Universidad
Complutense
Madrid

GRAN
TELESCOPIO
CANARIAS
GTC

MEGARA

Acronyms:

CP	Configuration Plan
CB	Configuration Block
MCS	MEGARA Control System
FMAT	Fiber MOS Assignment Tool
FMOSA	Fiber MOS Assignment file
FMOSAT	FMOSA Table
FMPT	Fiber MOS Positioning Tool
SAA	Stand Alone Application
MP	Motion Program
PP	Positioning Program
DP	Depositioning Program
RP	Robotic Positioner
EA	Exclusion Area
FMM	Fiber MOS Model
MPG	Motion Program Generator
PPA	Pair of Positions Angles
SP	Sky Point

INDEX

FMPT ARCHITECTURE

1. GETTING THE FMPT SOURCE CODE

All source files of the FMPT are included in the file Megara-fmpt-3.3.0.tar.gz (or .xz). This file contains the directory ‘megara-fmpt-3.3.0’ which also can be got from the following URL:

http://guaix.fis.ucm.es/hg/megarafmpt

2. INTERFACES

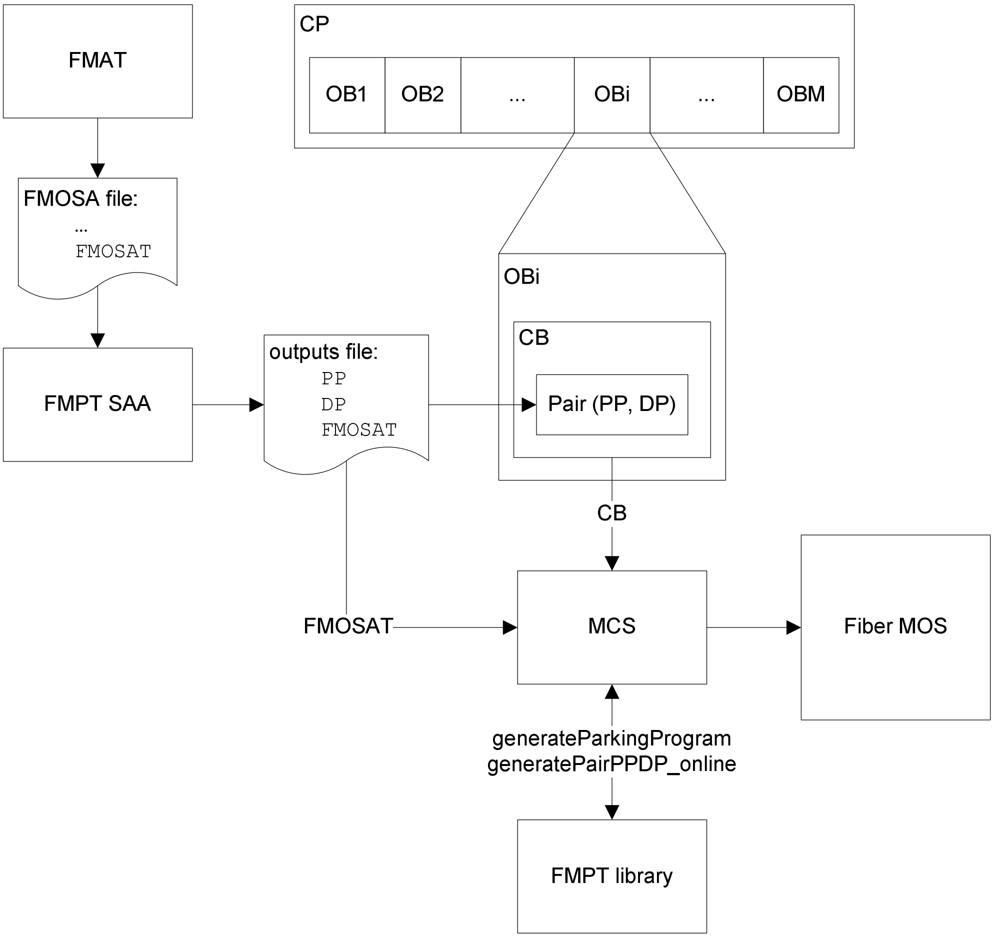


Figure 1: interfaces in the use of the FMPT.

3. STATUS DIAGRAM

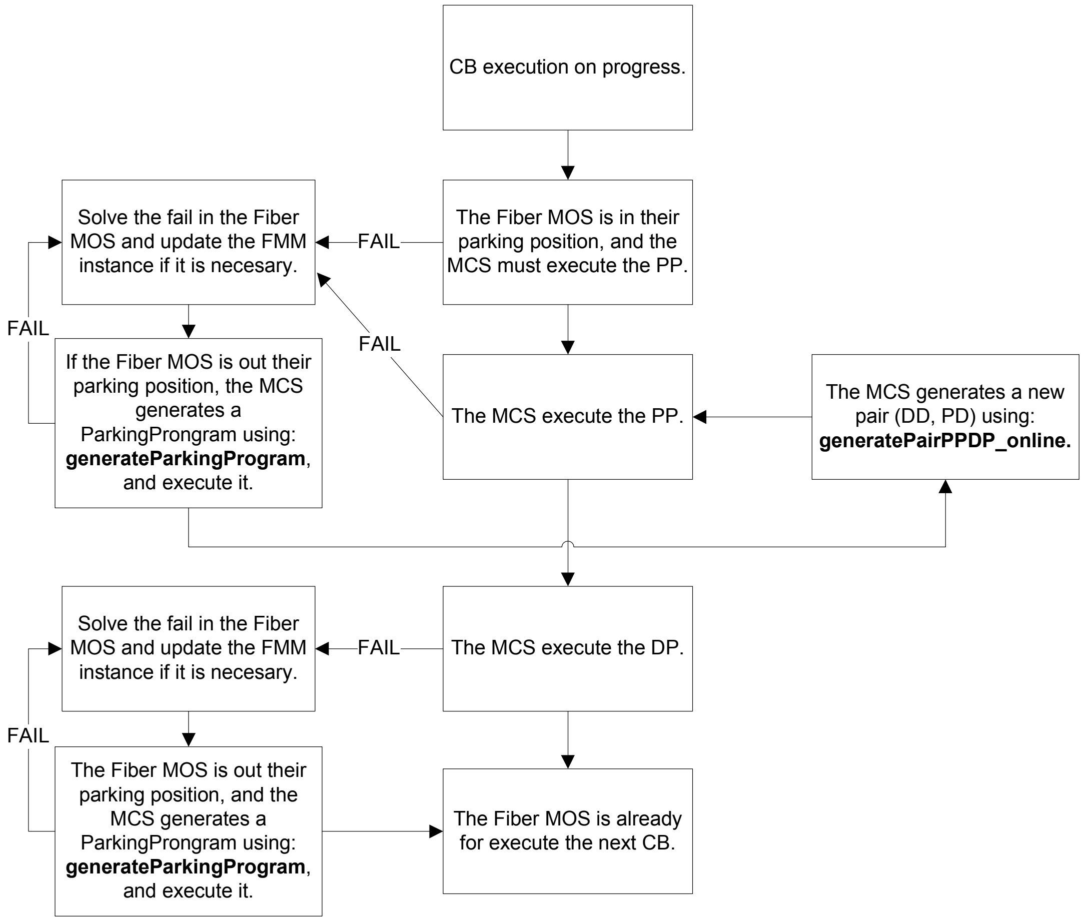

CB execution on progress.
The Fiber MOS is in their parking position, and the MCS must execute the PP.
FAIL
Solve the fail in the Fiber MOS and update the FMM instance if it is necesary.
FAIL
If the Fiber MOS is out their parking position, the MCS generates a ParkingProngram using: **generateParkingProgram**, and execute it.
FAIL
The MCS execute the PP.
The MCS generates a new pair (DD, PD) using: **generatePairPPDP_online.**
The MCS execute the DP.
FAIL
Solve the fail in the Fiber MOS and update the FMM instance if it is necesary.
FAIL
The Fiber MOS is out their parking position, and the MCS generates a ParkingProngram using: **generateParkingProgram**, and execute it.
The Fiber MOS is already for execute the next CB.

