1. SYSTEM_CONFIGURATION IS

ROOT_OBJECTS

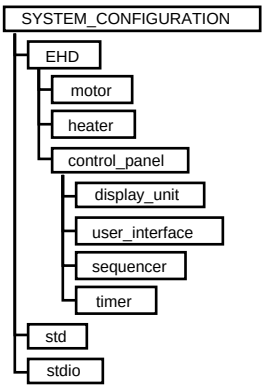
--|\\Kigafarz\home\stood\stood5.0\libs\std|--,

--|\\Kigafarz\home\stood\stood5.0\libs\stdio|--,

--|\\Kigafarz\home\stood\stood5.0\examples\EHD|--

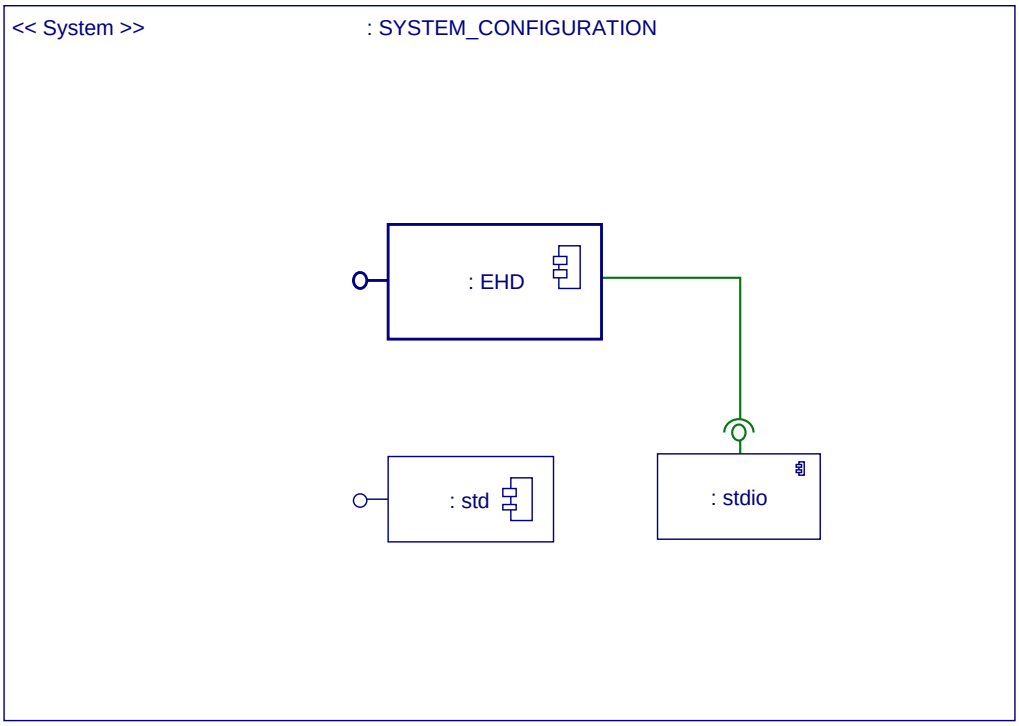
END

1.1. Design Tree

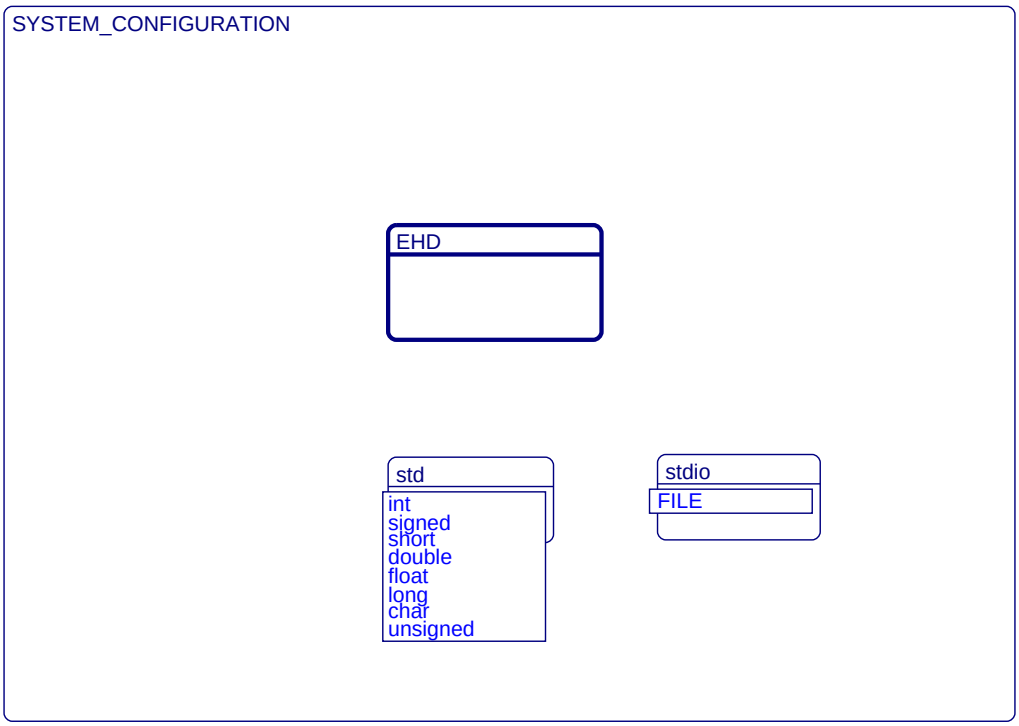


