SAP Remote Communications

A.Selmeci, T. Orosz
Óbudai Egyetem AREK, Székesfehérvár, Hungary
E-mail selmeci.attila@arek.uni-obuda.hu; orosz.tamas@arek.uni-obuda.hu

***Abstract*—SAP implemented a very deep level synchronous connection type based on industry standards, but early enough, in the first releases SAP enhanced this to Remote Function Call protocol and technology. The Remote Function Call technology opened the system to other SAP systems or external applications. Remote function call (RFC) is the fundamental technique for many different communication approaches within SAP solutions. Our document tries to uncover the different capabilities and usability of the SAP communication types like CPI-C, RFC, Q-API, tRFC, etc. In our study we figured out that main points of remote communication usage in a heterogeneous environment where SAP also in place are the business level communication from other SAP system or external application via business application programming interface (BAPI), and the loosely coupled distributed system environment, the ALE (application link enabling) communications based on RFCs.**

Keywords: CPI-C, RFC, BAPI, QAPI, ERP, Remote communication

I. INTRODUCTION

SAP is the market and technology leader in business management software, solutions, and services for improving business processes. Former IBM engineers founded SAP in 1972. The first real-time, integrated solution from sap was launched in 1973 as SAP R/1, where the 'R' stays for real-time computing. In this solution the presentation, application logic and the database layer were running on a single monolith machine. After some years SAP launched the SAP R/2, where the architecture was redesigned according to the market requirements giving two-tier solution where the presentation was moved to terminals of the IBM mainframe. The R/2 system was a very stable environment, but in mid-1990s SAP had to follow the IT paradigm shift from mainframe to the client-server architecture. With the client-server architecture SAP changed the name as well to show the three-tier layered architected by R/3. This architecture switch to he client-server widened the range of platforms to be used. The basic layer of the system remained the RDBMS (Relation Database Management System) giving a central point of the SAP system. The database contains everything in an SAP system beyond the business data (master- and transactional date), the so called repository (developed objects, like programs, screens, menus, tables structure definitions, etc.) and the configuration (customizing table entries) are also stored in the single database. The database could be managed by different RDBMS, like Oracle, DB2, earlier Informix, MSSQL, MaxDB and SyBase (for the new releases). The first database systems run on UNIX and VMS system, but after a while MS Windows (NT) extended its notability primarily by the MSSQL database. Later different Linux-distribution took part from the SAP platform cake as well. SAP decided early in the beginning that no database level communication would lead to a good result, so the application level integration was the main focus in communication. Of course not only the system wide integration was interesting, but the communication between the system and different clients or client application.

The SAP communication possibilities did not stop at the system-system communication or at the companies' border, but the Internet strategy mySAP.com extended the communication and collaboration requirements and opportunities. The first solution used additional techniques like ITS (Internet Transaction Server) for Web application or IACs (Internet Application Components), and SAP BC (Business Connector) to communication through http (using e.g. xml data).

The new generation of SAP communication direction was developed under the flag NetWeaver delivering an integration and application platform with many integration layers and components. The basis of the application platform is the SAP Web Application Server, which is the SAP's further improved application server technology containing and supporting the modern and accepted, standard protocols, like smtp, http, webdav, soap, etc. This direction opened the way to build not only cross-company collaborations, but service oriented applications and composites as well.

II. CPI-C COMMUNICATION

During the 1980s, while SAP brought to the market the redesigned mainframe based R/2 system, IBM developed the Common Programming Interface for Communication (CPI-C) as a platform-independent communication interface and a standard API for the SNA LU 6.2. The earlier IBM communication protocol, called Logical Unit 6.2 is part of the device independent System Network Architecture (SNA) protocol, which is a peer-to-peer communication between systems or devices, like terminal, printer. The SNA LU 6.2 provides an API for communicating with the above-mentioned partners, like system, terminal or other devices. Unfortunately other vendors did not correctly implemented IBM's protocol and API e.g. by modified implementation or offering different APIs. This led IBM to define the "Common Programming Interface for Communication" API. The CPI-C API allows the creation of multi-platform code, and with it provides a widely accepted and implemented surface in the peer-to-peer communication. As other vendors, SAP has also implemented the CPIC to enable

978-1-4673-1014-7/12/$31.00 ©2012 IEEE

the mainframe based R/2 systems to communicate with other system providing business information. The SAP implementation supports the later by X/Open adopted specification as an open standard.