Figure 2: status in the use of the FMPT.

4. ESTATUS OF A RP

RP STATE		DESCRIPTION
Parked		(p_1, p___3) = (0, 0)
Security Position		When either p_1>0 or p___3>0, and minimum distance between the RP and any adjacent RP, can’t be less than the sum of the SMPs of both.
Insecurity Position	Not collided and not obstructed	When there isn’t any adjacent RP whose arm is to a distance less than the sum of the SPMs of both RPs, and there aren’t obstacles that prevent the retraction of the arm.
	Not collided and obstructed	When there isn’t any adjacent RP whose arm is to a distance less than the sum of the SPMs of both RPs, and there is some obstacle that prevents retraction of the arm.
	Collided and not obstructed	When there is some adjacent RP whose arm is to a distance less than the sum of the SPMs of both RPs, and there isn’t others obstacles that prevent the retraction of the arm.
	Collided and obstructed	It is considered simply collided.

Figure 3: status of a RP.

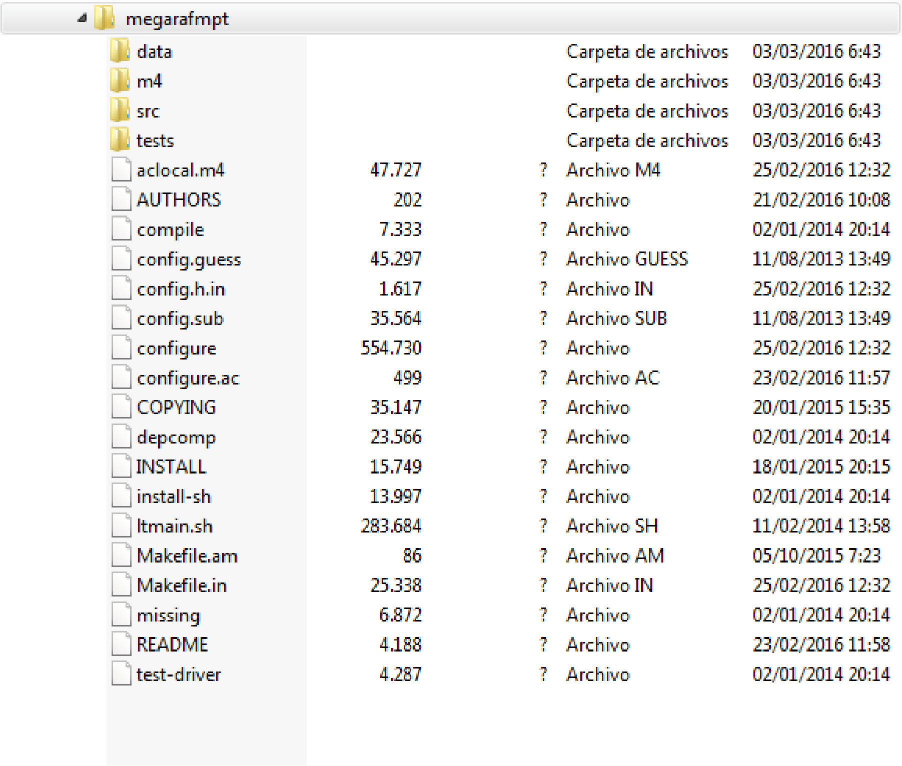
Note that SPM values can be referred to either the RPs are set for generate a pair (PP, DP) or RPs are set for generate a parking program.

5. FILES

The last version FMPT Stand Alone Application is provided in a single file called:

‘megara-fmpt-3.3.0.tar.gz’ (or ‘.xz’)

In any case, it will contain the following files:



megarafmpt

data			Carpeta de archivos	03/03/2016 6:43
m4			Carpeta de archivos	03/03/2016 6:43
src			Carpeta de archivos	03/03/2016 6:43
tests			Carpeta de archivos	03/03/2016 6:43
aclocal.m4	47.727	?	Archivo M4	25/02/2016 12:32
AUTHORS	202	?	Archivo	21/02/2016 10:08
compile	7.333	?	Archivo	02/01/2014 20:14
config.guess	45.297	?	Archivo GUESS	11/08/2013 13:49
config.h.in	1.617	?	Archivo IN	25/02/2016 12:32
config.sub	35.564	?	Archivo SUB	11/08/2013 13:49
configure	554.730	?	Archivo	25/02/2016 12:32
configure.ac	499	?	Archivo AC	23/02/2016 11:57
COPYING	35.147	?	Archivo	20/01/2015 15:35
depcomp	23.566	?	Archivo	02/01/2014 20:14
INSTALL	15.749	?	Archivo	18/01/2015 20:15
install-sh	13.997	?	Archivo	02/01/2014 20:14
ltmain.sh	283.684	?	Archivo SH	11/02/2014 13:58
Makefile.am	86	?	Archivo AM	05/10/2015 7:23
Makefile.in	25.338	?	Archivo IN	25/02/2016 12:32
missing	6.872	?	Archivo	02/01/2014 20:14
README	4.188	?	Archivo	23/02/2016 11:58
test-driver	4.287	?	Archivo	02/01/2014 20:14

Figure 4: files of the FMPT.

The source code is located in the following folders:

- ‘megara-fmpt-3.3.0/data’: containing a subfolder with the Fiber MOS Model instance, and and other folder containing some FMOSA files how examples.
- ‘megara-fmpt-3.3.0/src’: containing the source files ‘.cpp’ and ‘.h’.
- ‘megara-fmpt-3.3.0/test: containing the source test files ‘.cpp’ and ‘.h’

6. SRC MÓDULES

For more legibility, the source files can be grouped into subsets:

General functions			
Math: vclemu Exceptions Vectors Constants Scalars Geometry	Strings: Strings StrPR StrSymbolic TextFile	Lists: SlideArray PointersSlideArray ItemsList PointersList Vector	Operators: Quantificator Function ComposedMotionFunction RampFunction SquareFunction MotionFunction

Figure 5: general functions of the FMPT.

Models		Generator
Fiber MOS Model: Instruction RoboticPositionerList1 RoboticPositionerList2 RoboticPositionerList3 ExclusionAreaList FiberConnectionModel FiberMOSModel FileMethods	Telescope Projection Model: SkyPoint ProjectionPoint FMOSATable PositionerCenter Tile TelescopeProjectionModel	Motion Program Generator: MessageInstruction MotionProgram Allocation AllocationList MotionProgramValidator MotionProgramGenerator Outputs

Figure 6: specific functions of the FMPT.