1.2. Inheritance Tree

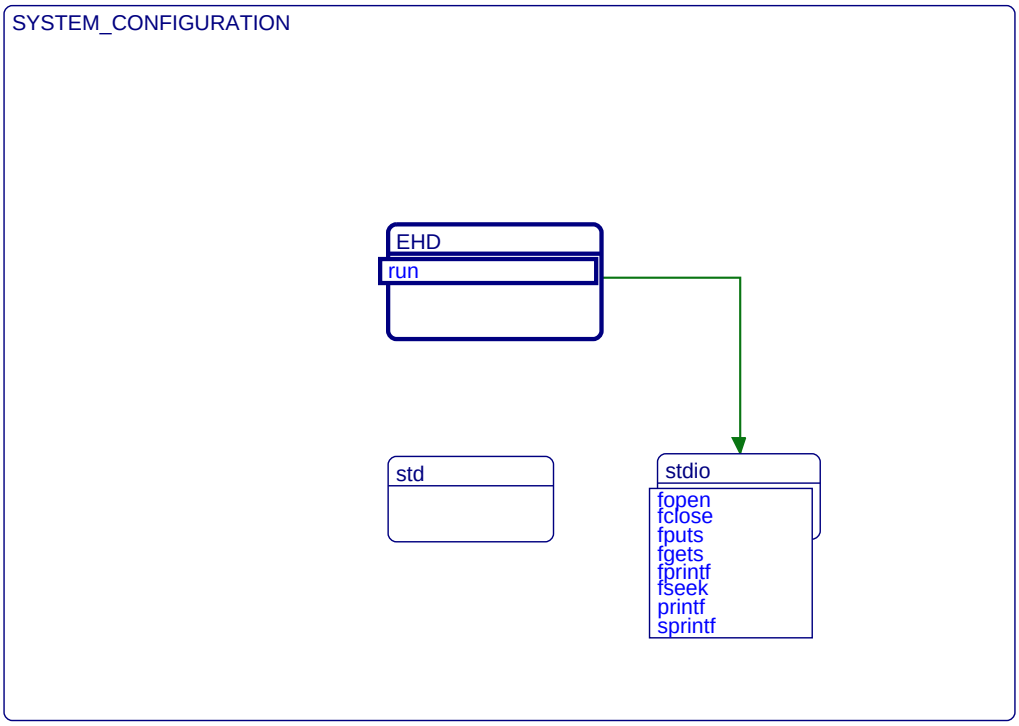
1.3. UML Structure Diagram



1.4. Structural (types) Diagram



1.5. Functional (oper.) Diagram



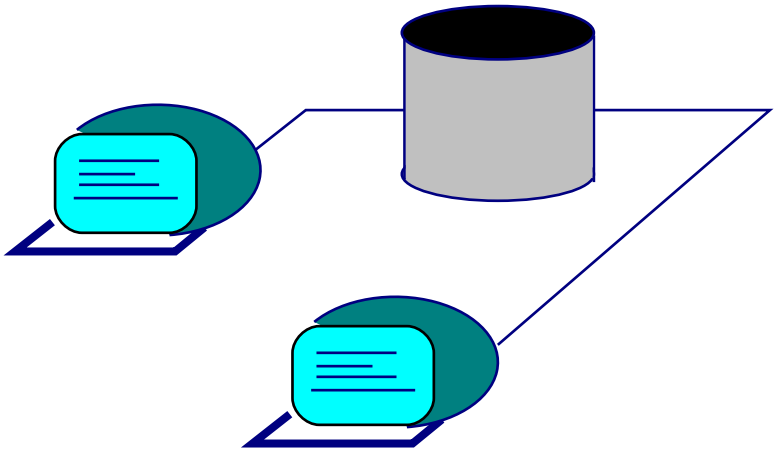
1.6. Project Description (text)

The System Configuration shows the environment of the project.

