Group	: Computational Linguistics
Topic	: Rosetta3.software

Title	: **Process Communication in Rosetta3**
Author	: Joep Rous

Doc.Nr.	: 283
Date	: July 28, 1989
Status	: concept
Supersedes	: 12 October, 1988
Distribution	: Project
Clearance	: Project
Keywords	:

Institute for Perception Research

Contents

1 Introduction

The Rosetta3 system in execution consists of three processes: the Control process, the Analysis process and the Generation process. The choice for this structure is motivated in document 73; "Global design decisions". For the communication between the processes the mailbox concept is used. There is, however, one exception. The result of applying Analysis to an input sentence is a set of IL-trees. These IL-trees have to be passed to the Generation process. Since for the implementation of such a set of IL-trees a large amount of data might be needed, mailboxes are not the best choice for efficiency reasons. For this kind of data transfer we make use of a global buffer which is shared between the Analysis and Generation process.

2 Mailbox Communication

2.1 Mailbox functions

In this section we will give an overview of the primitive operations on mailboxes that have been implemented in Rosetta.

MB_Open(mbxname, access, create, var mbxid) Opens a mailbox for use by the current process.
- The name of the mailbox that is to be opened should be specified by means of the parameter **mbxname**.
- The **access** parameter can be used to specify whether the mailbox is to be used for *taking* or *putting*.
- The **create** parameter indicates whether a new mailbox must be created or an existing one must be used. If a new mailbox must be created, the value of the parameter mbxname must be unique with respect to the current process tree. If an existing one must be used that mailbox should already be created by another process with the name that is specified in **mbxname**.
- If the mailbox can be opened a unique identification of the mailbox is returned by means of the parameter **mbxid**. This identification should be used in subsequent mailbox calls.

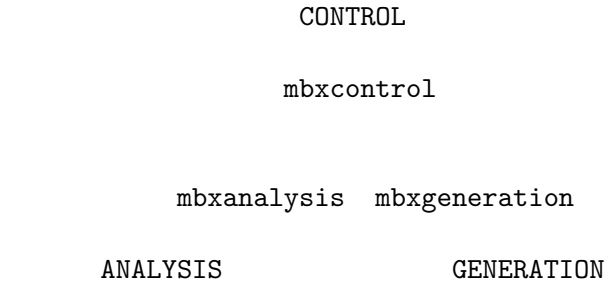
MB_Close(mbxid, delete) Closes a mailbox. After a call to this function the mailbox cannot be used for communication anymore.
- The unique identification of the mailbox that is to be closed should be specified by means of parameter **mbxid**.
- The **delete** parameter specifies whether the mailbox should be deleted after it has been closed. This is only possible if the current process has also created the mailbox. A mailbox can only be deleted if there is no connection to any other process.

MB_Take(mbxid, var msg) Takes a message from the specified mailbox. If there is no message in the mailbox, the current process will be suspended until a message arrives.
- The unique identification of the mailbox that is to be used should be specified by means of parameter **mbxid**.
- The message that is to be read is assigned to parameter **msg**.

MB_Put(mbxid, msg) Puts a message into the specified mailbox. The current process will be suspended until the message is read by some other process.
- The unique identification of the mailbox that is to be used should be specified by means of parameter **mbxid**.
- The message that is to be put into the mailbox is specified by means of parameter **msg**.

2.2 Infra Structure

In Rosetta each process will have its own mailbox. Furthermore, there will be only direct communication between the Analysis and the Control process and between the Generation and the Control process. Therefore, there will be no connection between the Generation mailbox and the Analysis process and between the Analysis mailbox and the Generation process.



The direction of the arrows in the above figure indicates whether the mailbox is used by a process for *putting* or for *taking*.

2.3 Process Synchronization

Except for information exchange, mailboxes are used to synchronize the Analysis, Generation and Control processes. This is possible because MB_Put and MB_Take are synchronous functions. The Rosetta system is activated by starting the Control process. After an initialization phase in which Control creates its own mailbox, the Analysis and Generation processes are created. The Control process waits after each process creation until it receives a message from the created process. This message indicates that the process has finished its initialization phase in which it has opened

the Control mailbox and in which it has created its own mailbox. Next, the Control process can open the mailbox of the created process:

```
program Control;
begin
    MB_Open('control', takeaccess, yes, mbxcontrol);
    ...create analysis process and wait for ready message....
    MB_Take(mbxcontrol, analysis_ready_msg_1)
    MB_Open('analysis', putaccess, no, mbxanalysis);
    ...create generation process and wait for ready message....
    MB_Take(mbxcontrol, generation_ready_msg_1)
    MB_Open('generation', putaccess, no, mbxgeneration);
    .....
end.

program Analysis;
begin
    MB_Open('analysis', takeaccess, yes, mbxanalysis);
    MB_Open('control', putaccess, no, mbxcontrol);
    ...initialize process and inform Control when ready....
    MB_Put(mbxcontrol, analysis_ready_msg_1)
    .....
end.

program Generation;
begin
    MB_Open('generation', takeaccess, yes, mbxgeneration);
    MB_Open('control', putaccess, no, mbxcontrol);
    ...initialize process and inform Control when ready....
    MB_Put(mbxcontrol, generation_ready_msg_1)
    .....
end.
```