The main point from SAP was the communication between mainframe and UNIX machines, which communicate with TCP/IP protocol. The R/2 systems did not contain e.g. HR (Human Resources) or as the new era call it HCM (Human Capital Management) solution, but in the newer technology, in R/3 system SAP introduced the HR/HCM module. If a company wanted to use R/2 as the main financial and logistic system, the HR as an R/3 system had to be connected to them to communicate the cost center information from the R/2 and to retrieve the payroll date to pay for the employees from the R/3. The two systems communicated using the CPI-C communication channel, because it has various protocols, like SNA LU 6.2 and TCP/IP.

SAP provides platform-specific libraries for CPI-C communication as CPI-C subsets in C programming language. For the earlier R/2 systems the cpicslib for SNA and of course for the newer systems the cpicslib for TCP/IP are delivered. Before using any CPI-C communication we should decide between two usage sets: CPI-C started set and the advanced function calls.

The starter set contains very limited number of commands, but these are enough to set the communication channel, communicate and close it. The function calls are the following:

- Setting up the connection:
  - CMINIT (Initialize connection, with connection parameter defined by side information table)
  - CMALLC (Allocate conversation, this tries to build up the logical session)
  - CMACCP (Accept conversation, the remote program sends the starting request)
- Information exchange (communication):
  - CMSEND (Send data)
  - CMRCV (Receive data, this call receives the status messages as well)
- Close communication:
  - CMDEAL (Deallocate conversation)

Each of the calls can be managed with parameters. As an example the send data call (CMSEND) contains information beyond the data itself (stored in buffer), the conversation id, the length of the data, whether the partner program will send back data or just receives our one, and of course a return code.

The advanced set contains conversion calls for ASCII and EBCDIC, and security call if the R/2 system uses external security system.

All these function calls are defined in a C header file (`cpic.h`), which can be used if the CPI-C library is applied for communication. There are some additional SAP-specific CPI-C functions, which give more parameter to a SAP CPI-C communication interface. These functions are not part of the standard CPI-C function set, so they can not set parameters for them. As a code snippet the following lines show a C program example:

```
main (int argv, char ** argc)

CM_RETCODE return_code;
CONVERSATION_ID convid;
..
SAP_CMACCP(argv);
CMACCP(convid,&return_code);
CMRCV(...);
CMSEND(...);
CMDEAL(...);
```

The SAP R/3 systems are not communicating in C level language, but on ABAP level the logon data should be built to the target system and if it is needed the data is converted to EBCDIC (e.g. for an R/2 system). The side info table as mentioned earlier must be given as well, but in an SAP system it is stored in the TXCOM table (maintained by transaction SM54) on the caller side. To be able to use the ABAP layer for communication programs SAP implemented the CPI-C starter set within the ABAP language. The above-mentioned six CPI-C calls are implemented in a single statement:

```
COMMUNICATION comstep ID id [cpic options].
```

Here the ‹comstep› stands for the CPI-C calls as communication steps, but the names of the steps are descriptive as always in the ABAP language: INIT, ALLOCATE, ACCEPT, SEND, RECEIVE, DEALLOCATE. The ‹id› identifies the actual connection. The ‹cpic_option› is different according to the call. In case of INIT call the option contains the **DESTINATION ‹d›** closure as well to give a symbolic name of the communication partner. This symbolic name is used in the side info table as well.

To define in an ABAP program the communication partner we should use a logon structure, which contains beyond the security information (client, user, password, language), the partner program name and the form routine name to be called. After allocating the connection the caller should send this information, but the first part of the information structure is the logon header reserving 12 characters. The first 4 characters are important at the beginning of the communication, because it contains the so-called request ID. This request ID can be from the caller side **CONN**, which means I want to connect, log on. The partner will answer with the same structure of logins string, but the header contains **APPC**, if the acknowledgement is positive, or the partner sends back **FREE** with the meaning of a termination message. The next 4 characters are always in case of CPI-C communication ‘CPIC’ and the mode number is 1 as default. So the standard sender login header contains ‘CONNCPIC1’ and the positive answer contains ‘APPCCPIC1’.