A FMM is accurate representation of the real Fiber MOS which allow represent the RPs in any possible status, including some setting status of the controller, and possible anomalous status of the RP, for simulate the motion of the RPs and determine the risk of collision between their arms and any possible obstacles. A FMM is used by a MPG for generate MPs which avoid the risk of collision. The files describing the FMM and the MPG has been named with the same name of the main class that each describe, although some files can contains more things or only functions. They are briefly described below:

Instruction: an instruction serve for configure the motion function of a RP of the FMM.

RoboticPositionerList1/2/3: a RP list represent the RPs of the FMM.

ExclusionAreaList: an EA list represents the EAs of the FMM. This is the occupied space by the IFU placed in the center of the Fiber MOS.

FiberConnectionModel: a fiber connection model can be used for determine the Id of each fiber connected to each RP, and their position in the focal plan and the pseudo-slot. This is a characteristic not used in the FMPT, but it was useful during MEGARA project developing.

FiberMOSModel: a FMM is composed by a RP list and an EA list, and it is used by a MPG for generate MPs without risk of collisions.

FileMethods: contains functions for facilitate the loading and saving of the Fiber MOS Model instance.

SkyPoint: a projection point can be calculated from a SP, using the telescope projection model. This is a characteristic not used in the FMPT, but it was useful during MEGARA project developing.

ProjectionPoint: a set of projection points allow represent the points where the fibers must be positioned as close as possible of them.

FMOSATable: a FMOSA table is a structure for facilitate the load and saving of the files type FMOSA. For see a description of file type FMOSA, see either the user manual of the FMPT or the user manual of the FMAT.

PositionerCenter: a positioner center is a structure containing the properties (Id, x0, y0). A positioner center list allows facilitate the loading and saving of tables with the fields (Id, x0, y0). This is useful for process the table provided by the manufacturer of the Fiber MOS.

Tile: a tile is a structure, containing a projection point list whose coordinates are given respect the coordinate system of the focal plane of the telescope. This is a characteristic not used in the FMPT, but it was useful during MEGARA project developing.

TelecopeProjectionModel: a Telescope Projection Model allows get a tile from the whole set of SPs and a given set of Fiber MOS Pointing Parameters. This is a characteristic not used in the FMPT, but it was useful during MEGARA project developing.

MessageInstruction: a message instruction is a structure containing the Id of a RP and an Instruction. A MP is composed by a list of list of message instructions.

MotionProgram: a MP is a structure composed by a list of list of message instructions.

Allocation: an allocation is a structure containing an Id of a RP and a projection point.

AllocationList: an allocation list is part of the MPG, and it is given by the FMOSA file.

MotionProgramValidator: a MPV allows check if a MP produces any collision. For this, the MPG requires a FMM. The FMM shall be configured with the starting positions from with the MP will be executed. Furthermore the FMM shall be configured with the purpose of check the MP, so that the SPM of the RPs is right.

MotionProgramGenerator: a MPG allow generates MPs without risk of collision. The MPG can be used for generates either parking program or a pair (PP, DP).

outputs: an outputs structure is composed by the data {PP, DP, FMOSAT}, and it facilitates the loading and saving output files.

7. FLUX DIAGRAM

The main function is located in the file:

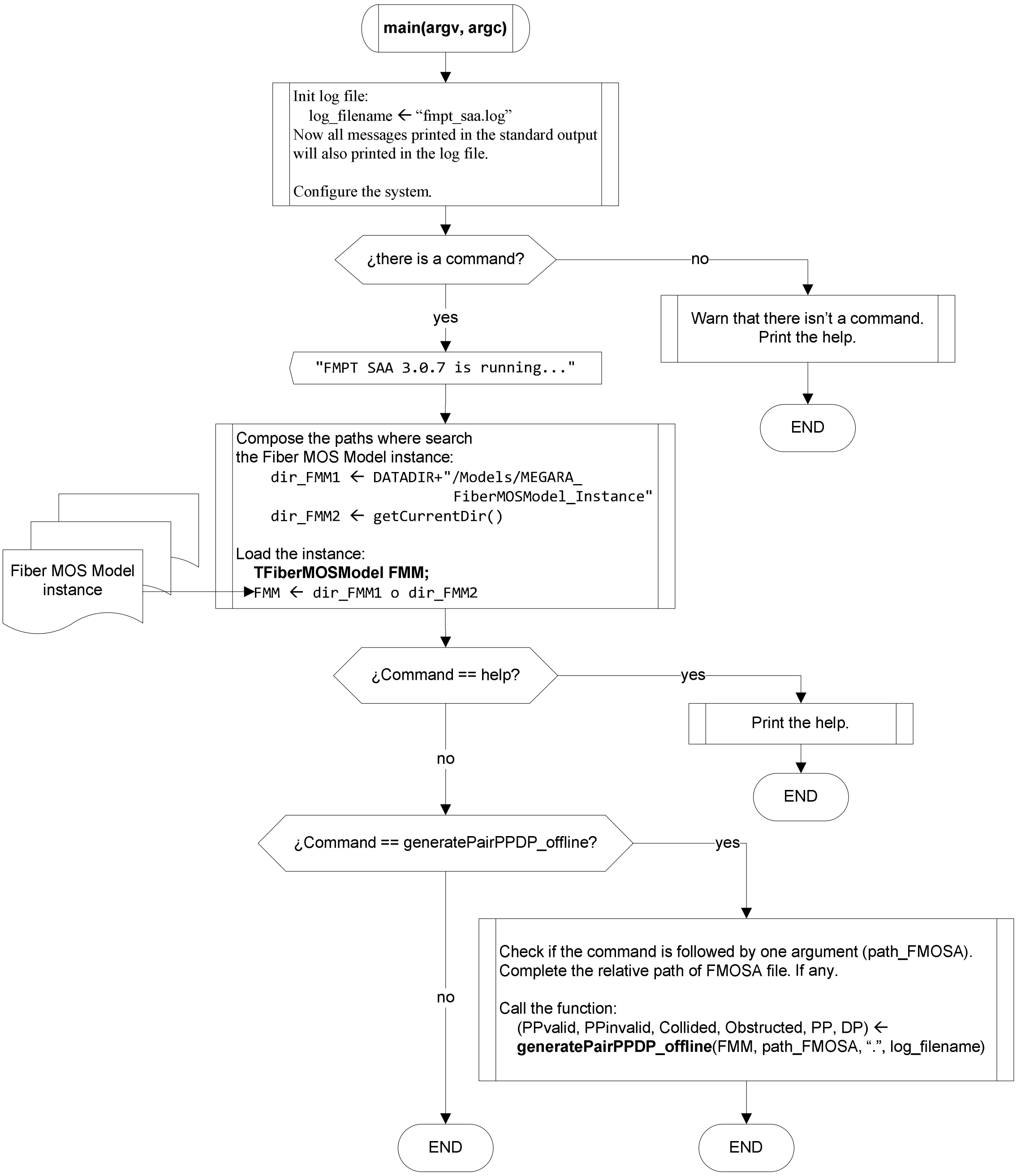
‘megara-fmpt-3.3.0/src/main.cpp’

Let’s see a flux diagram since the main function to the enough degree of depth, for serve as a starting point to understand the rest of code:

7.1 Function int main(argc, argv)

The main function is called when the program is executed in a console of the system. The arguments of the main function are the same arguments that are written after the name of the program. The first of these arguments is always a command followed by their arguments.

