.

- *Object model.* It captures the static structure of a system by showing the system objects, the relationships between these objects, and the attributes and operations that characterize each class of objects.

- *Dynamic model*: It describes those aspects of a system which deal with flow of control, interactions and sequences of operations.

- *Functional model.* It describes the computations within a system by specifying how output values in a computation are derived from input values, without regard for the order in which the values are computed.

Each model is equipped with an intuitive representation which facilitate greatly its understanding and the communication with the customer.

The three above models are not completely independent but each model can be examined and understood by itself to a large extent since the interconnections between the three models are limited and explicit.

The following sections describe the above models in more detail.

A.1 Object Model

The Object-Oriented approach to the development of system focuses first on identifying objects from the application domain, then fitting procedures around them. As a consequence of this approach the object model is considered the most important of the three OMT models.

The object is the main modelling mechanism of the object model. An object is characterized by an identity which distinguishes it from the others.

A class describes a group of objects with similar *attributes*, common *operations*, and common *relationships* to other objects.

An attribute is characterized by a name which is unique within a class. Each attribute has a value for each object instance, this value may vary on time. The values admitted are pure data value, i.e. they cannot be objects.

An operation is a function or transformation that may be applied to or by objects in a class. The same operation may have different implementations (*methods*) in different classes. An object "knows" its class and hence the right implementation of the operation. When an operation has methods on several classes it is important that the methods all have the same signature, i.e. the number and types of arguments and the type of the result value must be the same.

A set of *constraints* restricts the possible values of attributes and operations. Each constraint is expressed under the form of a conditions to be satisfied.

A graphical notation is provided to express object models. Figure A.1 shows an *object diagram* composed of a *class diagram* (left) and an *instance diagram* (right). A class diagram is drawn as a box which may have as many as three regions. The regions contain, from top to bottom: class name, list of attributes, and list of operations. Attributes and operations may or may not be shown; it depends on the level of detail desired. If they are shown, a “+” or “-” symbol indicates, respectively, whether they are public or private. A public property can be referred by the other classes, whilst the private one can be referred only by the owner class. Constraints on the class components are written under the class box. An instance diagram is a round box which contains a value for each attribute.

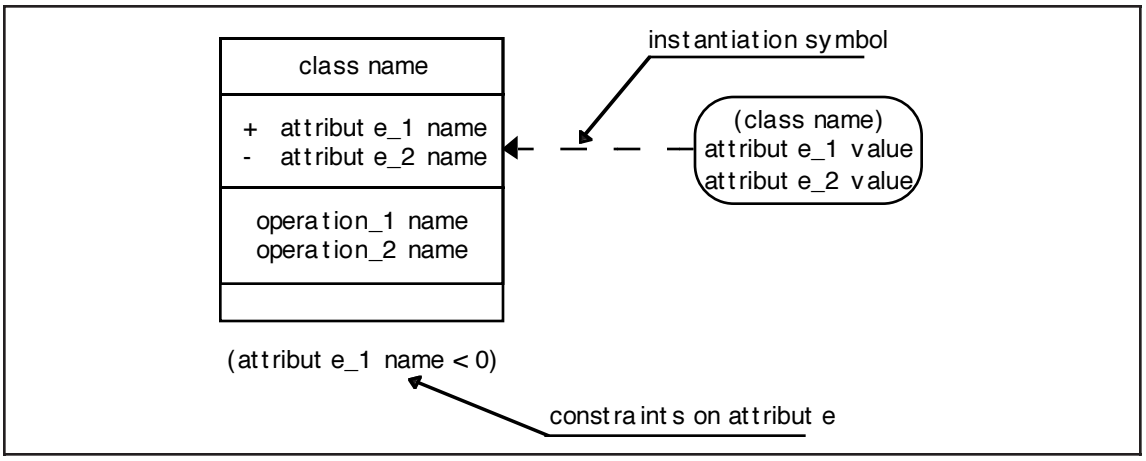


Figure A.1- An Object Diagram

Multiplicity constraints, which specify how many instances of a class may relate to a single instance of an associated class, can be expressed graphically by using appropriate line terminators. The possible line terminators are shown in figure A.2.