Now Analysis and Generation are ready to execute the program body. First, the Analysis process is started. Together with the start message Control sends information about debugging, printing etc.. The Analysis process executes its body and informs the Control process when it is ready. This ready message also contains information about the result of Analysis. If the result is correct then the Control process activates the Generation process. The result of the application of Generation may be ambiguous. Each time the Generation process finds a result it sends a synchronization message to Control. In Control it is decided (by the user) whether the Generation process should be stopped, reset or continued.

If the Generation process is ready, that is, if it has presented all its results or if the user has decided to continue with (the analysis) of a new sentence, both Analysis

and Generation have to clear their windows on the terminal screen. First Generation clears its screen, sends a message to Control and waits for the next sentence. Next, Analysis is triggered by a message from Control to clear its screen and to wait for the next sentence (if any).

```
program Control;
begin
    .....
    ...ask user about debugging, printing, etc....
    repeat
        MB_Put(mbxanalysis, analysis_start_msg)
        ...wait until analysis is ready....
        MB_Take(mbxcontrol, analysis_ready_msg_2)
        if analysis result is correct then
            repeat
                MB_Put(mbxgeneration, generation_start_msg)
                ...wait until generation is ready....
                MB_Take(mbxcontrol, generation_ready_msg_2)
                ask user whether generation should be stopped, reset, continued..
            until stop or reset
            MB_Put(mbxgeneration, generation_stop-reset_msg)
            ...wait until generation ready....
            MB_Take(mbxcontrol, generation_ready_msg_3)
        else
            ask user whether system should be stopped or reset..
        fi
        MB_Put(mbxanalysis, analysis_clear_window_msg)
        ...ask user about debugging, printing, etc....
    until  stop
    ...stop the analysis and generation process ...
    MB_Put(mbxanalysis, analysis_stop_msg)
    MB_Put(mbxgeneration, generation_stop_msg)
    .....
end.


program Analysis;
begin
    .....
    ...wait for permission to continue with process body...
    MB_Take(mbxanalysis, analysis_start_msg);
    while message contains no stop instruction do
        ..execute analysis body and evaluate result..
        MB_Put(mbxcontrol, analysis_ready_msg_2)
        MB_Take(mbxanalysis, analysis_clear_window_msg);
```

```
        ...clear the analysis window...
        MB_Take(mbxanalysis, analysis_start/stop_msg);
    od
    .....
end.


program Generation;
begin
    .....
    ...wait for permission to continue with process body...
    MB_Take(mbxgeneration, generation_start_msg);
    while message contains no stop instruction do
        ..execute generation body (backtracking) and evaluate result..
        while still results and not (stop or reset) do
            MB_Put(mbxcontrol, generation_ready_msg_2)
            MB_Take(mbxgeneration, generation_start/stop-reset_msg);
        od
        if no results then
            MB_Put(mbxcontrol, generation_ready_msg_2)
            MB_Take(mbxgeneration, generation_stop-reset_msg);
        fi
        ...clear the generation window...
        MB_Put(mbxcontrol, generation_ready_msg_3)
        MB_Take(mbxgeneration, generation_start/stop_msg);
    od
    .....
end.
```

2.4 Message Structure

All messages that are passed between the Analysis, Generation and Control process have the same structure. The message is implemented as a record in which different fields are defined for storing different kinds of information. If a process receives a message, it reads only the fields which are relevant at that specific point in the program. If a process receives a wrong message the behaviour of the system is undefined. The system will probably crash because of a run-time error or it will be put into a wait state from which it cannot be activated.