For see a detailed description of use of the FMPT SAA, consul the user manual (document TEC_MEG_171), wear can be found a description of the commands, their arguments, and the input and output files for each functionality of the application FMPT SAA.



```
Init log file:
   log_filename ← "fmpt_saa.log"
Now all messages printed in the standard output
will also printed in the log file.

Configure the system.
```

¿there is a command?

Warn that there isn't a command.
Print the help.

```
"FMPT SAA 3.0.7 is running..."
```

```
Compose the paths where search
the Fiber MOS Model instance:
    dir_FMM1 ← DATADIR+"/Models/MEGARA_
                         FiberMOSModel_Instance"
    dir_FMM2 ← getCurrentDir()

Load the instance:
   TFiberMOSModel FMM;
   FMM ← dir_FMM1 o dir_FMM2
```

¿Command == help?

Print the help.

¿Command == generatePairPPDP_offline?

```
Check if the command is followed by one argument (path_FMOSA).
Complete the relative path of FMOSA file. If any.

Call the function:
   (PPvalid, PPinvalid, Collided, Obstructed, PP, DP) ←
   generatePairPPDP_offline(FMM, path_FMOSA, ".", log_filename)
```

Figure 7: the main flux diagram of the FMPT.

7.2 void generatePairPPDP_offline(PPvalid, DPvalid, Collided, Obstructed, PP, DP, FMM, input_path, output_dir, log_filename)

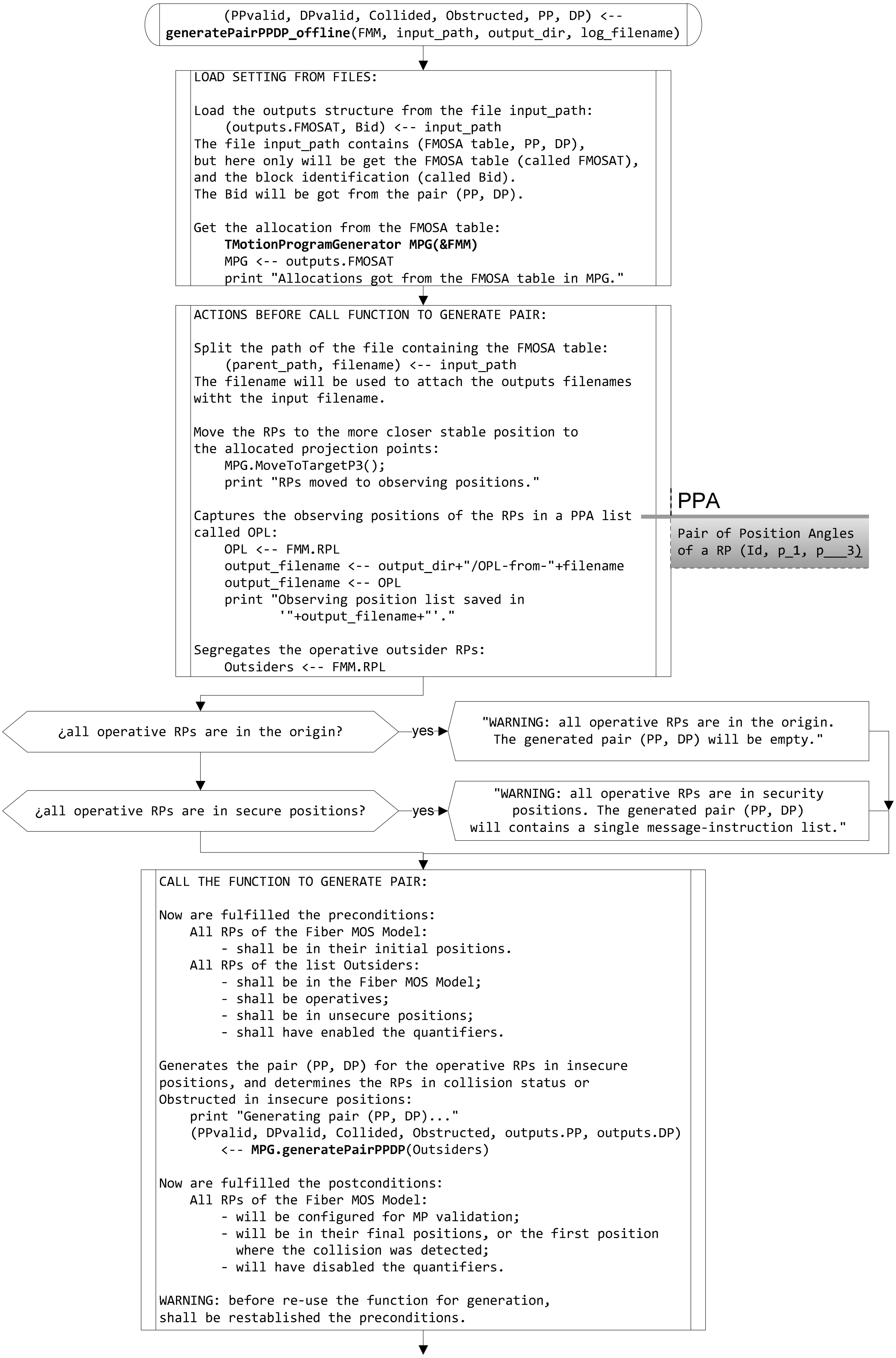
This is a function of the MPG which serves for generate a pair (PP, DP) from a FMOSA file.

Inputs:

- FMM: the Fiber MOS Model configured with the instance loaded from the directory: '/usr/local/share/megara-fmpt/Models/MEGARA_FiberMOSModel_Instance'. The MPG has access to the MPG, since the MPG is built attached to the FMM.
- inputh_path: the path of the FMOSA file, containing the allocations (of RPs to projection points).
- output_dir: the directory where the output files will be saved.
- log_filename: the name of the log file, where will be wrote the same that is wrote in the standard output (it is to say, the console).

Outputs:

- PPvalid: indicates if the generated PP, avoid any collisions.
- DPvalid: indicates if the generated DP, avoid any collisions.
- Collided: list of RPs whose security positions can't be recovered, because are in collision status.
- Obstructed: list of RPs whose security positions can't be recovered, because are obstructed in insecurity positions, locked by adjacent obstacles.
- PP: the positioning program to be generated.
- DP: the depositioning program to be generated.



```
(PPvalid, DPvalid, Collided, Obstructed, PP, DP) <--
generatePairPPDP_offline(FMM, input_path, output_dir, log_filename)
```