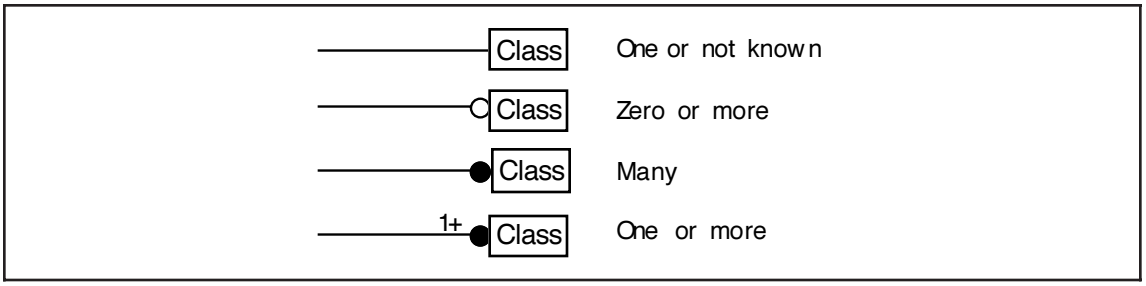


Figure A.2 - Multiplicity constraints

Association is a modelling mechanism which describes a group of links between objects. All the links in an association connect objects from the same classes. Associations may be binary, ternary, or higher order; independently from the arity they are inherently bi-directional; this means that, for example, even if the name of a binary association reads in a particular direction, the binary association can be traversed in either directions. Figure A.3 shows how binary association are represented in a class diagram. Each association in a class diagram corresponds to a set of links in the instance diagram.

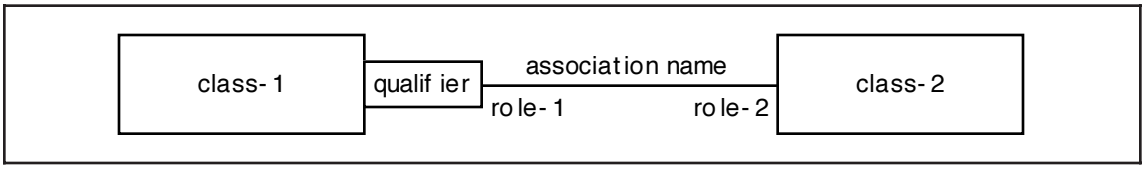


Figure A.3 - Binary association

Several modelling mechanisms are provided to enrich the semantics of an association. Roles and qualifiers are two of these mechanisms.

Role can be attached to a binary association. Each role identifies uniquely an object, or a set of objects, associated with an object to the other end.

A *qualifier* is a special attribute that distinguishes among a set of objects at the many end of an multiple association.

Aggregation is a particular case of association relationship. This relationship, often called *part-of*, links an aggregate object to its component objects. The component objects may or may not exist apart from the aggregate, in addition they can appear in multiple aggregates.

Figure A.4 shows the graphical representation of an aggregation relationship.

