The Generator of Modules
$G^{en}{}_{o}M$

Sara Fleury - Matthieu Herrb

November 2010

Agenda

Agenda

G^{en}$_{o}$M: Generator of Modules

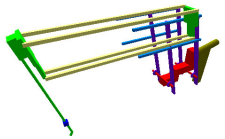
adam

junior

lama

h2

h2bis



lapa

karma

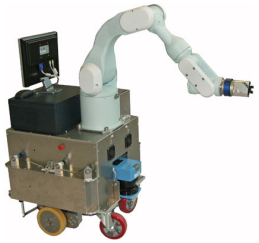
diligent

scout

G^{en}$_{o}$M: Generator of Modules

dala

rackham



jido

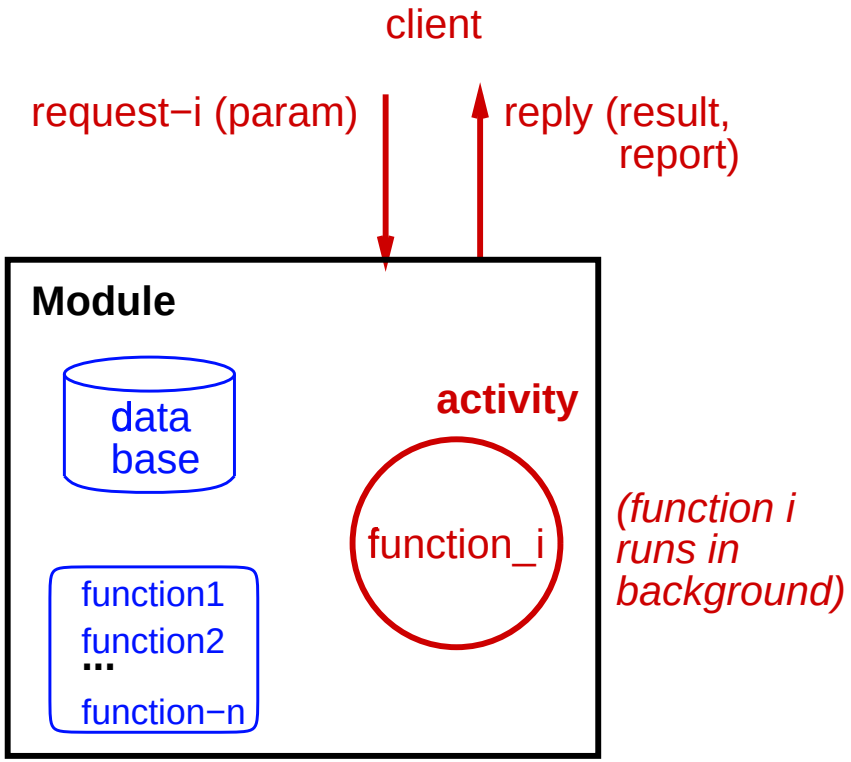
lhassa

hrp2

Pioneer / Player

$G^{en}{}_{o}M$: Generator of Modules

Tool to integrate processing functions into independant servers or **modules**



Activity: function in processing

Reports: “incorrect-parameters”, “solution-not-found”, “not-enough-memory”, “unforeseen-case”

Why encapsulate functions ?

Software integration on embedded systems

Requirements:

- Controllable system
  - processing start/stop
  - dynamic parametrization
  - error recovery
- Communicating system
  - data transfers
  - use of other functions

Why encapsulate functions (cont.)

Requirements (cont.)

- Standard interfaces
  - list of available functions
  - Data structures of input/outputs
  - Validity domains
  - List of failure reports
- Standard organization of files

Why encapsulate functions (cont.)

Pros:

→ makes integration simpler (semi-automatic)

→ makes maintenance simpler (**sustainability** and **perenity**)

Cons:

→ Need to learn Genom !

Module creation principle

4 steps:

1. Describe the module (`.gen` file):
   - name
   - list of services (requests): parameters, functions, . . .
2. Generate the module (`genom`)
3. Incrementally fill-in algorithms
4. Compile everything

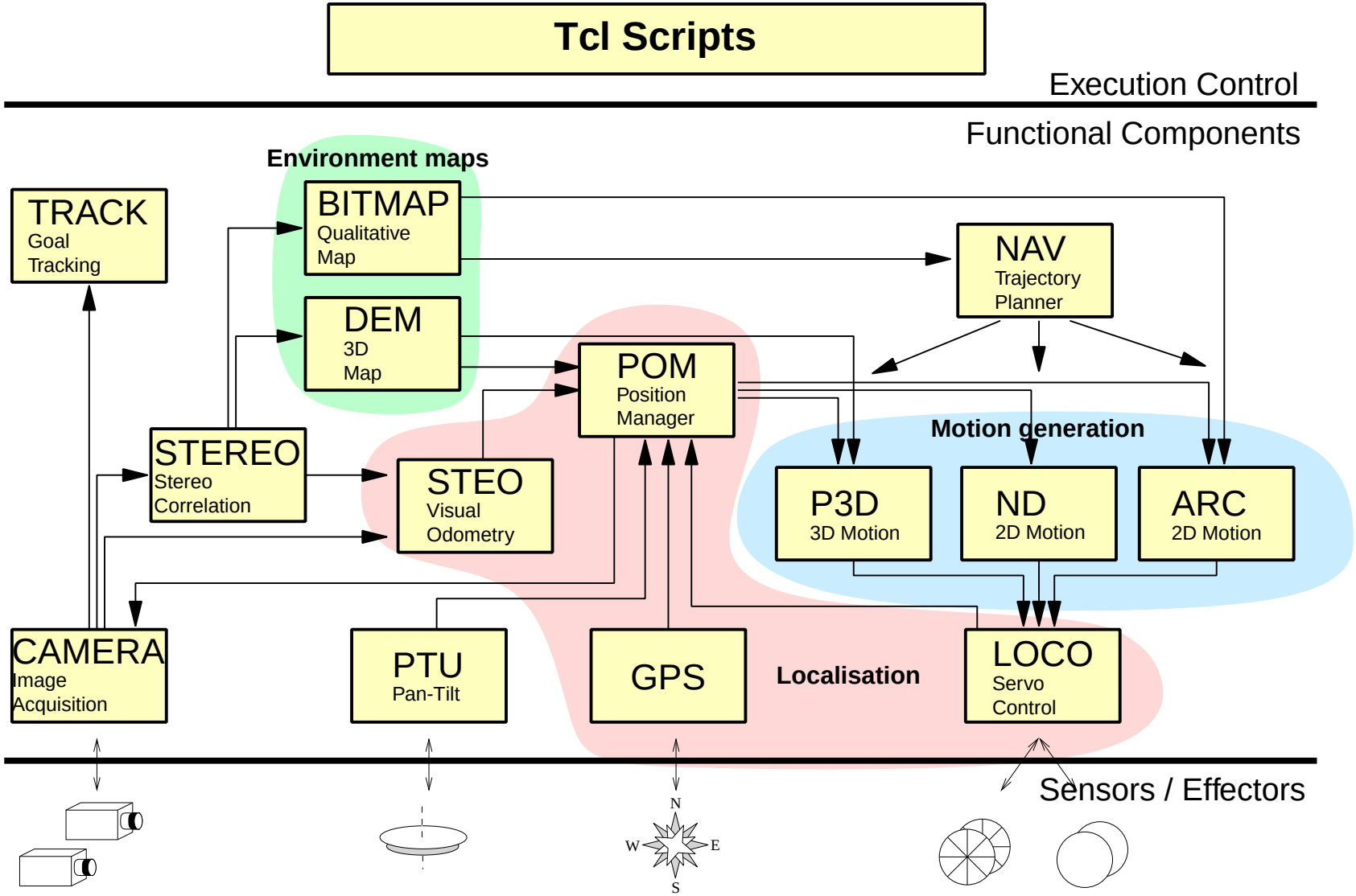
Result:

- one executable program: the module itself,
- libraries to communicate with the module,
- an interactive test program.