The simple data declaration looks like this:

```
DATA:   d       TYPE c LENGTH 8 VALUE 'PARTNER',
        id      TYPE c LENGTH 8,
        BEGIN OF conn msg,
         header(12)   TYPE c VALUE 'CONNCPIC1',
         client(3)    TYPE c VALUE '100',
         user(12)     TYPE c VALUE 'SELMACI',
         password(8)  TYPE c VALUE 'ALMA11',
         language(1)  TYPE c VALUE 'E',
         corr(1)      TYPE c VALUE ' ',
         program(8)   TYPE c VALUE 'ZMCPIC R',
         routine(30)  TYPE c VALUE 'REM_CPIC_FORM',
        END OF conn_msg,
        conn_send TYPE x LENGTH 75,
        conn_recv TYPE x LENGTH 75,
```

```
      buf       TYPE c LENGTH 10,
      length    TYPE x LENGTH 4,
      data_inf  TYPE xstring,
      status    TYPE xstring,
      rc        LIKE sy-subrc.
```

In the above declaration snippet the destination (d) is defined and the connection structure is filled with data. The SEND and RECEIVE calls communicate the data in hexadecimal form, so before and after the call the `conn_msg` structure must be converted to `conn_send` and from `conn_recv` if we want to use them. The connection structure shows the login information and the remote program and form name in the program-to-program communication. The real calling test program execution part is the following:

```
* Example for constants, which could be defined
CONSTANTS: cm_ok TYPE x LENGTH 2 VALUE '0000'.

COMMUNICATION INIT DESTINATION d ID id
  RETURNCODE rc.
IF rc <> cm ok.
  WRITE:/ 'ERROR: Initialization problem...'.
  EXIT.
ENDIF.
COMMUNICATION ALLOCATE ID id.
* The CONN SEND contains the  (to hexa string
*format) converted communication logon info
COMMUNICATION SEND BUFFER conn send ID id.
* Here accepts the called program (form) the
*communication request and sends back status
COMMUNICATION RECEIVE ID id
  BUFFER conn_recv  DATAINFO data_inf
  STATUSINFO status RECEIVED length.
* The BUFFER returns the received login data
* DATAINFO gives info on data received
* (no, complete, incomplete)
* STATUSINFO returns whether the sent
* authorization was received
* RECEIVED returns the length of read data
... "CONN_RECV conversion to CONN_MSG...
IF conn msg-header(4) = 'FREE'.
  WRITE:/ 'ERROR: Remote program terminated the
connection!'.
  EXIT.
ENDIF.
```

We defined check steps into the test program to show the possible process breaks. The last part of the code contains the data exchange and the closing. We use here the `buf` (buffer) field to send and receive the data itself.

```
* Send some information
buf = 'My Request'.
COMMUNICATION SEND ID id BUFFER buf.
* Receive answer
COMMUNICATION RECEIVE ID id BUFFER buf
  DATAINFO data_inf STATUSINFO status
  RECEIVED length.
WRITE:/ 'Received answer:', buf.
* Closing connection
COMMUNICATION DEALLOCATE ID id.
```

Of course the receiver program (`ZMCPIC_R`) must exist having a form defined under the requested name. The declaration in this example code could be almost the same (without login structure).

```
FORM rem cpic form.
  COMMUNICATION ACCEPT ID id.
  COMMUNICATION RECEIVE ID id
    BUFFER   buf      RECEIVED   length
    DATAINFO data inf STATUSINFO status.
  IF buf = 'MyRequest'.
    buf = 'Answer'.
  ENDIF.
  COMMUNICATION SEND ID id BUFFER buf.
ENDFORM.                    "REM_CPIC_FORM
```

Each of the predefined communication constants for CPI-C programming in ABAP language are collected in an include program, called RSCPICDF. As we have shown here the CPI-C programming in ABAP environment is not so complicated, but for today use it is not sufficient any more because of its low level. The CPI-C served as a synchronous communication channel, but for many communications the partner is not available always and this problem should be managed.

III. Q-API – Asynchronous Data Transfer

As we discussed the data transfer is synchronous from program-to-program using CPI-C communication. So the parties can communication at the same time when both are available simultaneously. In the SAP world many systems are not really critical ones, so they do not run 24 hours a day as the linked e.g. in manufacturing industries. If we use synchronous transfer and the connection is broken, special procedures should be taken into consideration (e.g. recovery in the receiver system as well). If the receiver system could not complete the process (e.g. it is overloaded) and data remains, the same situation occurs on the sender program side as well. The asynchronous communication became a necessary way for solving this issue.