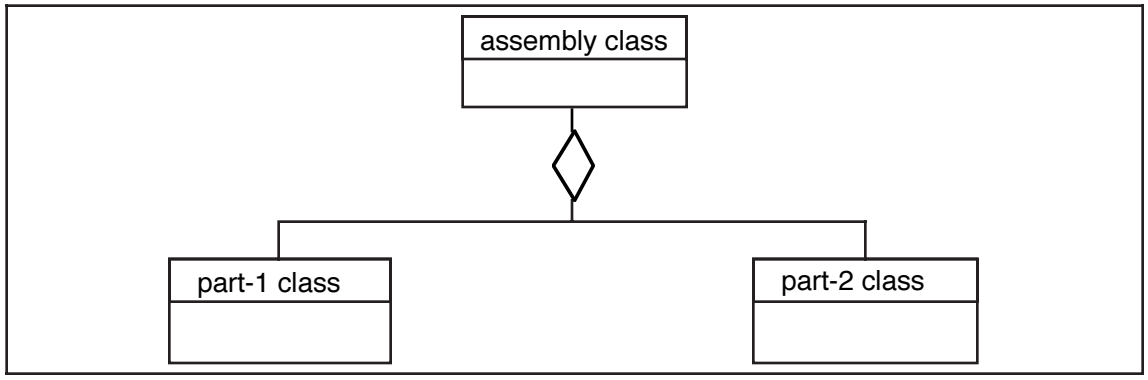


Figure A.4 - Aggregation

Another particular kind of association relationship is *generalization*. This relationship links a class to more refined versions of it. The class being refined is called *superclass* and each refined version is called *subclass*. Attributes and operations common to a group of subclasses are attached to the superclass and shared by each subclass. Each subclass is said to *inherit* the features of its superclass. Each subclass can add to the features inherited its own specific attributes and operations.

An instance of a subclass is simultaneously an instance of all its ancestor classes. As a consequence, any operation of any ancestor class can be applied to an instance of the subclass.

Figure A.5 shows a graphical representation of the generalization relationship. A triangle connects a superclass to its subclasses. The superclass is connected by a line to the apex of the triangle. The subclasses are connected by lines to horizontal bar attached to the base of the triangle.

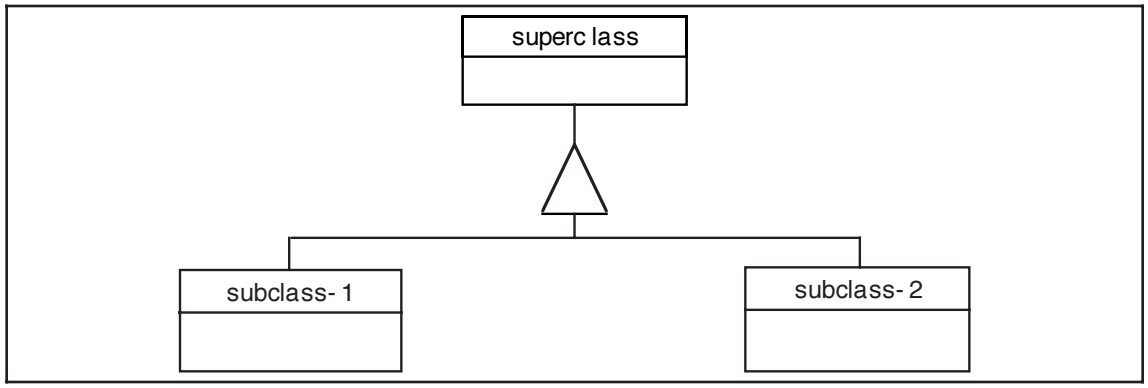


Figure A.5 - Generalization relationship

A.2 Dynamic Model

The dynamic model describes the sequencing of operations that occur in response to external stimuli, without consideration of what the operations do, what they operate on, or how they are implemented.

This model is built taking into account the scenarios of typical interaction sequences between user and system. These scenarios show the sequences of events that occur during one particular execution of a system. A diagram, called *event trace*, represents these ordered list of events and the objects exchanging events.

Figure A.6 shows how an event trace diagram is drawn. This diagram shows each object as a vertical line and each event as an horizontal arrow from the sender object to the receiver object with time increasing from top to bottom.