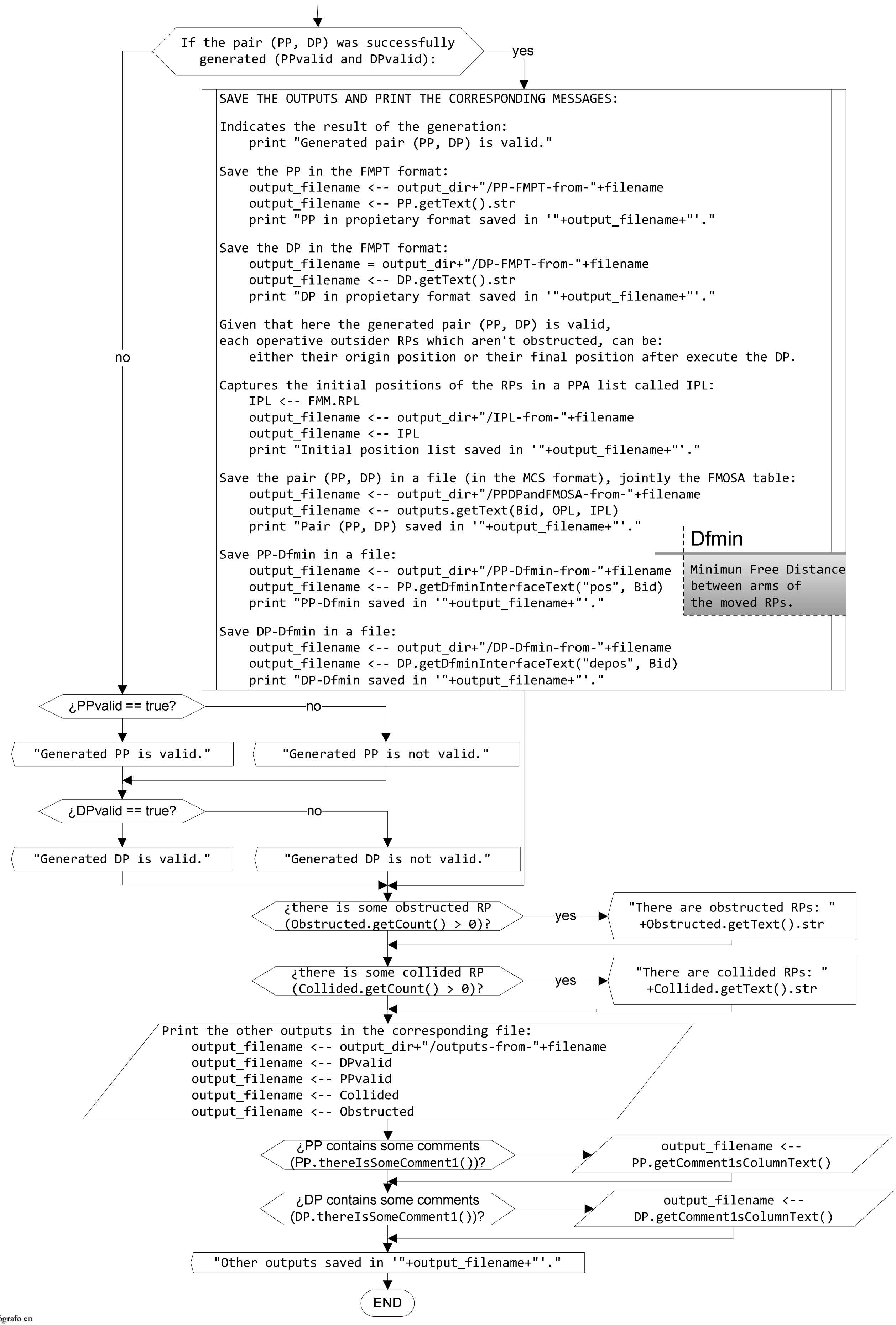
```
LOAD SETTING FROM FILES:

Load the outputs structure from the file input_path:
    (outputs.FMOSAT, Bid) <-- input_path
The file input_path contains (FMOSA table, PP, DP),
but here only will be get the FMOSA table (called FMOSAT),
and the block identification (called Bid).
The Bid will be got from the pair (PP, DP).

Get the allocation from the FMOSA table:
    TMotionProgramGenerator MPG(&FMM)
    MPG <-- outputs.FMOSAT
    print "Allocations got from the FMOSA table in MPG."
```

```
ACTIONS BEFORE CALL FUNCTION TO GENERATE PAIR:

Split the path of the file containing the FMOSA table:
    (parent_path, filename) <-- input_path
The filename will be used to attach the outputs filenames
witht the input filename.

Move the RPs to the more closer stable position to
the allocated projection points:
    MPG.MoveToTargetP3();
    print "RPs moved to observing positions."

Captures the observing positions of the RPs in a PPA list
called OPL:
    OPL <-- FMM.RPL
    output_filename <-- output_dir+"/OPL-from-"+filename
    output_filename <-- OPL
    print "Observing position list saved in
           '"+output_filename+"'."

Segregates the operative outsider RPs:
    Outsiders <-- FMM.RPL
```

PPA
Pair of Position Angles of a RP (Id, p_1, p___3)
¿all operative RPs are in the origin?
yes
"WARNING: all operative RPs are in the origin. The generated pair (PP, DP) will be empty."
¿all operative RPs are in secure positions?
yes
"WARNING: all operative RPs are in security positions. The generated pair (PP, DP) will contains a single message-instruction list."

```
CALL THE FUNCTION TO GENERATE PAIR:

Now are fulfilled the preconditions:
    All RPs of the Fiber MOS Model:
        - shall be in their initial positions.
    All RPs of the list Outsiders:
        - shall be in the Fiber MOS Model;
        - shall be operatives;
        - shall be in unsecure positions;
        - shall have enabled the quantifiers.

Generates the pair (PP, DP) for the operative RPs in insecure
positions, and determines the RPs in collision status or
Obstructed in insecure positions:
    print "Generating pair (PP, DP)..."
    (PPvalid, DPvalid, Collided, Obstructed, outputs.PP, outputs.DP)
        <-- MPG.generatePairPPDP(Outsiders)

Now are fulfilled the postconditions:
    All RPs of the Fiber MOS Model:
        - will be configured for MP validation;
        - will be in their final positions, or the first position
          where the collision was detected;
        - will have disabled the quantifiers.

WARNING: before re-use the function for generation,
shall be restablished the preconditions.
```

Figure 8: flux diagran of the function generatePairPPDP_offline.Part 1.



If the pair (PP, DP) was successfully generated (PPvalid and DPvalid):
yes
no
Dfmin
Minimun Free Distance between arms of the moved RPs.
¿PPvalid == true?
no
¿DPvalid == true?
no
¿there is some obstructed RP (Obstructed.getCount() > 0)?
yes
¿there is some collided RP (Collided.getCount() > 0)?
yes
¿PP contains some comments (PP.thereIsSomeComment1())?
¿DP contains some comments (DP.thereIsSomeComment1())?
END

```
SAVE THE OUTPUTS AND PRINT THE CORRESPONDING MESSAGES:

Indicates the result of the generation:
    print "Generated pair (PP, DP) is valid."

Save the PP in the FMPT format:
    output_filename <-- output_dir+"/PP-FMPT-from-"+filename
    output_filename <-- PP.getText().str
    print "PP in propietary format saved in '"+output_filename+"'."

Save the DP in the FMPT format:
    output_filename = output_dir+"/DP-FMPT-from-"+filename
    output_filename <-- DP.getText().str
    print "DP in propietary format saved in '"+output_filename+"'."

Given that here the generated pair (PP, DP) is valid,
each operative outsider RPs which aren't obstructed, can be:
    either their origin position or their final position after execute the DP.

Captures the initial positions of the RPs in a PPA list called IPL:
    IPL <-- FMM.RPL
    output_filename <-- output_dir+"/IPL-from-"+filename
    output_filename <-- IPL
    print "Initial position list saved in '"+output_filename+"'."

Save the pair (PP, DP) in a file (in the MCS format), jointly the FMOSA table:
    output_filename <-- output_dir+"/PPDPandFMOSA-from-"+filename
    output_filename <-- outputs.getText(Bid, OPL, IPL)
    print "Pair (PP, DP) saved in '"+output_filename+"'."

Save PP-Dfmin in a file:
    output_filename <-- output_dir+"/PP-Dfmin-from-"+filename
    output_filename <-- PP.getDfminInterfaceText("pos", Bid)
    print "PP-Dfmin saved in '"+output_filename+"'."

Save DP-Dfmin in a file:
    output_filename <-- output_dir+"/DP-Dfmin-from-"+filename
    output_filename <-- DP.getDfminInterfaceText("depos", Bid)
    print "DP-Dfmin saved in '"+output_filename+"'."
```

```
"Generated PP is valid."
"Generated PP is not valid."
"Generated DP is valid."
"Generated DP is not valid."
"There are obstructed RPs: "
 +Obstructed.getText().str
"There are collided RPs: "
 +Collided.getText().str
```

```
Print the other outputs in the corresponding file:
    output_filename <-- output_dir+"/outputs-from-"+filename
    output_filename <-- DPvalid
    output_filename <-- PPvalid
    output_filename <-- Collided
    output_filename <-- Obstructed
```

```
output_filename <--
PP.getComment1sColumnText()
output_filename <--
DP.getComment1sColumnText()
"Other outputs saved in '"+output_filename+"'."
```