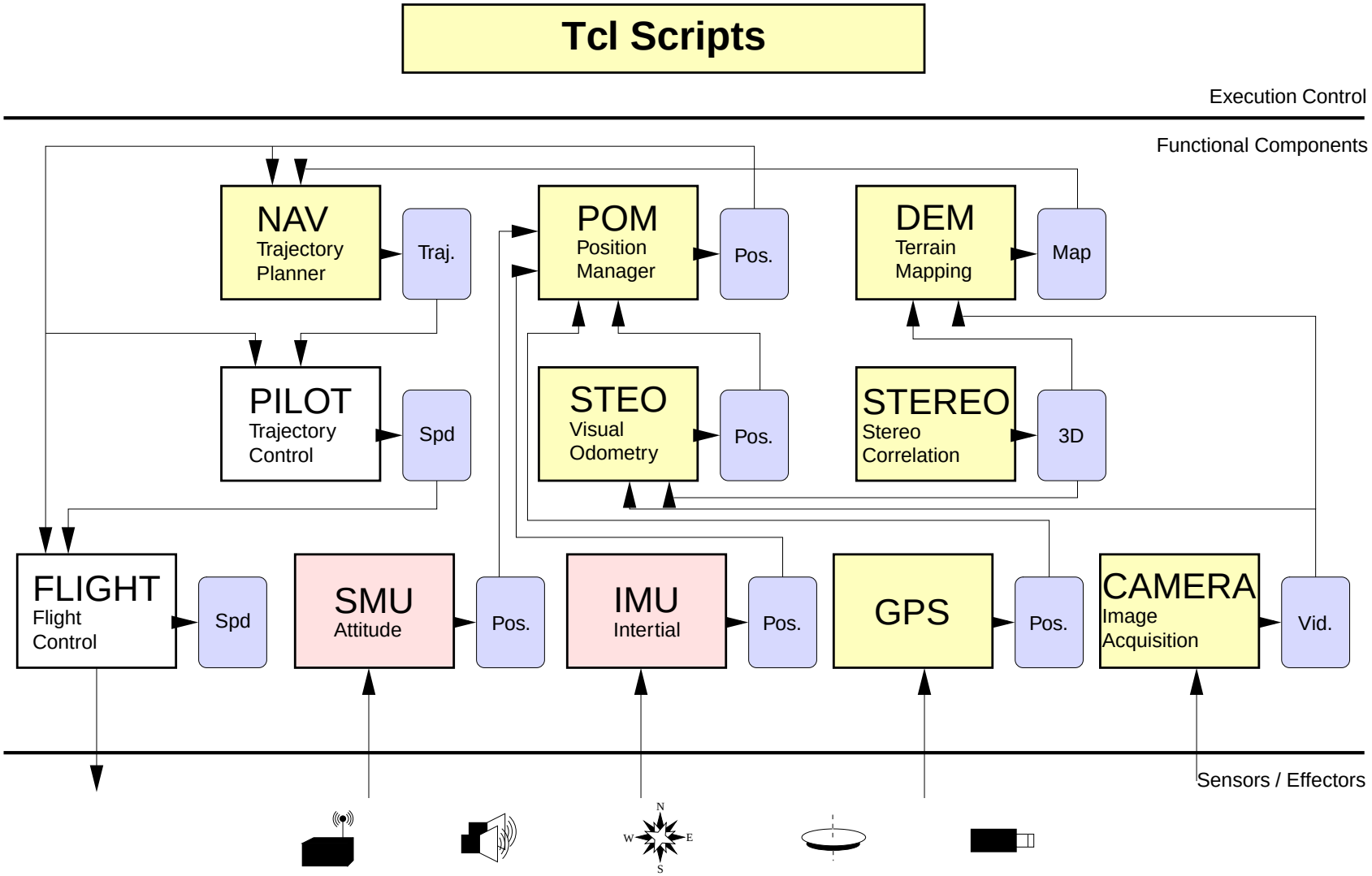
What can a module do ?

- “simple” computations
- device control
- servoing
- monitoring
- use data produced by other modules