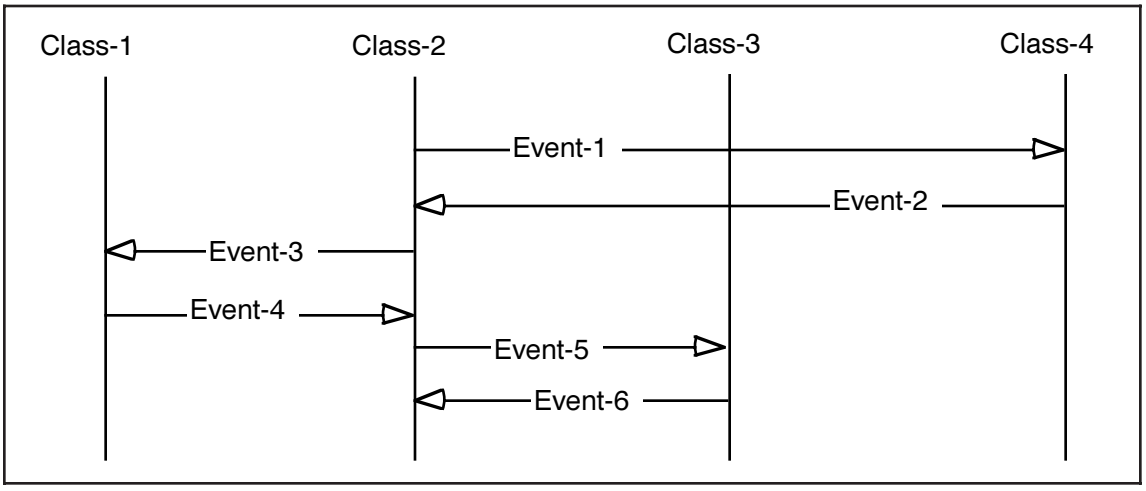


Figure A.6 - An event trace

The dynamic model consists of multiple *state diagrams*. A state diagram is a graph, whose nodes are states of an object and whose directed arcs are state transitions of the same object labelled by event names. These diagrams specify the valid patterns of state transitions for an object, i.e. the events for each state together with the transition triggered by that event. The state diagrams for the various classes combine into a single dynamic model via shared events.

Figure A.7 shows how state diagrams are depicted. A state is drawn as a rounded box containing an optional name. A transition is drawn as an arrow from the receiving state to the target state; the label on the arrow is the name of the event causing the transition.

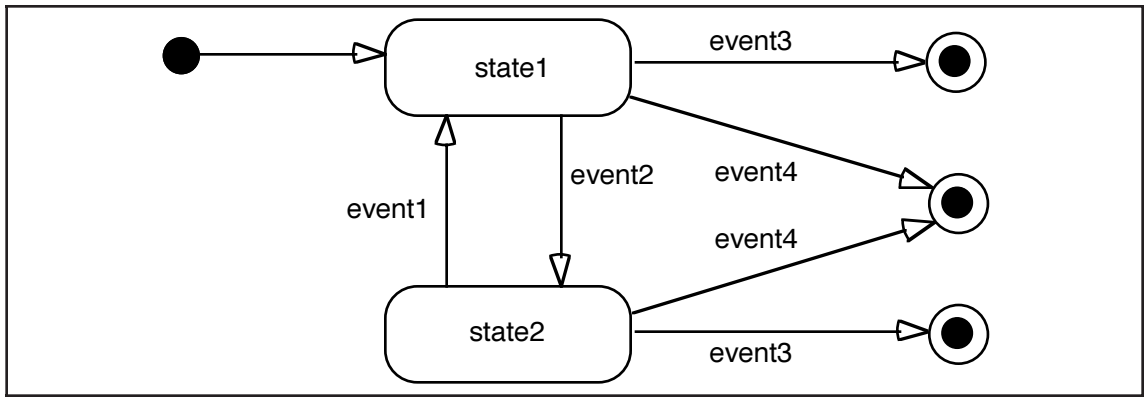


Figure A.7 - A state diagram

The meaning of a state diagram is the following. If an object is in a state and an event labelling one of its transitions occurs, the object enters the state on the target end of the transition. If more than one transition leaves a state, then the first event to occur causes the corresponding transition to fire. If an event occurs that has no transition leaving the current state, then the event is ignored. A sequence of events corresponds to a path through the graph.

Transitions can be enriched by *conditions* which works as guards. A guarded transition fires when its event occurs, but only if the guard condition is true.

Operations can be attached to states or transitions to specify what the object does in response to events. Operations are represented as either actions or activities.