Figure 9: flux diagran of the function generatePairPPDP_offline.Part 2.

7.3 generatePairPPDP(DPvalid, Collided, Obstructed, DP, Outsiders)

This is a function of the MPG which serves for generate a pair (PP, DP) from a FMM with the RPs in their observing points, and the list Outsiders. This functions is used either online or offline. The use of this function requires a series of previous preparations, which can be summarized how load the FMM instance and load the allocations of RPs to projection points. For the use offline has been implemented the function generatePairPPDP_offline, and the use online has been illustrated in the example provided with the FMPT, in the file:

‘Main_example_generatePairPPDP_online.cpp’

Implicit input parameters:

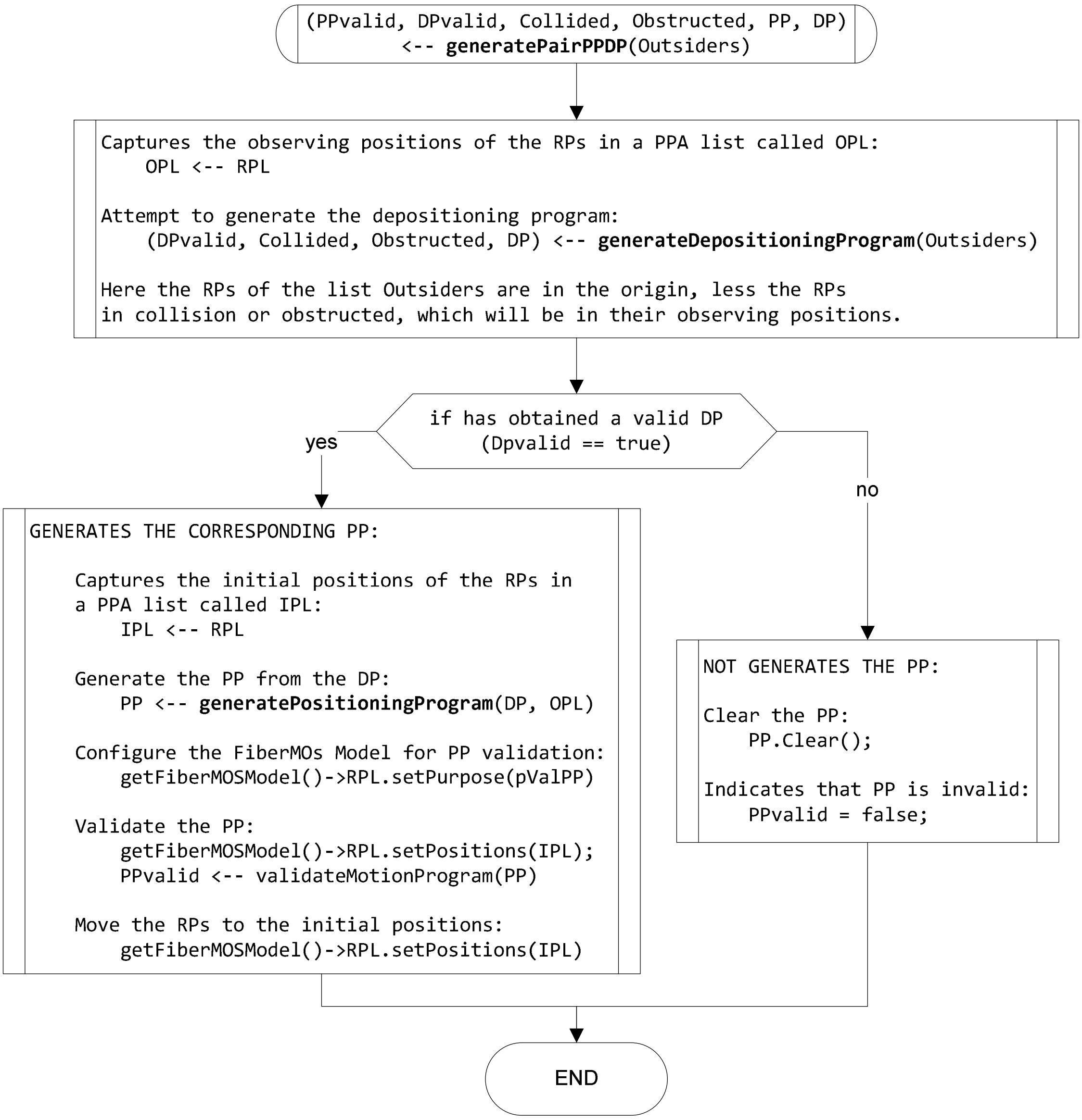
- RP-to-projection-point Allocation list: the MPG must have previously assigned the allocation. These allocations must be got from the FMOSA file passed how argument to the command generatePairPPDP_offline.
- The FMM with the RPs in their observing positions: the MPG has access to the MPG, for that the MPG must be built attached to the FMM.

Explicit input parameters:

- Outsiders: list of RPs whose observing positions are in insecurity positions.

Return parameters:

- PPvalid: indicates if the generated PP, avoid collisions.
- DPvalid: indicates if the generated DP, avoid collisions.
- Collided: list of RPs whose security positions can’t be recovered, because are in collision status.
- Obstructed: list of RPs whose security positions can’t be recovered, because are obstructed in insecurity positions, locked by adjacent obstacles.
- PP: the positioning program to be generated.
- DP: the depositioning program to be generated.



```
(PPvalid, DPvalid, Collided, Obstructed, PP, DP)
        <-- generatePairPPDP(Outsiders)
```

```
Captures the observing positions of the RPs in a PPA list called OPL:
    OPL <-- RPL

Attempt to generate the depositioning program:
    (DPvalid, Collided, Obstructed, DP) <-- generateDepositioningProgram(Outsiders)

Here the RPs of the list Outsiders are in the origin, less the RPs
in collision or obstructed, which will be in their observing positions.
```

```
if has obtained a valid DP
    (Dpvalid == true)
```

```
GENERATES THE CORRESPONDING PP:

    Captures the initial positions of the RPs in
    a PPA list called IPL:
        IPL <-- RPL

    Generate the PP from the DP:
        PP <-- generatePositioningProgram(DP, OPL)

    Configure the FiberMOs Model for PP validation:
        getFiberMOSModel()->RPL.setPurpose(pValPP)

    Validate the PP:
        getFiberMOSModel()->RPL.setPositions(IPL);
        PPvalid <-- validateMotionProgram(PP)

    Move the RPs to the initial positions:
        getFiberMOSModel()->RPL.setPositions(IPL)
```

```
NOT GENERATES THE PP:

Clear the PP:
    PP.Clear();

Indicates that PP is invalid:
    PPvalid = false;
```

Figure 10: flux diagram of the function generatePairPPDP.