The easiest way to transfer data asynchronously is the buffering on the sender side. It is done in many solutions by applying a temporary storage, which lets the data flow sequentially go through. This sequential storage is usually implemented as a queue. The Queue Application Programming Interface (Q-API) is an SAP solution for asynchronous data transfer. In the SAP implementation the data belonging to a transaction are buffered, queued in database table and send to the target program or systems via synchronous communication layer. SAP provides remote function modules for implementing this communication layer.

After the data is placed in a queue, a scheduled job executes a send program based on CPI-C communication. This driver program (e.g. the standard SAP **`RSQAPI20`** program) reads the queue and sends the data. The remote program is defined in a queue attribute. The program reads the queue and tries to establish the connection to the remote program defined in the attribute. If the connection could not be built up, the program schedules again itself to try it again later (depending on parameters). The data of a Logical Unit of Work (LUW) is deleted from the queue only if the entire data flow belonging together is transferred. If the transfer is terminated during sending the data, the LUW must be rolled back on the remote side as well.

According to this procedure three main queue manipulation modules are defined for creating the queue:

- QUEUE_OPEN: Generates a queue and creates the queue header
- QUEUE_PUT: Place data in a queue. The data is defined as records. One record is a LUW. To really store the data record in the queue a **`COMMIT WORK`** command should follow the module.
- QUEUE_CLOSE: Closing the queue. The closing process schedules the driver program to immediately start if the queue was defined (by

QUEUE_OPEN) with start mode 'A', which means automatic transmission.

Not only automatic (for a defined date and time) queue processing start is possible, but also the driver program can be started manually ('M'), or even after an event was triggered. The event-controlled scheduling does not mean that a special SAP event is created or triggered. The event-controlled means here whether the queue is activated. The standard delivered SAP_QEVENT event is triggered internally with the name of the queue given via event parameter.

The defined driver program and the management tools implemented within the SAP system are using other modules as well:

- QUEUE_GET: Read data from a queue. It is important to use (certainly) before sending the data, because the program should know (read) what to send.
- QUEUE_DELETE: Delete data from a queue. This module is used, when the record (LUW) was successfully sent.
- QUEUE_ERASE: Delete queue is used when not only a LUW, but the whole queue also should be deleted.
- QUEUE_SCHEDULE: Schedule queue processing. This is used when the queue driver program run is defined separately for background processing.

A queue is created by the QUEUE_OPEN module and is identified by a name. The queue attributes are defined during the opening phase. These attributes or parameters store some technical information like security data, and control the queue with processing information. The queue itself contains of elements or blocks, which have a header part for administration and a data part. The header segment is fixed (in length), but the data section is stored transparently in variable length storage. The variable section has of course length limitation on ABAP level (defined as a structure field in the Dictionary), but the data can be divided into many blocks, which are recognizable by the application. These divided queue elements build a queue unit. The data elements in the unit, which belong together as parts of a transaction, have specific sequence ID determining the order.

SAP delivers statements as well for queue handling, like:

```
OPEN QUEUE <queue>
TRANSFER <record> TO QUEUE <queue>
CLOSE QUEUE <queue>
```

As we explained above the modules must conclude by **`COMMIT WORK`** or **`ROLLBACK WORK`**. Both of the standard SAP statements are working for Q-API in an enhanced manner by calling the **`QCOMMIT`** or **`QROLLBACK`** functions. These functions place the transfer data related to transaction in the queue, or remove from the database.

IV. Remote Function Call

As we mentioned above, the CPI-C communication is on a very low logical level, so SAP further improved and simplified the implementation program-to-program of communication processes. In the new method the Remote Function Call (RFC) is a call to another system, where a function module is executed. These calls are made via the RFC interface, which provides the following encapsulated tasks:

- Connection open (setting up the line to the partner)
- Data conversion before passing through
- Data transfer
- Unified error handling
- Closing connection

In this encapsulation model the remaining task for the caller is to provide the parameters for the target program (function module) to complete the communication. In SAP environment one or both sides of the communication is an SAP work process, which executes the ABAP codes. The following Fig. 1 below shows the calling possibilities from the SAP systems.