At this level, software libraries, hardware drivers and other interacting applications can be shown.

The environment of the application will be slightly different depending on wether we are developing the software to be actua
control system, or we are building an ov(cf.apply_power)erall software simulation of the system. In the former case, the envi
interfaces to the harware equipments, whereas in the latter case, only a few C libraries can be included, in order for our mai
able to use them.

1.7. Project Sketch



1.8. Project Table

	date	author	action
version 1	05 Dec 2003	P. Dissaux	sent to ERAU
version 2	01 Mar2004	P. Dissaux	included into release 5.0 b 1

1.9. List of Requirements

apply_power

cycle_heater

display_speed

display_status

display_time

heater_cycle

motor_speed

set_speed

2. OBJECT EHD IS

2.1. PASSIVE

2.2. pragmas

```
PRAGMA reverse
    (option => --|1|--,
     separator => --|no|--)
PRAGMA main
    (operation_name => --|run|--)
PRAGMA cc
    (compiler => --|gcc|--)
PRAGMA static
    (keyword => --|static|--,
     condition => --|no|--)
```

2.3. DESCRIPTION

2.3.1. PROBLEM

2.3.1.1. Statement of the Problem (text)

The component EHD represents the main application to be developed.

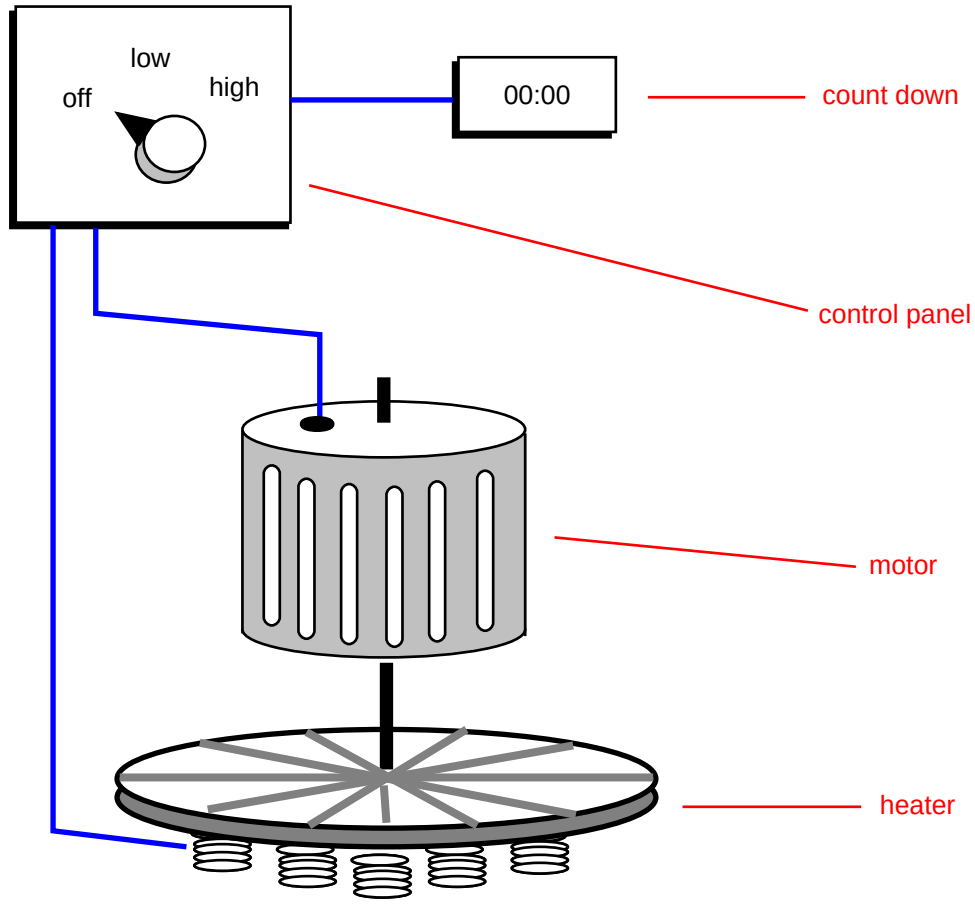
Its black box view shows the EHD within its environment, and only its provided interface is shown.