An *action* models an instantaneous operation which can be attached to transitions or to entering or exit a state. One kind of action is sending an event to another object.

An *activity* is associated with a state and represents a sequence of actions that takes time to complete. The sequence of actions begins on entry to the state and stops when completed.

Figure A.8 shows the graphical notation for conditions, actions and activities. A condition may be listed within square brackets after an event name. An activity is indicated within a state box by the keyword "do:" followed by the name or description of the activity. An action is indicated on a transition following the event name by a "/" character followed by the action name. All these constructs are optional.

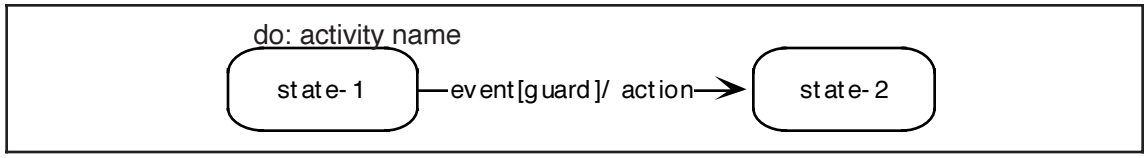


Figure A.8 - Condition, Action and Guard

States may have substates that inherit the transitions of their superstates, just as classes have suclasses that inherit the attributes and operations of

their superclasses. Any transition or action that applies to a state applies to all its substates, unless overridden by an equivalent transition on the substate. Substates in turn may enclose other substates. Figure A.9 shows a superstate. It is drawn as a large rounded box enclosing all of its substates and the transition among them.

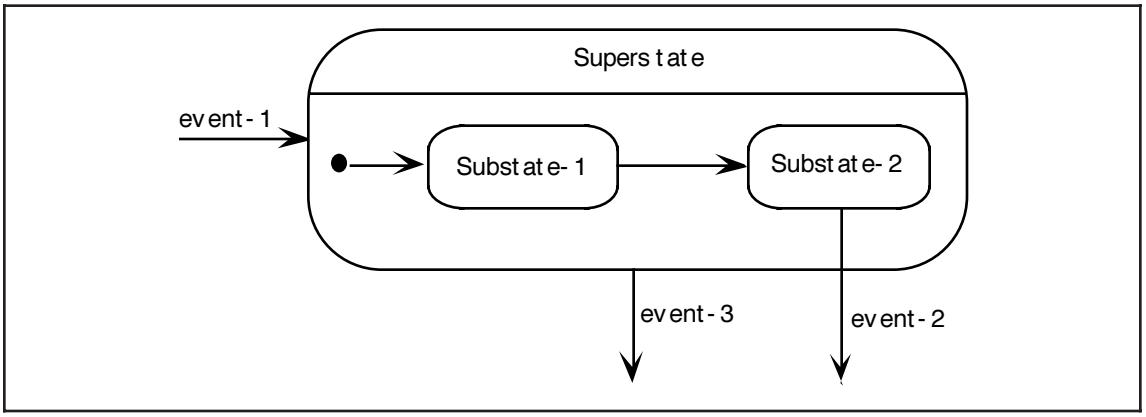


Figure A.9 - State generalization

A.3 Functional Model

This model describes the computation within a system. It shows the functional relationships of the values computed by a system including input values, output values and internal data stores, without regard for how and when they are computed.

The main modelling mechanism of the functional model is the *data flow diagram*. A functional model is actually a set of data flow diagrams which describe the meaning of the operations in the object model and the actions in the dynamic model, as well as any constraints in the object model.

The data flow diagram is constructed in terms of other modelling mechanisms: *processes* that transform data, *data flows* that move data, *actors* objects that produce and consume data, and *data store objects* that store data passively. Figure A.10 shows the graphical notation for data flow diagrams. A processes and data stores are drawn as an ellipse containing a description of the transformation, usually its name; data flow are drawn as arrows between the producer and the consumer of the data value labelled with the description of the data, usually its name or type; finally, actors are drawn as a rectangle to show that they are objects.