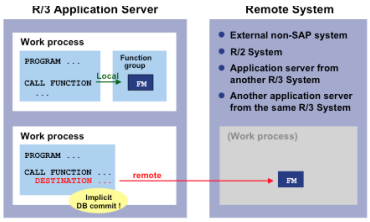

Figure 1 RFC from within SAP environment

RFC interface consist of the following:

- Calling ABAP programs: in the program-to-program communication any ABAP can call a remote function. As the figure above shows, the `CALL FUNCTION <fu.module> … DESTIONATION` statement should be used. The `DESTIONATION` closure tells to the system, where the caller program is running, that the called function module runs in another system. No further ABAP code is needed to set up communication, convert and transfer the data, everything happens as part of the statement. A function module must be configured as remote enabled if a code wants to call it remotely. In special cases remote functions can be called from within the same system as well.
- Calling a non-SAP program: SAP delivers libraries to create external application, which provide functions like an SAP system to be called remotely or consume remote enables functions.

Within SAP systems function modules are modularization units. All programs are made up of processing blocks, which we can be modularized using either local procedures (called forms), locally defined object-oriented methods, global include programs, OOP methods or function modules. Modularization makes ABAP programs easier to read and maintain, as well as avoiding redundancy, increasing the reusability of source code, and encapsulating data.

Function modules are global procedures implementing a special functionality; they are defined in the function library and are collected to function groups. Every function module needs a function group which acts as containers and the function modules belonging to the same function group can share data internally. This shared or (from the function group point of view) global data remains hidden from the caller and can be manipulated only through the function modules belonging to the sap

function group. Function modules also support exception handling, which allows us to catch certain errors while the function module is running.

The SAP systems deliver a wide range of predefined function modules that we can call from any ABAP program. Function modules play an important role beyond the remote communication in updating the databases and in other places in the system.

Each function module has an interface, which contains the parameters for data transfer and the exception definition as well. Parameters are divided to four groups according their duty. From function module point of view we can define IMPORTING, EXPORTING, CHANGING and TABLES parameters. The CHANGING parameter is for parameters, which are sent, but their values can be changed within the called function module. In older systems for remote functions the CHANGING parameters are not allowed. The TABLES parameter is used to handle so-called internal tables (like arrays in other programming languages) during calls. It is used to provide big amount of data. In local function module calls the internal tables are passed on by reference to minimize the data load. It is not possible by RFCs, so SAP uses delta-managing mechanism to minimize network load. If the internal table content is modified in the called function module, the system collects the modified entries and sends back only the delta.

Error handling during function module call is managed by exceptions. The standard (old fashion) style is the EXCEPTIONS part of the interface. The exceptions occurring inside the function module are escalated to the caller as different return codes. During the design time different exception can be defined (e.g. file_name_missing, directory_not_found, etc.). These definitions do not have returning values, but they are used only as ID-s. The returning values are assigned to these ID-s by the caller. If an exception occurs within the function module the execution of the module terminates and the control is given back to the caller by setting the return code (sy-subrc) to the value assigned to the exception ID in the caller program. Newer releases can use object oriented exception classes as well. Remote enabled functions can give back two always provided standard exceptions:

- COMMUNICATION_FAILURE is triggered by the system if the specified destination is not maintained, or if the connection to the remote system cannot be established.
- SYSTEM_FAILURE is triggered if the function module or C routine that we want to start in the remote system is not available.

The RFC is based on the above-explained CPI-C technology, so for partner communication a side info tables should be defined. This table is filled separately in an SAP transaction (SM59) and stored in a database table. The key of each rows the so-called RFC destination as an ID. In the RFC call we give this ID in the DESTINATION closure of the CALL FUNCTION statement. There are different types of RFC destinations depending on the usages. Certainly we have R/2 and R/3 type connections, but for external communication there are TCP/IP and HTTP types also available. The HTTP type connections are used for WebService calls as well. There are so-called beyond some others, logical and internal connection types for special later discussed connections. The function call is the same within an SAP system for the different remote partners even it is another SAP system or an external C server program. The difference is defined in the RFC definition table only.