Its white box view shows the contents of the EHD, in terms of subcomponents and internal connections between them.

This project is a simple example model, based on the following requirements for an Electric Hair Dryer system:

1. The system shall allow user to select motor speed (off, low, or high).
2. The system shall apply power to motor depending on the selected speed setting.
3. The system shall cycle the heater (30 seconds on and 30 seconds off) when in low and high speed modes
4. The system display shall show the selected speed, heater status and the count down time when the heater is on.

2.3.1.2. Sketch of the Problem



2.3.1.3. Referenced Documents (text)

This example have been specified by the ERAU.

2.3.1.4. Analysis of Requirements

2.3.1.4.1. Structural Requirements (text)

The EHD system should contain a motor, a heater and some control/display unit.

2.3.1.4.2. Functionnal Requirements (text)

The motor can be off or work at low or high speed.

The heater can be on or off.

2.3.1.4.3. Behavioural Requirements (text)

The heater can work only if the motor is on.

When the motor is on, the heater must be switched on and off periodically.

2.3.1.5. Local Environment

2.3.1.5.1. Parent General Description (text)

The EHD simulator uses the predefined C types, that are provided by the std library, and basic i/o provided by the stdio libra

2.3.2. SOLUTION

2.3.2.1. General Strategy (text)

The EHD system is developed as a pure software simulator.

Its functional interface thus only consists in a main procedure, used to start the application.

All the other interactions with the outside world will be managed by a user dialog.

2.3.2.2. Identification of Child Modules (text)

The EHD simulator is composed of:

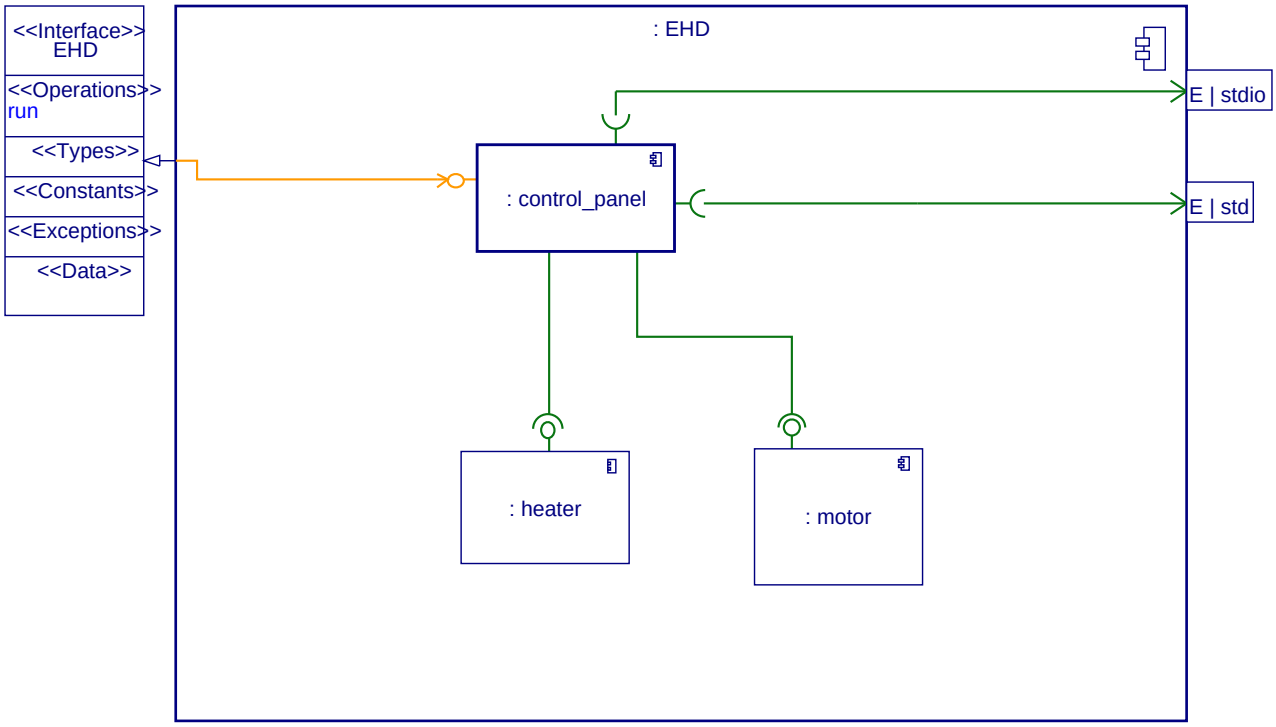
- a control panel to manage the user interface.