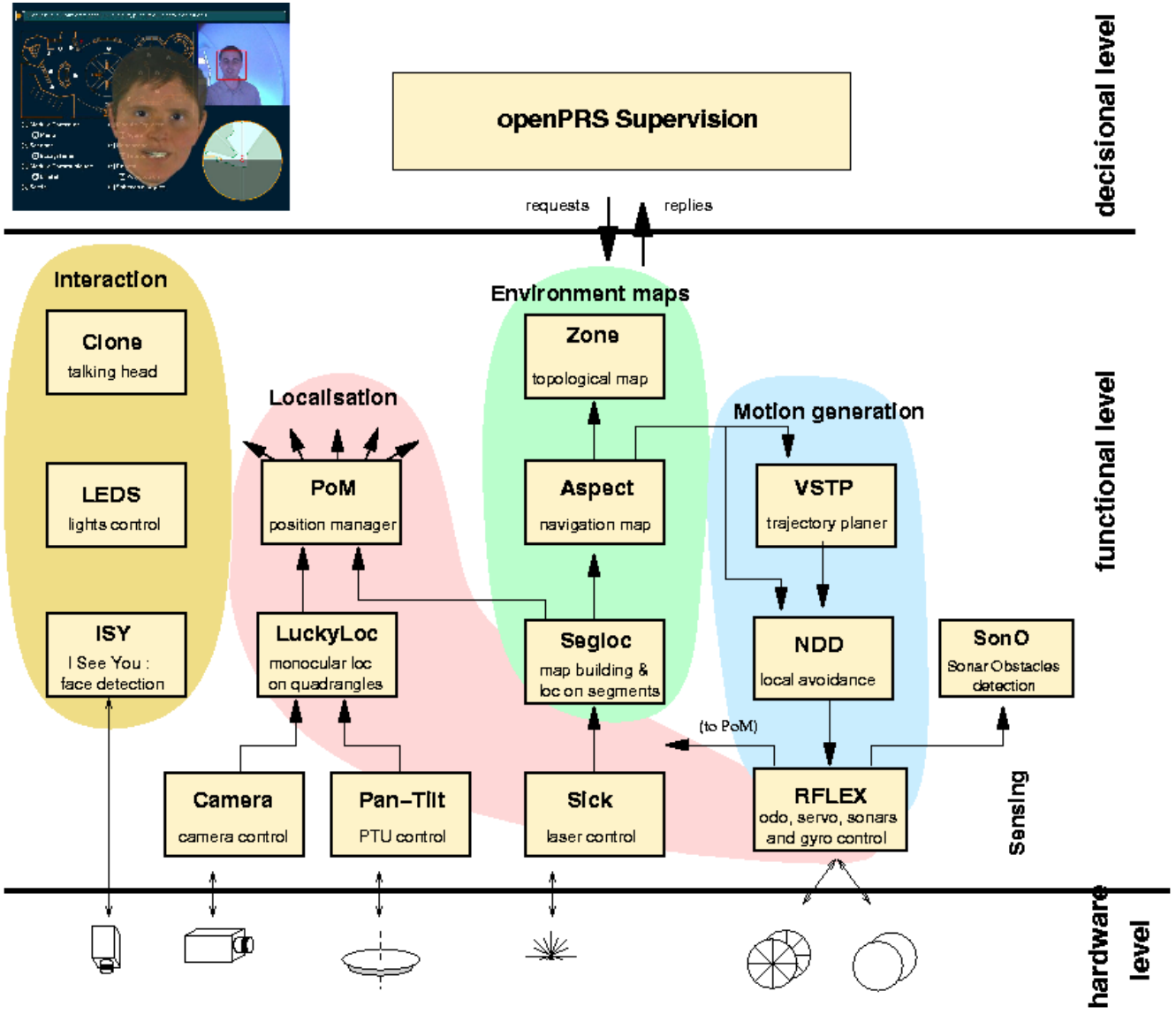
Sample modules: Lama



Sample modules: Karma



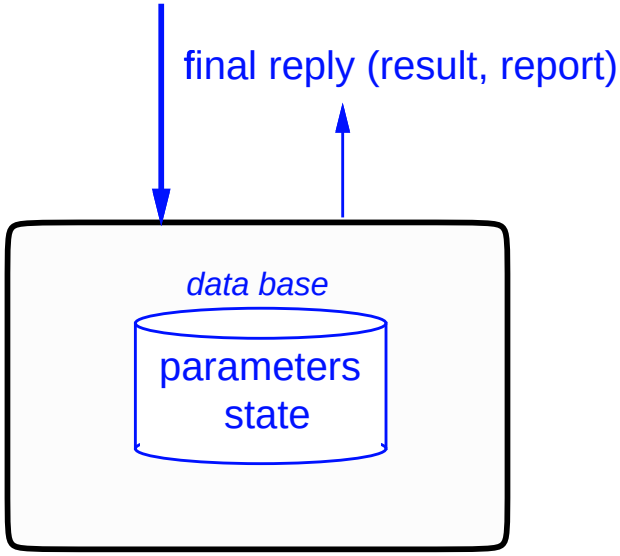
Sample modules: Rackham



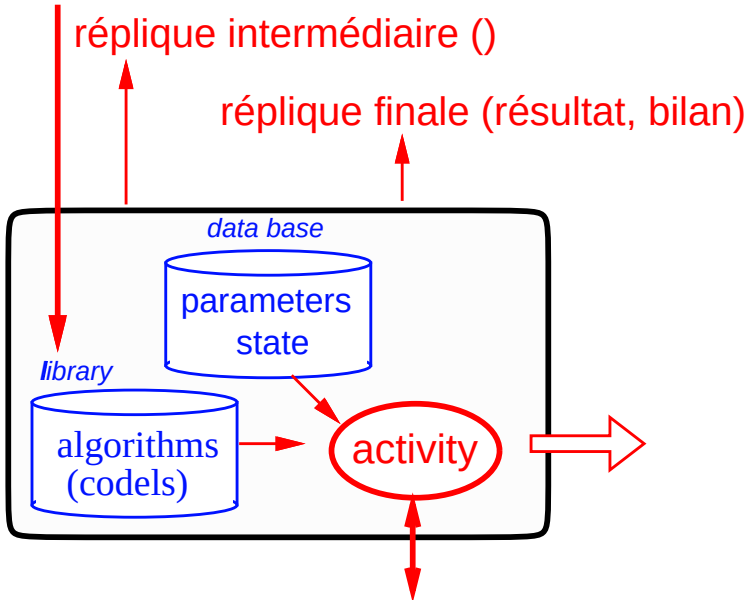
How to communicate with a module?

Requests (`msgLib`)

control request (parameters)



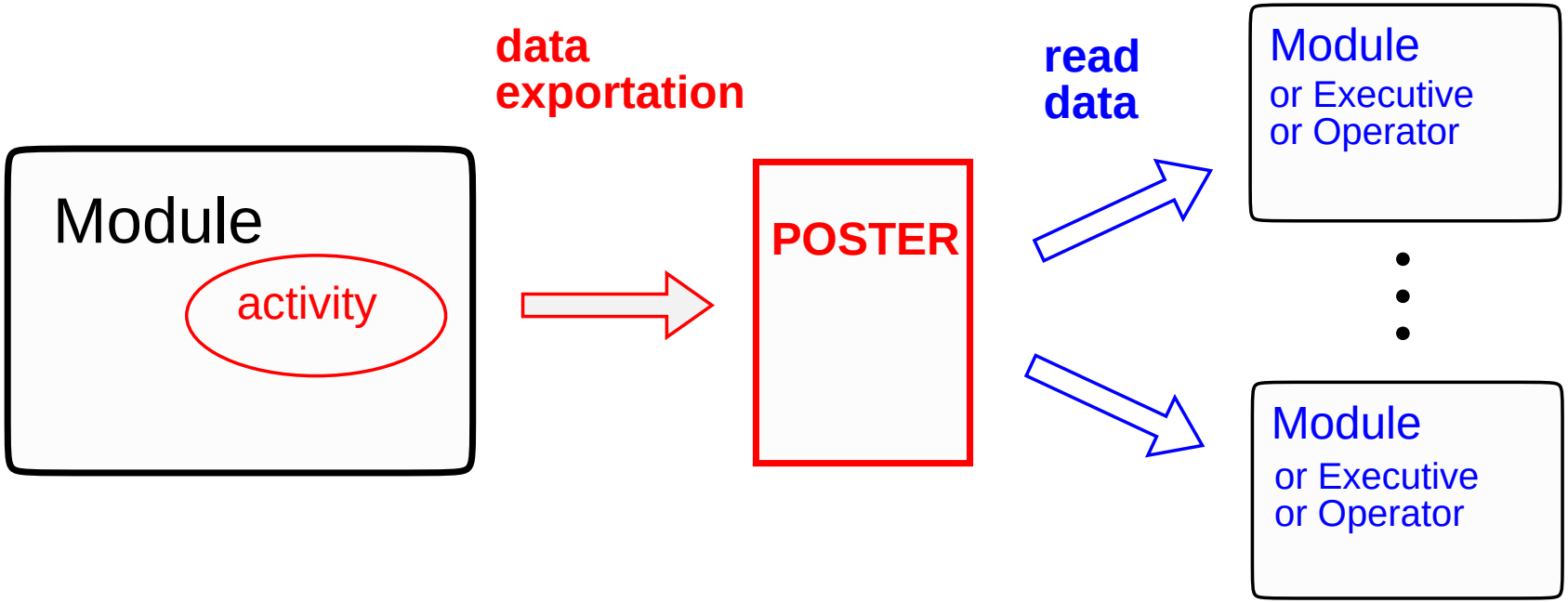
execution request (paramters)



$\rightarrow$ **control flow**

How to communicate with a module?