The structure of the message record is :

```
communicationblock = record
                         msg                : msgtype;
```

```
        nextsyn,
        nextlexsense,
        nextstructsense    : boolean;
        clearwindow        : boolean;
        stoplevel          : leveltype;
        intmode,
        batchmode          : boolean;
        printerf,
        debug              : array[leveltype] of boolean;
        ifdescr            : ifdescrtype
    end;
```

The meaning of the fields in this recordtype is as follows:

msg Will be explained below.

nextsyn Used by the generation process to inform control that there are syntactic ambiguous translations (used in: generation_ready_msg_2).

nextlexsense Used by the generation process to inform control that there are lexical ambiguous translations (used in: generation_ready_msg_2)

nextstructsense Used by generation to inform control that there are structural ambiguous translations (used in: generation_ready_msg_2).

clearwindow If TRUE then the screen window of the current process should be cleared. (used in : generation_stop-reset_msg, analysis_clear_window_msg)

stoplevel Specifies the last component of the system which is to be applied. (used in : analysis_start_message, generation_start_message)

intmode Specifies whether the system runs in interactive mode. (used in : analysis_start_message, generation_start_message)

batchmode Specifies whether the system runs in batchmode. (used in : analysis_start_message, generation_start_message)

debug Specifies which component should be executed in debug mode. (used in : analysis_start_message, generation_start_message)

printerf Specifies which component interfaces should be presented on the terminal screen. (used in : analysis_start_message, generation_start_message)

ifdescr Is used to pass the final result of the Analysis process to Generation. (used in: analysis_ready_msg_2, generation_start_msg)

The **msg** field can have the following values:

startmessage, stopmessage Indicates that a process should continue or stop. (used in : analysis_start_msg, generation_start_msg, generation_stop-reset_msg, analysis_stop_msg, generation_stop_msg)

EmptyAnResult The Analysis process has not produced a result. (used in: analysis_ready_msg_2)

CorrectAnResult The Analysis process has produced a result. (used in: analysis_ready_msg_2)

ILNextSyn The Generation process has to present the next syntactic ambiguity. (used in: generation_start_msg)

ILNextLexSense The Generation process has to present the first result of the evaluation of G-Transfer of the current IL-tree. (used in: generation_start_msg)

ILNextStructSense The Generation process has to present the first result of the evaluation of the next IL-tree. (used in: generation_start_msg)

ILFinished There are no ambiguities belonging to the current IL-tree and the current IL-tree is the last one. (used in: generation_ready_msg_2)

AmbigInfo There are still structural, lexical or syntactic ambiguities. (used in: generation_ready_msg_2)

3 Global Buffer Communication

Communication via global buffers is especially useful for the transfer of bulk data. A global buffer is shared between the communicating processes. Therefore no physical transport of the data is needed, process A can write the data into the buffer while process B reads them from the same buffer. Of course these actions must be synchronized in some way or the other. In the Rosetta system reading and writing is synchronized on a high level. The Control process guarantees that Analysis and Generation are mutual exclusive, that is, either Analysis is active or Generation is active.

There will be one single function for global buffer communication:

GlobBuf_CreateBuffer(name, bytesize, var startaddress) Creates a global buffer of name **name** and of size **bytesize**. The startaddress of the buffer is returned in parameter **startaddress**. If a global buffer with name **name** does not exist it is created. If the buffer has already been created by another process, only the address of the existing buffer is returned. A process is not allowed to create two buffers with the same name.

From this interface description it is clear that the function cannot be used using ISO-Pascal. The usage in Rosetta is as follows. First a buffer type is defined :

```
TYPE buffertype  = array[1..MaxElts] of bufferelt;
     pbuffertype = ^buffertype;
```

Next, a buffer is created using the function described above:

```
VAR  buffer     : pbuffertype;
     bufferaddr : INTEGER;
BEGIN
   GlobBuf_CreateBuffer('DataBuffer1', Size(buffertype), bufferaddr);
```

Finally, we use a *typecast* operation to give variable **buffer** the correct value.

```
buffer := bufferaddr::pbuffertype;
```

If these actions have been performed the rest of the software acts as if the variable **buffer** got its value by means of an ordinary bf NEW(buffer) statement. However, there is a snake in the grass. It is not useful to pass pointer values from one process to another by means of a global buffer since they are only relevant within the process space of the current process. For example, if the bufferelements in the buffer would be connected by means of a linked list by process A, then the connections would have no meaning for process B. If process B would access a bufferelement via a linked list pointer this access would probably result in a run-time error.

4 Module Overview

GENERAL:MB Contains the implementation of the abstract datatype *mailbox*.

GENERAL:ANALYSIS Body of the Analysis process, including process communication and synchronization.

GENERAL:GENERATION Body of the Generation process, including process communication and synchronization.

GENERAL:CONTROL Body of the Control process, including process communication and synchronization.

VMS:GLOBBUF Implementation of the abstract datatype *shared global buffer*.

GENERAL:HILTREE Implementation of the Analysis-Generation interface datastructure together with some primitive operations. This module uses the *global buffer* concept.