- a heater.

- a motor.

2.3.2.3. Structural Description

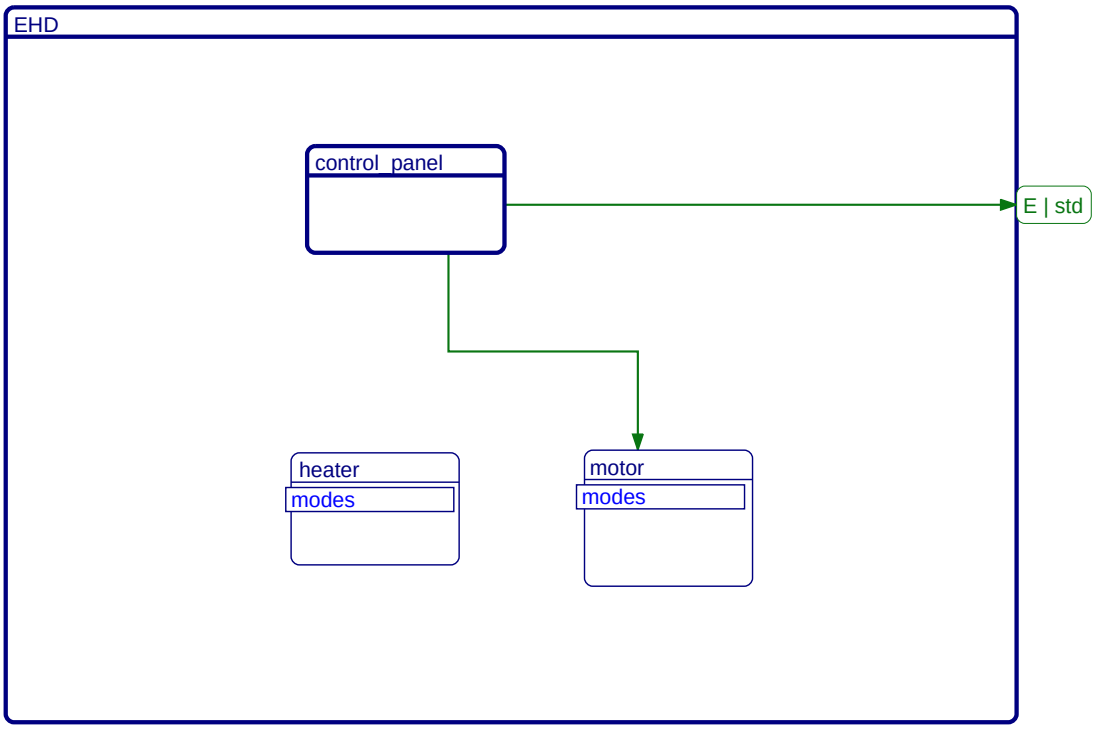
2.3.2.3.1. UML Structure Diagram



2.3.2.3.2. Identification of Data Structures (text)

At the higher level, the EHD simulator doesn't provide any data structure.