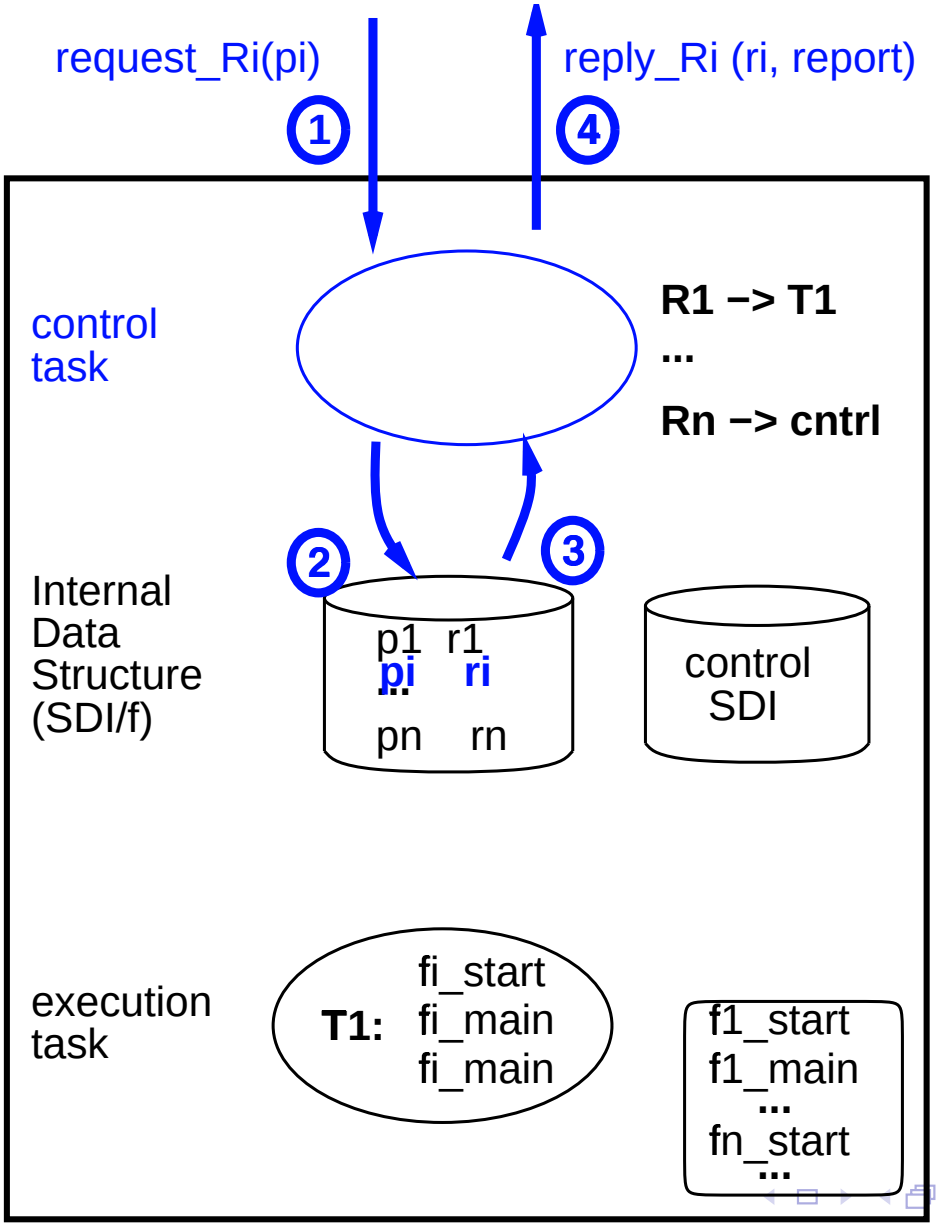
Posters (`posterLib`)



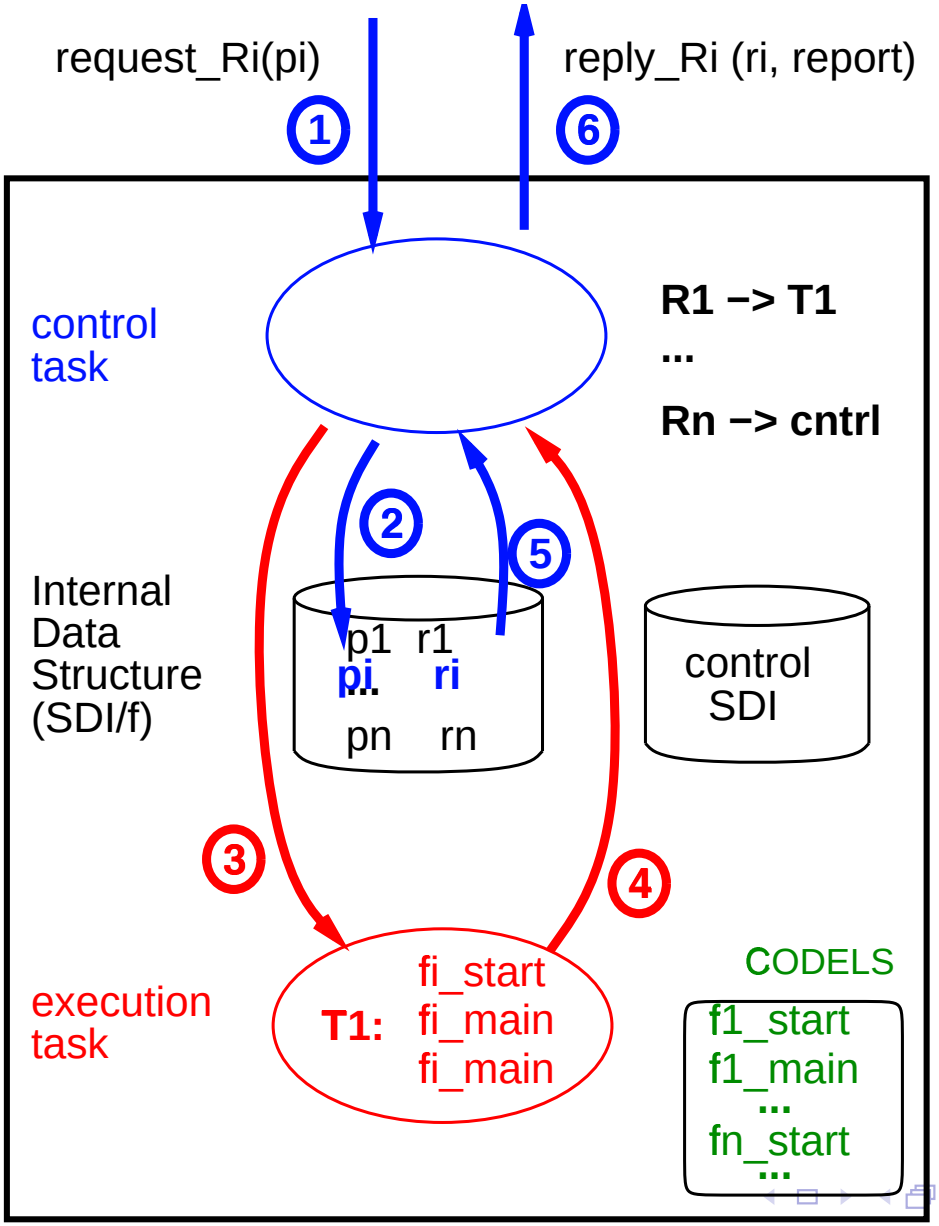
→ **data flow**

`pocolibs` (`http://softs.laas.fr/openrobots`)