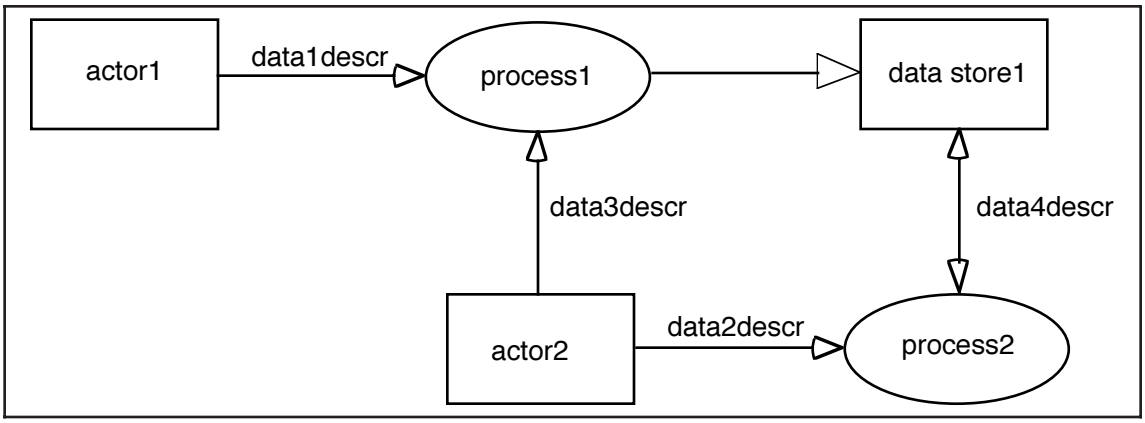


Figure A.10 - A data flow diagram

Each process in a data flow diagram has a fixed number of input and output data arrows, each of which carries a value of a given type. The inputs and outputs can be labelled to show their role in the computation, but often the type of value on the data flow is sufficient. An high level process can be expanded into an entire data flow diagram. Each input and output of the process is an input or output of the new diagram. The new diagram may have data stores that are not shown in the higher level diagram. Diagrams can be nested to an arbitrary depth, and the entire set of nested diagrams form a tree. Eventually the nesting of diagrams terminates with simple functions. These functions can be specified in various ways, including mathematical functions, table of input and output values for small finite sets, pre e post conditions. pseudo-code and natural language.

An actor is an active object that drives the data flow graph by producing or consuming values. Actors are attached to the inputs and outputs of a data flow graph: it lies on the boundary of the data flow graph but terminates the flow of data as sources and sinks of data. A data store is a passive object within a data flow diagram that stores data for later access. It merely responds to requests to store and access data.

A data flow diagram is particularly useful for showing the high-level functionality of a system and its breakdown into smaller functional units. A process can be expanded into another data-flow diagram.

A.4 StP/OMT

StP/OMT is a set of tools that provides an UNIX based environment for defining object-oriented models according to the Object Modeling Technique (OMT), and generating code in several languages (C, C++, Ada, Smalltalk). StP/OMT is a product from Interactive Development Environments (IDE). Main features of StP/OMT are:

- *Object Model Diagram Editor*

  It provides a symbol palette, menu and display marks, for drawing object model diagrams using OMT notation.

- *Class Table Editor*
  It allows the designers to capture the full definitions of methods and attributes for each class in the model.

- *Use Case Editor*
  It is used to develop representations based on user-centered analysis, and integrate them with the rest of the model. In particular it's possible to draw use case scenarios, event trace diagrams, event flow diagrams, and context diagrams.

- *Dynamic Model Diagram Editor*
  It permits to create state diagrams for those classes in the object model, who have a relevant dynamic behavior. The dynamic model is described according to Harel State Transition notation.

- *State Table Editor*
  It completes the definitions of states, actions and activities required by the dynamic model.