7.4 generateDepositioningProgram(DPvalid, Collided, Obstructed, DP, Outsiders)

This is a function of the MPG which serves for generate a DP.

Implicit input parameters:

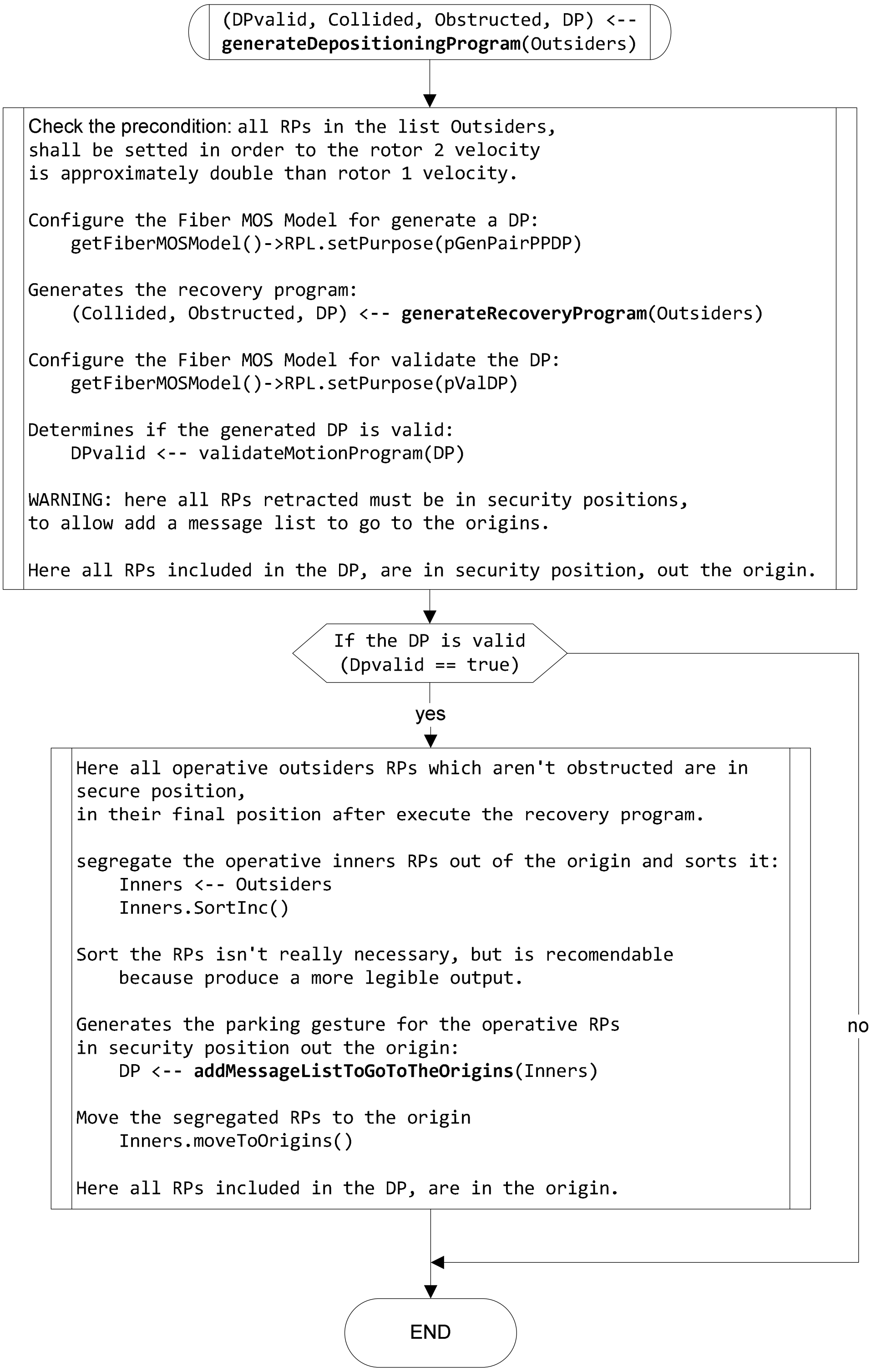
- RP-to-projection-point Allocation list: the MPG must have previously assigned the allocation. These allocations must be got from the FMOSA file passed how argument to the command generatePairPPDP_offline.
- The FMM with the RPs in their observing positions: the MPG has access to the MPG, for that the MPG must be built attached to the FMM.

Explicit input parameters:

- Outsiders: list of RPs whose observing positions are in insecurity positions.

Return parameters:

- DPvalid: indicates if the generated DP, avoid collisions.
- Collided: list of RPs whose security positions can't be recovered, because are in collision status.
- Obstructed: list of RPs whose security positions can't be recovered, because are obstructed in insecurity positions, locked by adjacent obstacles.
- DP: the depositioning program to be generated.



```
(DPvalid, Collided, Obstructed, DP) <--
generateDepositioningProgram(Outsiders)
```

```
Check the precondition: all RPs in the list Outsiders,
shall be setted in order to the rotor 2 velocity
is approximately double than rotor 1 velocity.

Configure the Fiber MOS Model for generate a DP:
    getFiberMOSModel()->RPL.setPurpose(pGenPairPPDP)

Generates the recovery program:
    (Collided, Obstructed, DP) <-- generateRecoveryProgram(Outsiders)

Configure the Fiber MOS Model for validate the DP:
    getFiberMOSModel()->RPL.setPurpose(pValDP)

Determines if the generated DP is valid:
    DPvalid <-- validateMotionProgram(DP)

WARNING: here all RPs retracted must be in security positions,
to allow add a message list to go to the origins.

Here all RPs included in the DP, are in security position, out the origin.
```

```
If the DP is valid
(Dpvalid == true)
```

```
Here all operative outsiders RPs which aren't obstructed are in
secure position,
in their final position after execute the recovery program.

segregate the operative inners RPs out of the origin and sorts it:
    Inners <-- Outsiders
    Inners.SortInc()

Sort the RPs isn't really necessary, but is recomendable
    because produce a more legible output.

Generates the parking gesture for the operative RPs
in security position out the origin:
    DP <-- addMessageListToGoToTheOrigins(Inners)

Move the segregated RPs to the origin
    Inners.moveToOrigins()

Here all RPs included in the DP, are in the origin.
```

Figure 11: flux diagram of the function generateDepositioningProgram.

7.5 generateRecoveryProgram(Collided, Obstructed, RecoveryProgram, Outsiders)

This is a function of the MPG which serves for generate a recovery program. A recovery program is a depositioning program less the last step to go to the initial position. It is to say, a recovery program is a program to go to the security positions avoiding collisions.