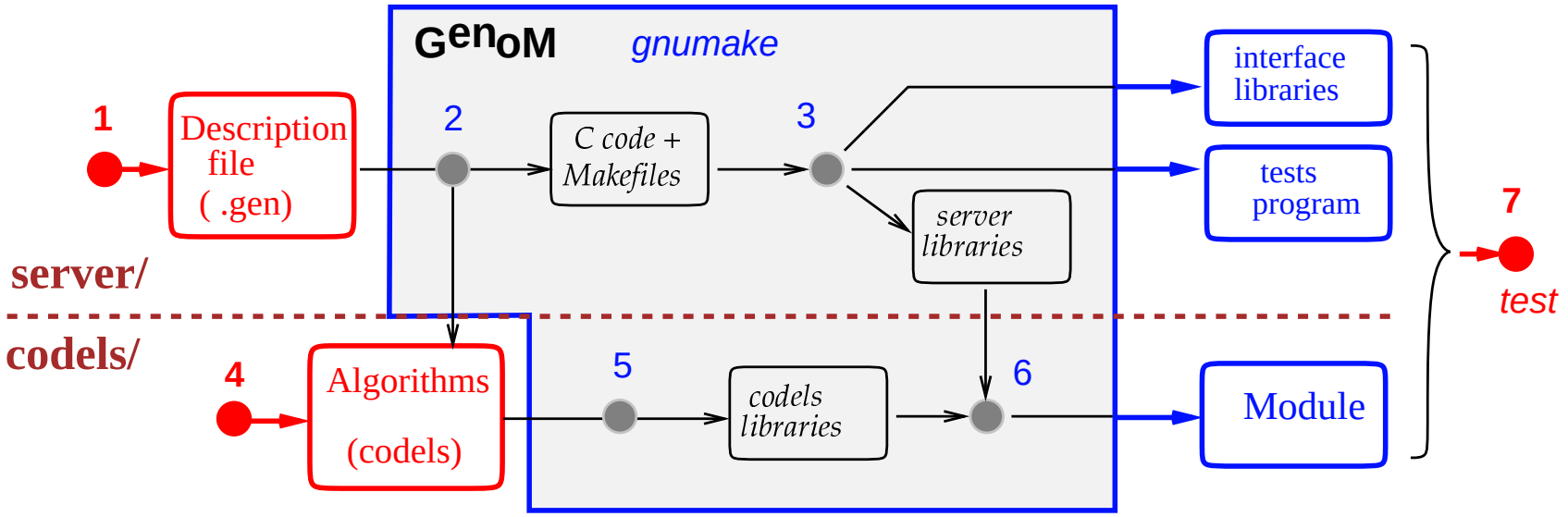
Behavior of a module: control request



Behavior of a module: execution request



Development cycle of a module



7 steps (including 4 “transparent”):

1.	**describe the module:** edit the `.gen` file,	
2. & 3.	generate + build the module (`make`),	iterate
4.	**write the algorithms** (codels),	while
5. & 6.	compile codels +	needed
	link edition with the module (`make`),	
7.	**test the module.**	

Summary of 1st part

Module: database
+ processing functions (set of **codels**)
+ execution engine

Requests: control and execution
Posters: data transfers
Activity: executing process of a request

Agenda

Directories setup

General configuration:

- Use *Robotpkg* to install genom and its dependencies.
- Openrobots tools installed in `$prefix` (`$HOME/openrobots`)
- Environment variables:
  - PATH `$prefix/bin`
  - PKG_CONFIG_PATH `$prefix/lib/pkgconfig`

Directory layout for demo:

`demo/`	module description data structures	`demo.gen` `demoStruct.h`
`demo/server/`	server	(generated by G$^{en}_{o}$M)
`demo/autoconf/`	autotools scripts	
`demo/codels/`	algorithms (codels)	`demoXxxCodels.c`

Components of a module description

The `demo.gen` file describes:

- module (name + identifier)
- internal database (C structure)
- requests
- posters
- execution tasks

Edition of the module: XEmacs genom mode

Configuration file `.emacs.d/init.el`

```
(setq load-path
  (cons (expand-file-name "~/openrobots/share/genom")
        load-path))
(setq auto-mode-alist
  (cons '("\\.gen$" . genom-mode) auto-mode-alist))
(autoload 'genom-mode "genom-mode" "Genom-mode" t)
```

(currently broken with modern emacs versions)

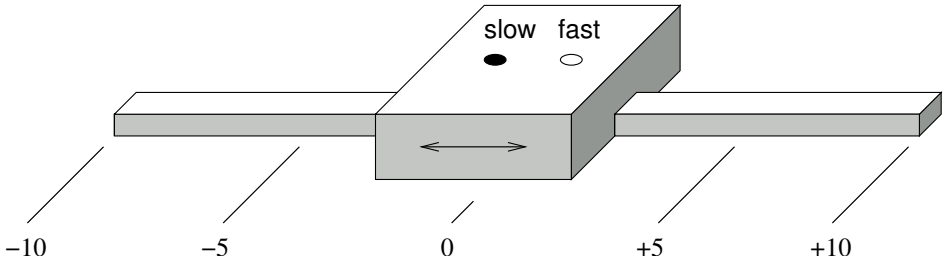
Emacs Genom mode

Commands (or menu):

`C-c C-m`	**m**odule creation (*1st command to call*)
`C-c C-i`	**i**mportation de structures
`C-c C-r`	**r**equest creation
`C-c C-p`	**p**oster creation
`C-c C-e`	**e**xecution task creation
`C-c C-b`	**b**uffer indentation
`C-c C-v`	**v**erify fields syntax
`C-c C-d`	**d**elete unused fields
`C-c C-h ...`	**h**elp on line

A first module example

How to control the motion of a mobile moving along one axis ?



- **One main service**: "move the mobile to a given position"
  ⇒ **execution request**: `Goto`
  - input: `position` to reach
- **parameters control**: "get or change speed reference"
  ⇒ **control request**:
  - `SetSpeed` with input: new speed reference
  - `GetSpeed` with output: current speed reference

Edition of demo.gen: module declaration

C-c C-m demo

```
/*-------------------------------------------------------------
 *                  --- Module DEMO ---
 *
 *  Description:        The only purpose is to do a demo
 *  Creation date:      Sat Jul 15 13:20:11 CEST 2006
 *  Author:             Sara Fleury
 *-------------------------------------------------------------

module demo {
      number:           <<number>>;
      version:          "0.1";
      email:            <email>;
      requires:         <package-or-module> ...;
      internal_data:    DEMO_STR;
      uses_cxx:         0;
};

/*-------------------------------------------------------------
 *                  Structures and IDS
 *-------------------------------------------------------------*/
#include "demoStruct.h"

typedef struct DEMOCAL_STR {

} DEMOCAL_STR;
```

demo.gen: creation of a control request

```
/*-----------------------------------------------------------------
 *                Requests
 *-----------------------------------------------------------------*/

request SetSpeed {
    doc:                    "<some doc>";
    type:                   control;
    input:                  <name>::<sdi-ref>;
    input_info:             <default-value>::"<doc>" , ...;
    output:                 <name>::<sdi-ref>;
    c_control_func:         demoSetSpeedCntrl;
    fail_msg:               <msg-name> , ...;
    interrupt_activity:     <exec-rqst-name> , ...;
};
```

demo.gen: creation of an execution request

```
request Goto {
    doc:                    "<some doc>";
    type:                   exec;
    exec_task:              <<exec-task-name>>;
    input:                  <name>::<sdi-ref>;
    input_info:             <default-value>::"<doc>" , ...;
    posters_input:          <struct-name>, ...;
    output:                 <name>::<sdi-ref>;
    c_control_func:         demoGotoCntrl;
    c_exec_func_start:      demoGotoStart;
    c_exec_func:            demoGotoExec;
    c_exec_func_end:        demoGotoEnd;
    c_exec_func_inter:      demoGotoInter;
    fail_msg:               <msg-name> , ...;
    interrupt_activity:     Goto, <exec-rqst-name>, ...;
};
```

demo.gen: creation of a poster

```
poster Status {
    update:                 auto;
    data:                   <<name>>::<<sdi-ref>> , ...;
    exec_task:              <<exec-task-name>>;
};
```

demo.gen: creation of an execution task

```
exec_task MotionTask {
    period:                 <number>;
    delay:                  <number>;
    priority:               <<number>>;
    stack_size:             <<number>>;
    c_init_func:            demoMotionTaskInit;
    c_func:                 demoMotionTaskPerm;
    c_end_func:             demoMotionTaskEnd;
    fail_msg:               <msg-name> , ...;
};
```

Description of the module demo.gen

```
/* ——— Module declaration ——— */
module demo {
      number:                 9000;
      internal_data:          DEMO_STR;
      version:          "0.1";
      email:            sara@laas.fr;
      internal_data:    DEMO_STR;
      uses_cxx:         0;
};

/* ——— Structure definitions ——— */
#include "demoStruct.h"
#include "demoConst.h"

/* ——— Database of the module ——— */
typedef struct DEMO_STR {
      DEMO_STATE_STR state;         /* Current state */
      DEMO_SPEED     speedRef;      /* Speed reference */
      double         posRef;        /* Position reference */
      double         monitor;       /* Positions monitors */
}DEMO_STR;
```