- *Functional Model Editor*
  It sketches the behavior of a single class operation without regard for time or sequencing between events. It allows to generate data flow diagrams showing objects as data that flow into and out from processes. Such a editor uses symbols and display marks provided by De Marco/Yourdon and Rumbaugh notations.

- *Object Annotation Editor*
  It is claimed at adding comments and information on diagrams and objects defined with the previous mentioned tools. It is also used for adding code or pseudo code to implement functionalities of the system being modeled.

- *Script Manager*
  The script manager is used for checking consistency and generating code from the definitions given by the editors.

StP/OMT assures several cross-checks among the various models of the entire project to validate the consistency and completeness. As far as code generation is concerned (C++ code in our case), we will obtain for each class in the model:

- the interface files including all the class declarations, definitions and, optionally, inline member functions;

- the implementation files containing include statements, member functions headers, bodies, and, if supplied during the previous development phases, the C++ code annotations.

Appendix B: YAZ

At the outset of the ARCA project, it was intended to make use of the ISO Development Environment (ISODE) toolset as the implementation basis of the ARCA target software. The ISODE environment provides the possibility to host OSI upper layer applications over the TCP/IP suite.

However, problems surfaced with the ISODE suite during the initial phase of the project. ISODE is distributed in two versions, one of which is public domain and free of cost. The public domain version is also entirely unsupported. This was considered to be a major disadvantage by the ARCA participants. The other version of ISODE is supported by the ISODE Consortium, for which membership at several levels is possible. It was discovered that commercial/industrial membership costs several tens of thousands of dollars per year. (One reason for the high cost of membership is that ISODE is a comprehensive set of tools for an entire range of OSI applications, with special emphasis on those used by large corporations—message handling, file serves, directory services, and terminal services, as well as network management services.) For the relatively restricted aims of the ARCA project, this was considered to be a prohibitively high price to pay. Therefore a search was begun for alternative support software.

The CANSEARCH software from the Canadian National Library Service was also examined. However, it also is not supported officially, and only handles Z39.50 running over TCP/IP (rather than OSI), and therefore did not entirely fulfill the requirements of the ARCA project.

Finally, in the Spring of 1995, the decision was made to adopt a toolkit that had recently made its appearance in Europe. *YAZ* (Yet Another Z39.50/SR Toolkit) is a toolkit, developed by Index Data in Copenhagen, Denmark, that provides basic functionalities to implement Z39.50 and SR target systems.

The basic level consists of three primary modules:

- *ASN*
  This is a set of C representations for all of the APDUs defined in the Z39.50 and SR protocols according to the ASN.1. Moreover, for each APDU type, a function is provided that allocates and initializes an instance of that type.

- *ODR (Open Data Representation)*
  This is a mechanism for representing a new ASN.1 type and implementing the encoders/decoders Basic Encoding Rules (BER) for that type.

- *COMSTACK*
  This provides an interface for establishing a connection endpoint and exchanging BER encoded data over the TCP/IP or OSI transport layers.

The three modules are independent and can be arranged by the user according to his/her own requirements and future enhancements of the protocol services.

The following considerations motivated our choice of the YAZ system over the other candidates:

- The toolkit is in the public domain, with no royalties or similar restrictions on its use.

- It is reliable and documented (This is less so for several other packages.).

- It is fully supported by its supplier, with a listserver for users as well as direct support for any questions, installation problems, or bug fixes.

- It has encoder-level support for most, if not all, of Z39.50-1995, allowing ARCA to depart immediately from the latest protocol version.

- It is the only package that includes a transparent interface to both the OSI upper-layers and TCP/IP, as well as Z39.50 and SR (the latter point will be void when the two are merged, although the backwards-compatibility will still be useful.)

- There is a counterpart toolkit (called IrTcl) for the client side available from the same suppliers, and which entirely fulfills the requirements for the ARCA client implementation. Although it is not strictly necessary to use base toolkits from the same supplier for both target and client implementations, it undoubtedly makes cooperation and coordination of the work easier.

Detailed information about YAZ are available at the following address:

ftp.algonet.se/pub/index/yaz/