If an SAP RFC enabled function module is available in the caller system as well (e.g. part of the SAP standard delivery), it is callable locally (without the DESTINATION closure). SAP defines a default RFC-destination for those cases when an RFC-enable function module should be called, but the destination is implementation dependent. For example the component containing the function module (e.g. BAPI_FLIGHT_GETLIST) can be implemented within the same SAP system or in a separate one. The function module is called in the caller program, but the destination parameter (ID in the closure) is determined by customizing or during runtime. The next ABAP snippet shows a calling possibility of a remote enable function.

```
REPORT  zmaci050.
DATA: fli_dest_from TYPE bapisfldst,
      fli_dest_to   TYPE bapisfldst,
      it_flights    TYPE TABLE OF bapisfldat.

fli_dest_from-city = 'FRANKFURT'.
fli_dest_to-city   = 'NEW YORK'.
CALL FUNCTION 'BAPI_FLIGHT_GETLIST'
  DESTINATION 'NONE'
  EXPORTING
    destination_from = fli_dest_from
    destination_to   = fli_dest_to
  TABLES
    flight_list      = it_flights.
```

In the destination field a literal (‘NONE’) is used. This destination is a predefined internal (type) RFC-destination. If we use NONE as the destination the system will call our function module locally. If a real, remote R/3 destination is given, the system will call the same function in the remote system (defined by the RFC-destination) synchronously. As the code shows, a string literal is given as the destination ID. If we use a corresponding string type variable (filled from customizing or by runtime selection) the system will use the content of the variable as the destination ID. With this possibility we can change the communication partner without modifying the code itself.

There is another useful predefined internal RFC-destination called ‘BACK’. As its name explains itself it can be used for calling back to the caller system. Let us suppose that we have a remote enabled function module (FN_A) in system SYA and another remote-enabled function (FN_B) in system SYB. A caller program calls the FN_B function in the partner system and in the called function there is a remote function call to FN_A (defined in caller system) using the destination ‘BACK’. This function call will be executed in the caller system (independent which system called the FN_B function module).

As we discussed earlier with CPI-C the default technology for communication is the synchronous method, but in many cases the asynchronous data transfer is preferred. SAP implemented for this demand the asynchronous RFC (aRFC) technique. With this solution SAP made possible to finish the call after giving the task to the remote system. To understand the mechanism we should know some important operation information:

- SAP uses so-called dialog processes for RFC execution in the remote SAP system

- With the `STARTING NEW TASK '<task_id>'` closure of `CALL FUNCTION` statement we can start a function call in another dialog process

With these two "tool" we are ready to call a remote function asynchronously, because the system starts the function module in the remote system, but immediately gives back the control to the caller program. To have some information on the asynchronously called function SAP extended the functionality with the `RECEIVE RESULTS FROM FUNCTION` statement. In the caller program not only the remote function call is defined with destination and remote task ID, but a FORM definition as well which should be executed after the function execution completed in the remote system. The syntax would be the following:

```
START-OF-SELECTION.
  CALL FUNCTION 'BAPI_FLIGHT_GETLIST'
    STARTING NEW TASK 'DEMO TASK'
    DESTINATION 'REMOTE_SYSTEM'
    PERFORMING return_flight_list ON END OF TASK
    EXPORTING
      destination_from = fli_dest_from
      destination_to   = fli_dest_to.
WRITE:/ 'RFC call started asynchrously'.

AT USER-COMMAND.
  IF sy-ucomm = 'RFC COMPLETED'.
    WRITE:/ 'Remote data received'.
    LOOP AT it_flights INTO ls_flights.
      ...
    ENDLOOP.
  ENDIF.

FORM return_flight_list USING taskname.
  IF taskname = 'DEMO TASK'.
    RECEIVE RESULTS FROM FUNCTION
'BAPI_FLIGHT_GETLIST'
      TABLES
        flight_list = it_flights.
    SET USER-COMMAND 'RFC_COMPLETED'.
  ENDIF.
ENDFORM.                    "RETURN_FLIGHT
```

The program calls the function module, but gives over only the exporting parameters, because asynchronously we cannot give the actual parameters to the remote function. These parameters are given in the FORM where we receive the values and the exceptions if any occurred. This little code shows that the RFC is started asynchronously and the dialog step completes. During that time the remote system finishes the function module and because of the `PERFORMING` closure of the `CALL FUNCTION` statement the local 'sleeping' (caller) program wakes up and executes the `FORM` displaying the received data.

The aRFC solution works not properly if we want to execute asynchronously without knowing whether the remote system is running, but we want to rely on the execution. SAP enhanced the aRFC to transactional RFC (tRFC), which makes it possible to execute the remote function once and only once in the partner system. To define a function module starting as tRFC we should add the `IN BACKGROUND TASK` closure to the `CALL FUNCTION`. (If the `DESTINATION` is not added the function will be scheduled for background execution in the local system.)