Description of the module demo.gen (cont.)

```
/* ——— declaration of the services: the requests ——— */

/* Control requests */
request SetSpeed {
    doc:                "To change speed";
    type:               control;
    input:              speed::speedRef;
    input_info:         DEMO_DEFAULT_SPEED::"DEMO_SLOW or DEMO_FAST";
    c_control_func:     controlSpeed;
    fail_msg:           INVALID_SPEED;
};

request GetSpeed {
    doc:                "To get current speed value";
    type:               control;
    output:             speed::speedRef;
};
```

Description of the module demo.gen (cont. 2)

```
/* Execution requests */
request Goto {
    doc:                    "Goto the given position";
    type:                   exec;
    input:                  goal::posRef;
    input_info:             0::"position in m";
    c_control_func:         demoGotoCntrl;
    fail_msg:               TOO_FAR_AWAY;
    c_exec_func_start:      demoGotoStart;
    c_exec_func:            demoGotoExec;
    c_exec_func_end:        demoGotoEnd;
    c_exec_func_inter:      demoGotoEnd;
    interrupt_activity:     Goto;
    exec_task:              MotionTask;
};
```

demo: structures and constants

```
#ifndef DEMO_STRUCT_H
#define DEMO_STRUCT_H

typedef struct DEMO_STATE_STR {
  double position;  /* currente position (m) */
  double speed;     /* current speed (m/s) */
} DEMO_STATE_STR;

typedef enum DEMO_SPEED {
  DEMO_SLOW,
  DEMO_FAST
} DEMO_SPEED;
#endif
```

```
#ifndef DEMO_CONST_H
#define DEMO_CONST_H
#define DEMO_MACHINE_LENGTH     20.0        /* m */
#define DEMO_DEFAULT_SPEED      DEMO_FAST   /* DEMO_SLOW */
#endif
```

Summary of 2nd part

3 files to describe a module:

- description file `demo.gen`
- structures `demoStruct.h`
- constants `demoConst.h`

5 parts in the **.gen** file:

- module (name + identifier)
- database (data structures)
- requests
- posters
- execution tasks

Agenda

First generation of the module: genom demo.gen

```
blues% genom demo.gen
genom demo.gen: info: array MonitorInput added in SDI for request Monitor
genom demo.gen: info: array MonitorOutput added in SDI for request Monitor
perl -w ./demo.pl

Updating top directory
  creating autogen

Updating codels
  demoMotionTaskCodels.c changed, skipping
  demoCntrlTaskCodels.c changed, skipping
  Makefile.in changed, skipping

Updating autoconf
  creating genom.mk
...
Updating server
  creating Makefile.in
...
Creating build environment ...
  * Running aclocal
  * Running autoconf

If you already have a build of this module, do not forget to
reconfigure (for instance by running ./config.status --recheck)

Done.
```

GenoM outputs

Production of several files and directories:

- `Makefile.in`, `configure`...
- `server/`, `autoconf` and `codels/`

autogen*	configure.ac.user	local.mk.in
Makefile.in	demo.gen	server/
acinclude.user.m4	codels/	demoConst.h
autoconf/	configure*	demoStruct.h

GenoM outputs: the server

In `demo/server/` files used by genom and clients of the module, related to:

- server: `demoCntrlTask, demoMotionTask, ...`
- libraries to send requests and receive replies `demoMsgLib`
- libraries to access posters: `demoPosterLib`
- Test program: `demoTest.c`

GenoM outputs: codels

In `demo/codels/` files that will contain the algorithms.
1 codels file per task:

- Control codels: `democCntrlTaskCodels.c`
- Execution codels: `demoMotionTaskCodels.c`
- Autoconf infrastructure: `Makefile.in` `autoconf/configure`

Configuring the module

Create a build directory:

```
blues% mkdir build
blues% cd build
```

Call configure, telling where to install the module:

```
blues% ../configure --prefix=${HOME}/openrobots
checking build system type... i686-pc-linux-gnu
checking host system type... i686-pc-linux-gnu
checking for gcc... gcc
checking for C compiler default output file name... a.out
...
config.status: creating codels/Makefile
config.status: creating server/Makefile
config.status: creating demo.pc
config.status: creating local.mk
```

Building the module

Call: `make`

- re-generates the module if needed
- builds everything

```
blues% make
make all-posix
make[1]: Entering directory '/home/foo/openrobots/modules/demo/build'
make[2]: Entering directory '/home/foo/openrobots/modules/demo/build/server'
mkdir -p posix-build
/home/foor/openrobots/bin/mkdep -c"gcc" -oposix-build/dependencies -dposix-build
-t.lo -DUNIX    -I. -I../.. -I../../server -I/home/foo/openrobots/include
../../server/demoCntrlTask.c ../../server/demoModuleInit
...
creating posix-build/demo
make[2]: Leaving directory '/home/foo/openrobots/modules/demo/build/codels'
make[1]: Leaving directory '/home/foo/openrobots/modules/demo/build'
```

Installation

The module needs to be installed to be useful.

```
blues% make install
```

It is installed as specify with the `-prefix` option of `configure`. According to the destination you have chosen, you may need `root` priviledges.

Summary of 3rd part

1. Edit the module: **demo.gen**
2. Only the first time: generate **genom** and configure **configure**
3. Compile and generate: **make**
4. Install: **make install**

Agenda

UNIX session: execution

1. Start the communication server:

```
blues% h2 init
```

2. Start the module:

```
blues% demo -b
```

-b option allows to wait until the module has effectively started.

3. Start the client(s):

```
blues% demoTest 1
```

The test program demoTest

```
blues% demoTest 1
pocolibs execution environment version 2.8
Copyright (c) 1999-2010 CNRS-LAAS
client init ...ok.   Poster init ...ok.
---------------------------------------------------------------------------
  0: SetSpeed                2: Stop                  4: GotoPosition (E)
  1: GetSpeed                3: MoveDistance (E)      5: Monitor (nE)

55: posters  66: abort  77: replies(0)  88: state  99: QUIT  -99: END
---------------------------------------------------------------------------
demo1 (88)>
```

The test program demoTest (cont)

```
demo1 (88)> 3

Get current distRef using poster Mobileref (y/n) ? (n)

— Enter double distRef:  (0.000000) 1.0
Wait final reply (y/n/a) ? : y

Activity 0 started
start engine
stop engine
Final reply: OK


  0: SetSpeed                2: Stop                  4: GotoPosition (E)
  1: GetSpeed                3: MoveDistance (E)      5: Monitor (nE)

55: posters  66: abort  77: replies(0)  88: state  99: QUIT  —99: END

demo1 (3)>
```

UNIX session: end

1. Kill the module:

```
blues% killmodule demo
```

   or with the command `-99` in the `demoTest` menu.
   (then one can restart the module)

2. end the session:

```
blues% h2 end
```

Summary of 4th part

1. Edit the module: **demo.gen**
2. Only the first time: generate (**genom**) and configure (**configure**)
3. (Re) generate and compile: **make**
4. Install: **make install**
5. Execute: **h2 init, demo -b, demoTest 1, ... h2 end**

Agenda

Codels

- All the codels are in the directory: `codels/`
- In this directory we have one codels file per task.
- Thus for the demo we have:
  - `demoCntrlTaskCodels.c` for the control codels (`c_control_func`)
  - `demoMotionTaskCodels.c` for the execution codels (`c_exec_func, ...`)