Implicit input parameters:

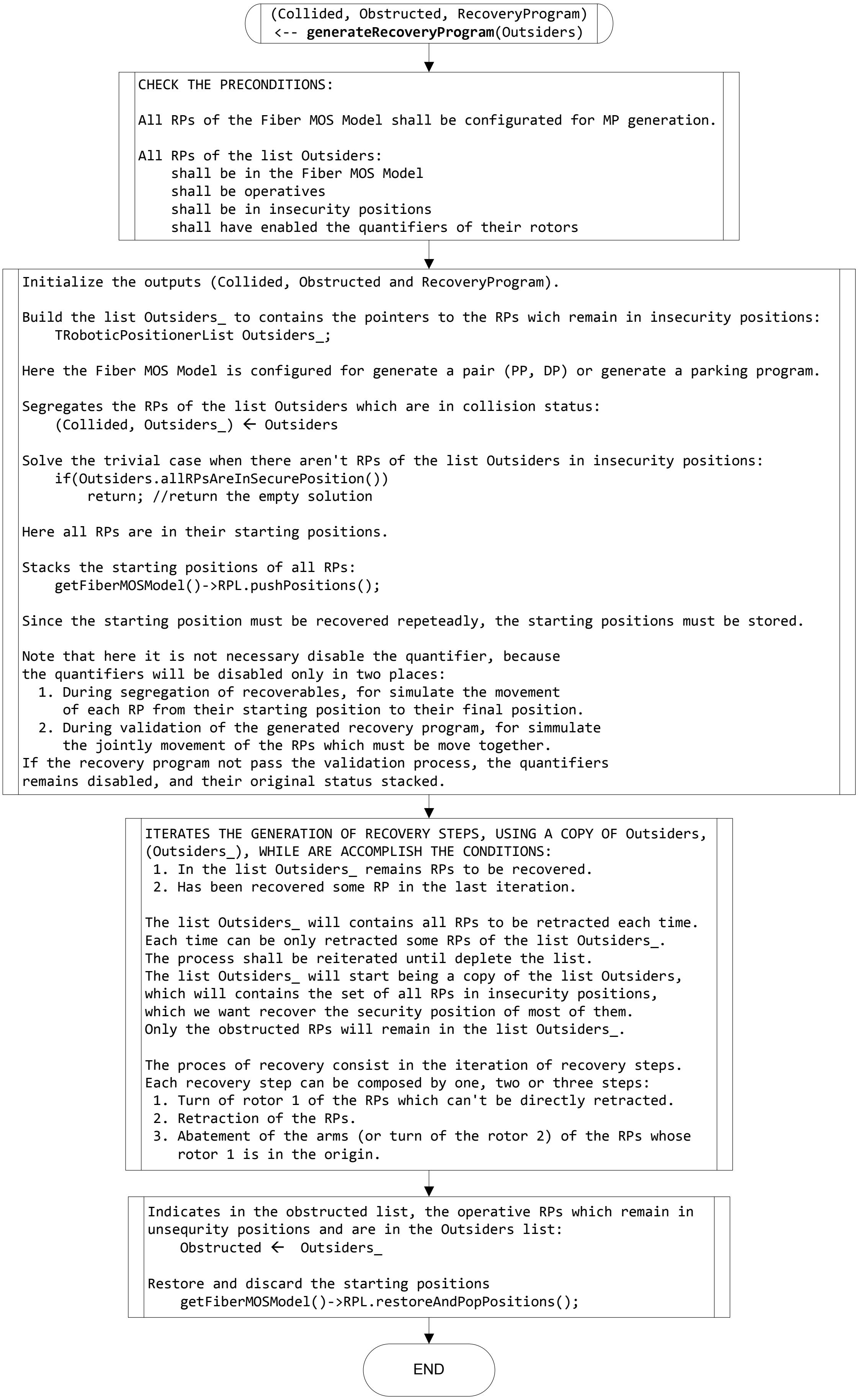
- RP-to-projection-point Allocation list: the MPG must have previously assigned the allocation. These allocations must be got from the FMOSA file passed how argument to the command generatePairPPDP_offline.
- The FMM with the RPs in their observing positions: the MPG has access to the MPG, for that the MPG must be built attached to the FMM.

Explicit input parameters:

- Outsiders: list of RPs whose observing positions are in insecurity positions.

Return parameters:

- Collided: list of RPs whose security positions can't be recovered, because are in collision status.
- Obstructed: list of RPs whose security positions can't be recovered, because are obstructed in insecurity positions, locked by adjacent obstacles.
- RecoveryProgram: the recovery program to be generated.



```
(Collided, Obstructed, RecoveryProgram)
<-- generateRecoveryProgram(Outsiders)
```

```
CHECK THE PRECONDITIONS:

All RPs of the Fiber MOS Model shall be configurated for MP generation.

All RPs of the list Outsiders:
    shall be in the Fiber MOS Model
    shall be operatives
    shall be in insecurity positions
    shall have enabled the quantifiers of their rotors
```

```
Initialize the outputs (Collided, Obstructed and RecoveryProgram).

Build the list Outsiders_ to contains the pointers to the RPs wich remain in insecurity positions:
    TRoboticPositionerList Outsiders_;

Here the Fiber MOS Model is configured for generate a pair (PP, DP) or generate a parking program.

Segregates the RPs of the list Outsiders which are in collision status:
    (Collided, Outsiders_) ← Outsiders

Solve the trivial case when there aren't RPs of the list Outsiders in insecurity positions:
    if(Outsiders.allRPsAreInSecurePosition())
        return; //return the empty solution

Here all RPs are in their starting positions.

Stacks the starting positions of all RPs:
    getFiberMOSModel()->RPL.pushPositions();

Since the starting position must be recovered repeteadly, the starting positions must be stored.

Note that here it is not necessary disable the quantifier, because
the quantifiers will be disabled only in two places:
  1. During segregation of recoverables, for simulate the movement
     of each RP from their starting position to their final position.
  2. During validation of the generated recovery program, for simmulate
     the jointly movement of the RPs which must be move together.
If the recovery program not pass the validation process, the quantifiers
remains disabled, and their original status stacked.
```

```
ITERATES THE GENERATION OF RECOVERY STEPS, USING A COPY OF Outsiders,
(Outsiders_), WHILE ARE ACCOMPLISH THE CONDITIONS:
 1. In the list Outsiders_ remains RPs to be recovered.
 2. Has been recovered some RP in the last iteration.

The list Outsiders_ will contains all RPs to be retracted each time.
Each time can be only retracted some RPs of the list Outsiders_.
The process shall be reiterated until deplete the list.
The list Outsiders_ will start being a copy of the list Outsiders,
which will contains the set of all RPs in insecurity positions,
which we want recover the security position of most of them.
Only the obstructed RPs will remain in the list Outsiders_.

The proces of recovery consist in the iteration of recovery steps.
Each recovery step can be composed by one, two or three steps:
 1. Turn of rotor 1 of the RPs which can't be directly retracted.
 2. Retraction of the RPs.
 3. Abatement of the arms (or turn of the rotor 2) of the RPs whose
    rotor 1 is in the origin.
```

```
Indicates in the obstructed list, the operative RPs which remain in
unsequrity positions and are in the Outsiders list:
    Obstructed ←  Outsiders_

Restore and discard the starting positions
    getFiberMOSModel()->RPL.restoreAndPopPositions();
```

Figure 12: flux diagram of the function generateRecoveryProgram.