The tRFC is also improved to manage queues for guarantee an LUW sequence directly by the application, like QAPI. When working with RFCs SAP manages separately the inbound and outbound queues to serialize the tRFC-s, and the technique for this is called queued remote function calls (qRFC). According to the requirements three data transfer scenarios are used:

- tRFC: It is suitable only for independent function module calls as shown in the example above; and we should keep in mind, that the sequence of the calls is not preserved
- qRFC with outbound queue: function modules stored in a queue are guaranteed to be processed only once and in the given sequence. It is suitable for external (non_SAP) communication.
- qRFC with inbound queue: as the first step the function modules stored in the outbound queue are transferred from it to the inbound queue of the target system. This guarantees that the content of the two queues can be controlled and the sequence of the function modules is preserved. In the target system an inbound scheduler processes the inbound queues in accordance with the specified resources. A restriction is that both the client and the server system must be SAP systems.

For handling qRFC SAP delivers administration transaction, and for programming provides the qRFC-API as well. The real process flow is the following. Let us suppose that in our example a qRFC is given with the third data-transferring scenario (an inbound queue and outbound queue exist). qRFC with inbound and outbound queues involves a 3-phase processing and transfer model (see below). All three phases are completely independent of each other. Separation of these phases ensures that asynchronous processing is as secure as possible.

1. The application data is written to the outbound queue. When is is completed, the data is saved in the database.
2. The QOUT Scheduler transfers this data from the database of the sending system, to the inbound queue of the target system.
3. The target system QIN Scheduler activates processing of the queue in the target system.

V. RFC USAGE AROUND SAP

We tried to show a wide range of available communication type available and possible in the SAP environment. Depending on the requirements and capabilities we can decide, which technology is better and useful.

SAP uses inside the solution the RFCs in many places. First of all as we discuss in another article the Business Framework Architecture is based on remote function. The public methods of the Business Objects are the BAPIs (Business API), which are implemented in the SAP systems as RFC-enabled function modules.

SAP communicates with the front-end software: SAPGUI (SAP Graphical User Interface) on the DIAG (Dynamic Information and Action Gateway) protocol, but in some cases the RFC is used as well. An example is the SAP BW Explorer, which is an MS Excel application, or the SAP Graphical Screen Painter. The SAP could be a client and a server in the RFC communications. If we define a Visual Basic macro within an Excel sheet and we use the RFC communication provided the SAP works as a server. We have to log in and execute the function module with parameters.

```
sub logon_to_sap
   Set ObjR3 = CreateObject("SAP.Functions")
```

```
'--get this info from your logon pad..
   ObjR3.connection.System            = "EC6"
   ObjR3.connection.Client            = "001"
   ObjR3.connection.User              = "SELMACI"
   ObjR3.connection.Password          = "MyPass"
   ObjR3.connection.Language          = "EN"
   ObjR3.connection.ApplicationServer = "saphost"
   ObjR3.connection.SystemNumber      = "00"
end sub
```

The execution snippet is the following (without data declaration:

```
sub read_data
'-Define abap function
   Set ObjR3Func = ObjR3.Add("BAPI_COSTCENTER_GETLIST")
'-Define  paramters and values we will send
   Set ObjPrm1 = ObjR3Func.exports("CONTROLLINGAREA")
   Set ObjPrm2 = ObjR3Func.exports("CONTROLLINGAREA_TO")
   Set ObjPrm3 = ObjR3Func.exports("COMPANYCODE")
   Set ObjPrm4 = ObjR3Func.exports("COMPANYCODE_TO")
   Set ObjPrm5 = ObjR3Func.exports("COSTCENTER")
   Set ObjPrm6 = ObjR3Func.exports("COSTCENTER_TO")
   Set ObjPrm7 = ObjR3Func.exports("PERSON_IN_CHARGE")
   Set ObjPrm8 = ObjR3Func.exports("PERSON_IN_CHARGE_TO")
   Set ObjPrm9 = ObjR3Func.exports("DATE")
   Set ObjPrm10 = ObjR3Func.exports("DATE_TO")
   Set ObjPrm11 = ObjR3Func.exports("COSTCENTERGROUP")

   ObjPrm1.Value = "GB01"
'--do it
   result = ObjR3Func.call

   If result = False Then
      Wscript.Echo  "Error reading sap - " &
ObjR3.connection.User & " - " & ObjR3.connection.System
      ObjR3.connection.LOGOFF
      WScript.Quit
   else
      Set ObjUdextab = ObjR3Func.Tables("COSTCENTER_LIST")
   End If
end sub
```

SAP system can be a client if a server program should be called through RFC communication. This could be an external program or even an EAI (Enterprise Application Integration) middleware. SAP delivers for external RFC programming different libraries: SAP Business Connector, SAP Java Connector (JCo), SAP Connector for Microsoft .NET, SAP NetWeaver RFC library (for C/C++ applications).