Control codels

```
STATUS demoSetSpeedCntrl(DEMO_SPEED *speed, int *report)
{
   /* Refuse *speed if the value is erroneous */
   if (*speed != DEMO_SLOW && *speed != DEMO_FAST) {
        *report = S_demo_INVALID_PARAMETER; return ERROR;
   }
   /* Parameter is valid: it will be recorded into the fIDS */
   return OK;
}

STATUS demoGotoCntrl(double *posRef, int *report)
{
   /* Refuse *posRef if the value is erroneous */
   if (fabs(*posRef) > DEMO_MCHINE_LENGTH/2.0) {
        *report = S_demo_INVALID_PARAMETER; return ERROR;
   }
   /* Parameter is valid: it will be recorded into the fIDS
      and Goto request will start */
   return OK;
}
```

Execution codels

```
ACTIVITY_EVENT demoGotoStart(double *posRef, int *report)
{
   printf("Start engine\n");
   return EXEC;
}

ACTIVITY_EVENT demoGotoExec(double *posRef, int *report)
{
   double dist = *posRef - SDI_f->state.position;
   switch(SDI_f->speedRef) {
       case DEMO_SLOW:
            if (fabs(dist) < 0.1) return END;
            SDI_f->state.position += sign(dist) * 0.1;
            return EXEC;
       case DEMO_FAST:
            if (fabs(dist) < 1) return END;
            SDI_f->state.position += sign(dist) * 1;
            return EXEC;
   }
}
```

Execution codels (cont)

```
ACTIVITY_EVENT demoGotoEnd(double *posRef, int *report)
{
   SDI_f->state.position = *posRef;
   printf("Stop engin\n");
   return ETHER;
}
```

Specific codels

- Initialisation codel `c_init_func`
- Terminaison codel `c_end_func`
- Permanent activity codel: `c_func`

→ one per execution task

```
#include "demoConst.h"

STATUS demoComputeInit(int *report)
{
  SDI_F->state.position = 0.0;
  SDI_F->state.speed = 0.0;
  SDI_F->posRef = 0.0
  SDI_F->speedRef = DEMO_SLOW
  return OK;
}
```

Summary of 5th part

1. Edit the module: **demo.gen**
2. Only the first time: generate (**genom**) and configure (**configure**)
3. (Re) generate and compile: **make**
4. Install: **make install**
5. Execute: **h2 init, demo -b, demoTest 1, ... h2 end**
6. Fill-in the codels (and goto **3**)

Agenda

Activities and codels

- generally: 1 activity $\rightarrow$ a sequence of codels:

```
start
exec
exec
...
exec
end
```

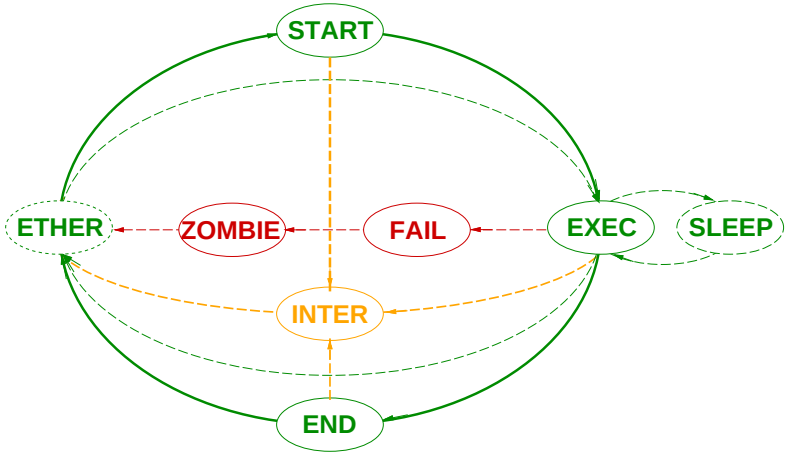
- Codels are not interruptible
- Activity are interruptible on transitions between 2 codels.
- On Interruption $\Rightarrow$ execution of the codel `inter`

Execution requests and codels

One example:

```
/* Translation on a given distance */
request Goto {
   type:                exec;
   input:               distance::distRef;
   c_control_func:      demoGotoCntrl;
   fail_msg:            TOO_FAR_AWAY;
   c_exec_func_start:   demoGotoStart;
   c_exec_func:         demoGotoExec;
   c_exec_func_end:     demoGotoEnd;
   c_exec_func_inter:   demoGotoInter;
   interrupt_activity:  Goto;
   exec_task:           MotionTask;
};
```

Activities decomposition in codels



état	codel (if exists)	
START	c_exec_func_start	starting
EXEC	c_exec_func	main
END	c_exec_func_end	terminating
FAIL	c_exec_func_fail	terminating (pb)
INTER	c_exec_func_inter	terminating (on interruption)
SLEEP		wait event
ETHER		*activity over*
ZOMBIE		*activity suspended*

Periodical task

→ Periodical task

```
exec_task Compute {
    period:                 10;     /* 100ms */
    priority:               20;     /* a hight priority */
    stack_size:             10000;
};
```

- usely 1 tic = 10ms on Unix/Linux
- priority: [0..255] (0: higher priority):
  used only on real-time OS
- stack_size: used only on real-time OS

The posters: automatics or manual

Exporting results

- Automatical update every time its execution task wakes up

```
poster Status {
    update:                 auto;
    data:                   state::state;
    exec_task:              MotionTask;
};
```

- Manual update only when necessary

```
poster Status {
    update:                 user;
    type:                   SPEED_REF;
    exec_task:              MotionTask;
};
```

Posters: manual update

Update from a codel:

```
ACTIVITY_EVENT demoGotoStart(double *posRef, int *report)
{
   printf("Start engin\n");
   if (demoStatusSPEED_REFPosterWrite
                            (DEMO_STATUS_POSTER_ID,
                             SDI_F->speedRef) != OK) {
    *report = S_demo_CANNOT_WRITE_POSTER;
    return ETHER;
  }
   return EXEC;
}
```

Summary of 6th part

1. Edit the module: **demo.gen**
2. Only the first time: generate (**genom**) and configure (**configure**)
3. (Re) generate and compile: **make**
4. Install: **make install**
5. Execute: **h2 init, demo -b, demoTest 1, ... h2 end**
6. Fill-in the codels (and goto **3**)

Agenda

Using module structures from another module

Import structures:

```
module pilo {
      number:           9000;
      version:          "0.1";
      requires:         demo;
      internal_data:    PILO_STR;
};

import from demo {
#include "demoStruct.h"
};
#include "piloStruct.h"

typedef struct PILO_STR {
      PILO_GOTO_STR goto;
      DEMO_POS_REF  posRef;
} PILO_STR;
```

Based on `pkgconfig`.

Read a poster from another module

```
request Track {
   type:   exec;
   ...
   posters_input:    LOCO_SPEED_REF;
   ...
};
```

Library: demoPosterReadLib

```
  POSTER_ID trackPosterId=-1;
  DEMO_REF ref;

  /* look for the poster */
  if (posterFind(track->posterName, &trackPosterId) == ERROR) {
    *report = S_demo_CANNOT_FIND_POSTER;
    return ETHER;
  }
  /* try to read the data for the first time */
  if (demoDEMO_REFPosterRead(trackPosterId, &ref) == ERROR) {
    *report = S_demo_CANNOT_READ_POSTER;
    return ETHER;
  }
...
```

Control of a module with tclserv: principle

- `tclServ` allows to control a set of modules from a unique interface
- Full interactive programming language (Tcl)
- ASCII data transfers (non sensitive to data representation like little endian / big endian / alignment).
- Graphical libraries Tk

`http://www.openrobots.org/wiki/eltclsh`

Module control with tclserv: usage

1. Generate the module with option `-t`
2. Start the server `tclserv`.
3. Start one (or several) tcl-shells:
   `eltclsh -package genom`
4. Start session (see example).