SAP itself is using the SAP Java Connector in the NetWeaver communication when the SAP ABAP Web Application Server should communication with the SAP J2EE Web Application Server. (This could be a case by using SAP Mobile Infrastructure and SAP ERP systems.)

As we mentioned above we show some example for the special RFC-destination types. We described trough some tested example the task and usage of the NONE and BACK internal destinations. There is another type of destination not declared yet, the logical ones. A logical destination is referring always to a physical one. We can use this type of destinations when our application refers to a wired (but logical type) RFC destination and during the customizing the system should be set to refer under the predefined (wired) name to a physical destination. With this method the application is not modified, but the end-user has the ability to configure the system according to his needs. SAP uses this technique as well for SAP Business Workflow.

VI. Conclusion

This article drive around the SAP delivered communication features to show and describe the possibility, ability, usability and functionality of the techniques. We realized that the CPI-C had some very good capability like opening and managing parallel communication lines simultaneously. This is not so easy with synchronous RFC, but it could be solved with asynchronous ones, which have other capabilities. The transactional programming of remote communications is important in our age, because of the wide usage of distributed systems and solution. This article does not intend to discover the WebService capabilities and describe the service oriented architecture possibilities, weaknesses and power of the SAP provided solution. Transactional and queued RFCs (tRFC and qRFC) are used not only in remote communication, but in parallel program execution for huge data amounts, like in IS-Utilities environments. Therefore, besides the common usage of remote communication technologies, such enhanced applications of RFCs are available and could be optimal.

Acknowledgement

The author(s) gratefully acknowledge the grant provided by the project TÁMOP-4.2.2/B-10/1-2010-0020, Support of the scientific training, workshops, and establish talent management system at the Óbuda University.

References

[1] Igor Barbaric, “Design Patterns in Object-Oriented ABAP” *Galileo Press Bonn-Boston, 2010*

[2] Rich Heilman, Thomas Jung, “Next Generation ABAP Development” *Galileo Press Bonn-Boston, 2007*

[3] Rita Ősz: Competencies in the Process of Computer Based Learning, 4th International Conference of PHD Students, University of Miskolc, 2003.

[4] Ősz Rita: Interakciók az e-learningben, Számalk konferencia, 2003, www.szamalk.hu/okk/e-learning

[5] Dr. Pletl Szilveszter- Ősz Rita: Problems of teaching intelligent systems, YUINFO Conference, Kopoanik, 2005 (www.yuinfo.org,yu)

[6] Ősz Rita-Pálmai Orsolya: A képernyő az interkulturális folyamatokban, 2009, XI. Dunaújvárosi Nemzetközi Alkalmazott Nyelvészeti és Kommunikációs Konferencia

[7] Tolga Pusatli, Sanjay Misra: Software Measurement Activities in Small and Medium Enterprises: an Empirical Assessment, Acta Polytechnica Hungarica Vol 8. No. 5. 2011

[8] Ősz Rita-Pálmai Orsolya-Váraljai Mariann: A tanulói környezet interkulturális kihívásai 2010, XII. Dunaújvárosi Nemzetközi Alkalmazott Nyelvészeti és Kommunikációs Konferencia

[9] Alok Mishra, Deepti Mishra: ERP Project Implementation: Evidence from the Oil and Gas Sector, Acta Polytechnica Hungarica Vol 8. No. 4. 2011

[10] Seebauer, M., and R. Nagy, "Párhuzamos rendszerek oktatásának bevezetése a Számítógéptechnikai Intézetben", Informatika a regionális felsőoktatásban. Konferencia, 2001. nov. 19., 2001.

[11] Selmeci, A., Orosz, I., Orosz, T., “SAP BAPI as a break-through and future communication enabler”, 2011. Dec. AIS2011