Module control with tclserv: options and commands

option	function
`-ack`	non blocking request sending
`-raw`	the result is presented as a raw list of data (not a nice display)

command	function
`replyof $rqstId`	get answer *blocking mode*. Generaly used with option `-raw`.
`cs::term $rqstId 0`	get answer *non blocking mode*.
`abort $rqstId`	abort the request
`kill <module>`	abort the module <module>
`die`	terminate tclserv and eltclsh itself

Module control with tclserv: examples

```
eltclsh > connect localhost
connecting to localhost:9472
connected to localhost
eltclsh> lm demo
(re)starting tclServ cs daemon on port 9472
demo loaded on localhost from ~/openrobots/share/modules
eltclsh > ::demo::MoveDistance
distance in m [0]> 0.5
status = OK
eltclsh >
```

Module control with tclserv: examples (cont.)

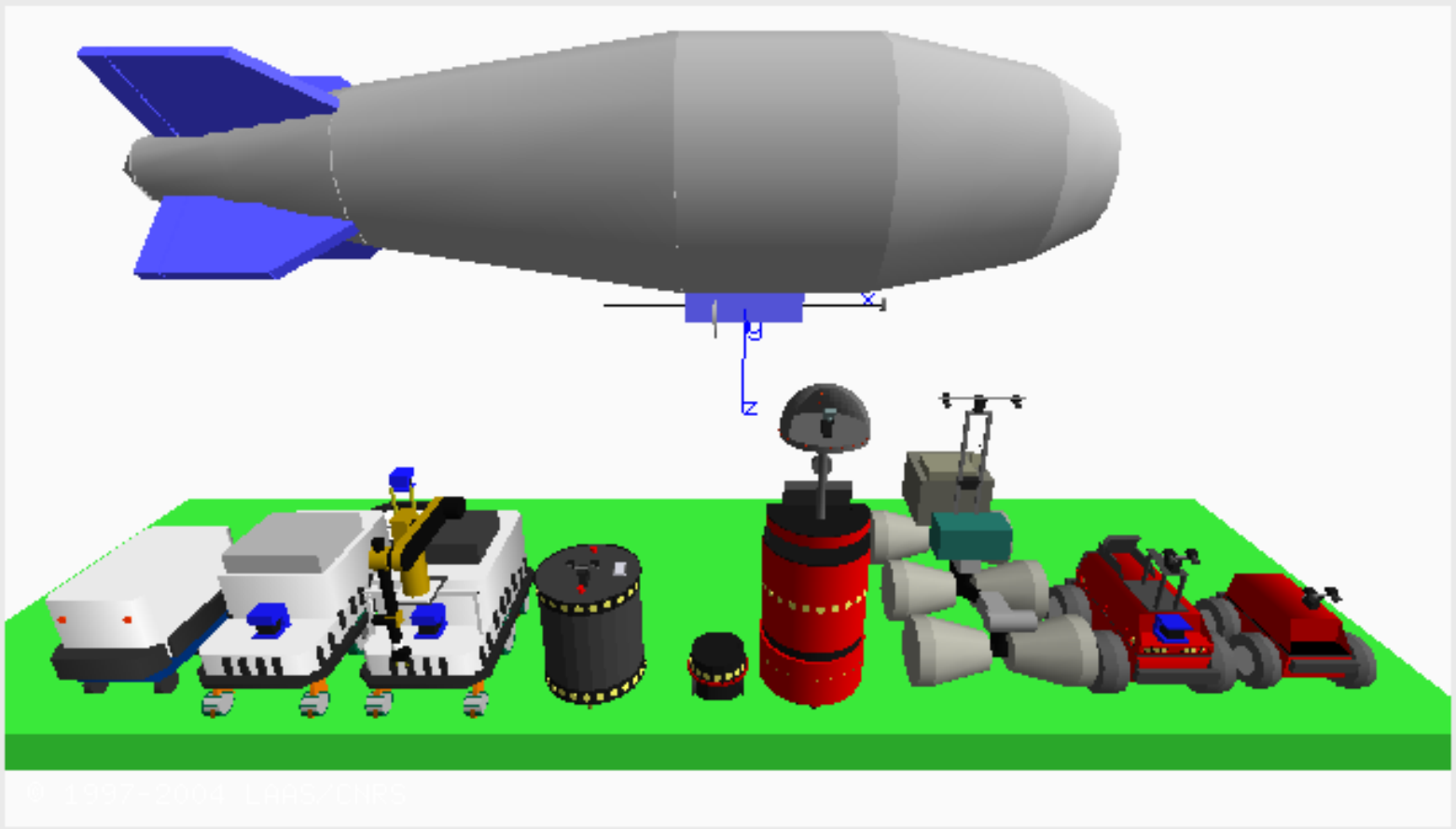
```
set demFused [dem::FuseDem -ack]
set classifDone [lclassif::Classif -ack]
replyof $demFused
```

Module control with tclserv: Tk and Gdhe

Modular and extensible 3D graphical interface with the Tk GUI toolkit:

- Gdhe `http://gdhe.openrobots.org/`

3D model of many robots (rackham, jido, dala, mana,...).



Module control with OpenPRS: principle

`http://openprs.openrobots.org/`

- Supervision
- Generate the module with option `-o`

OPs producted for each request:

nom	function
`DEMO-MOTION`	send the request *and* receive the *finale reply in blocking mode*
`DEMO-MOTION-REPORT`	idem with the report
`DEMO-MOTION-ASYNC`	send the request in non blocking mode.

The answer is a string in the data base:

```
(FR DEMO DEMO_MOTION $RQST-ID $REPORT $DATA).
```

Module control with OpenPRS: `transgen`

- Automatic production of OpenPRS relocatable.
- Similar to GenoM:
  - Description file `manip.tg`:

```
/* —— Declaration of a supervisor —— */
supervisor manip {
  module: xr4000;
  module: sick;
  module: m2d;
  module: segloc;
}
```

Module control with OpenPRS: transgen (continued)

- Call: `tranGen manip.tg`
- Compile in `server/`.
- Start OpenPRS:

```
blues% server/i386-linux/manip-xOpenPRS -n manip-cabby \
 -x server/data/manip-data.inc -x data/my-data.inc &
```

Module control with OpenPRS: Example of procedure

An OP that automatically starts the continuous localization request (SEGLOC-LOCLOOP-ASYNC) when the robot losts itself:

```
;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; |Localize|
;;;;;;;;;;;;;;;;;;;;;;;;;;
(defop |Localize|
    :invocation (! (EXECUTE LOC-LOOP))
    :context (& (UNCERTAINTY-LEVEL-1 $X1 $Y1 $T1)
                (ALARM-LEVEL-1 $L1))
    :body ((IF (! (EXECUTE LOC-LOCAL))
               (! (SPEAK "ok"))
               (! (SEGLOC-LOCLOOP-ASYNC $RQST-ID1))
               (! (CURRENT-MISSION-COMPLETED))
             ELSE
               (! (SPEAK "Bad localization. Try again"))
               (! (FAILED))
            )
           )
)
```

Agenda

Summary

1. Edit the module: **demo.gen**
2. First time only: generate (**genom**) and configure (**configure**)
3. (Re) generate and compile: **make**
4. Install: **make install**
5. Execute: **h2 init, demo -b, demoTest 1, ... h2 end**
6. Control with: **tclserv** and **Tcl/Tk** (`eltclsh, elwish, gdhe`) or **OpenPRS**

Online documents

- Official page: `http://genom.openrobots.org/`
- Robotpkg: `http://robotpkg.openrobots.org/`
- Mailling-list: `openrobots@laas.fr`
  `https://sympa.laas.fr/sympa/info/openrobots`
- (Intranet) Wiki: `http:/intranet.laas.fr/robots/wiki`