2.3.2.3.3. Structural (types) Diagram

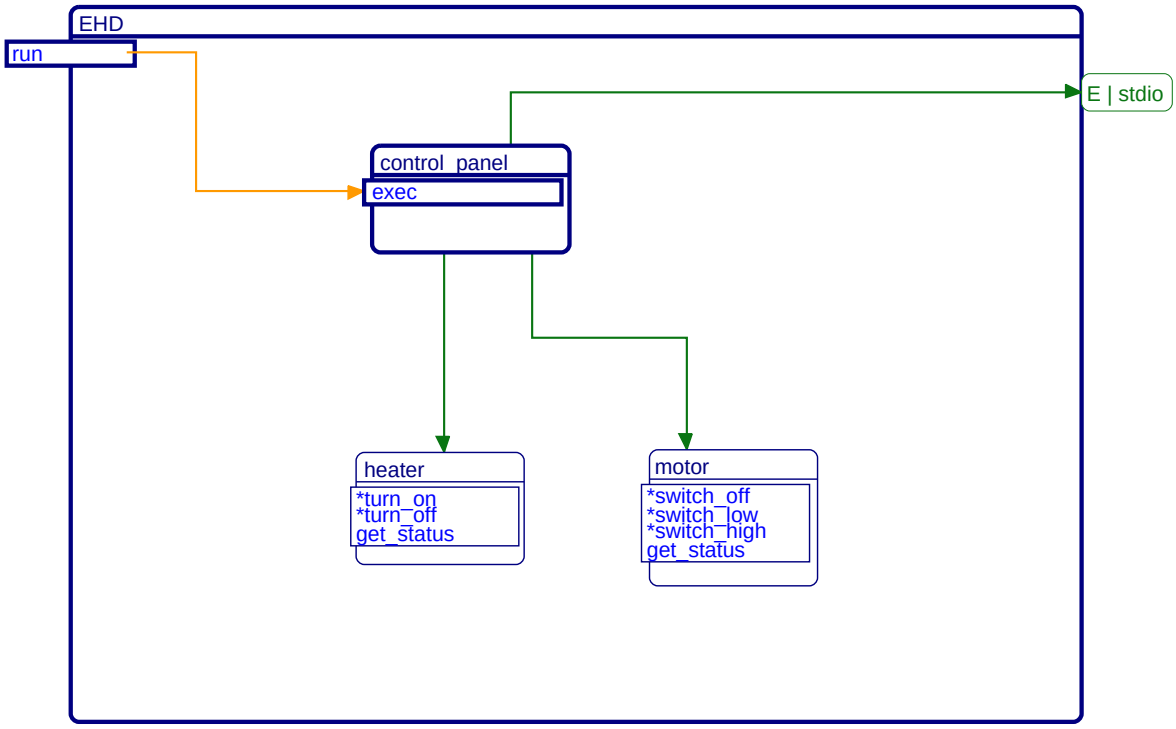


2.3.2.4. Functional Description

2.3.2.4.1. Identification of Operations (text)

At the higher level, the EHD only provides an operation to start the application.

2.3.2.4.2. Functional (oper.) Diagram



2.3.2.5. Behavioural Description

2.3.2.5.1. Identification of Local Behaviour (text)

The chosen implementation is a simple mono task application.

Thus, there no need for the EHD component to be set "active".

2.3.2.6. Justification of Design Decisions (text)

The design decisions are mainly justified by the need to provide a simple solution.

2.3.3. IMPLEMENTATION_CONSTRAINTS

The application must be coded in C language.

There is no Real Time Operating System available...

2.4. PROVIDED_INTERFACE

2.4.1. TYPES

2.4.2. CONSTANTS

2.4.3. OPERATION_SETS

2.4.4. OPERATIONS

2.4.4.1. run

2.4.4.1.1. operation spec. description (text)

The operation "run" is used to start the application.

It will be mapped to the main function during code generation.

It has no parameters (although the standard argc and argv parameters will be automatically added by the code generator)

It has no execution constraint (no trigger label in the operation properties section), as our application is single threaded.

It isn't abstract nor inherited, as it isn't a member function of a class.

We didn't mention its Worst Case Execution Time, as no schedulability analysis is required.

2.4.4.1.2. operation declaration (hood)

```
run;
```

2.4.4.1.3. operation properties (hood)

```
- trigger label :
- abstract : no
- inherited : no
```

2.4.4.1.4. real time attributes (hood)

```
WCET
```

2.4.5. EXCEPTIONS

2.5. OBJECT_CONTROL_STRUCTURE

2.6. REQUIRED_INTERFACE

```
OBJECT std;
    TYPES
        char; int;
    CONSTANTS
        NONE
    OPERATION_SETS
        NONE
    OPERATIONS
```

```
        NONE
    EXCEPTIONS
        NONE
OBJECT stdio;
    TYPES
        NONE
    CONSTANTS
        NONE
    OPERATION_SETS
        NONE
    OPERATIONS
        printf;
    EXCEPTIONS
        NONE
```

2.7. INTERNALS

2.7.1. OBJECTS

```
motor;
heater;
control_panel;
```

2.7.2. TYPES

2.7.3. CONSTANTS

2.7.4. OPERATION_SETS

2.7.5. OPERATIONS

2.7.5.1. run

2.7.5.1.1. implemented_by

```
control_panel.exec
```

2.7.6. EXCEPTIONS

2.7.7. OBJECT_CONTROL_STRUCTURE

2.7.7.1. implemented_by

```
control_panel;
```

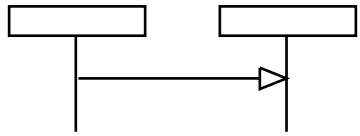
3. OBJECT motor IS

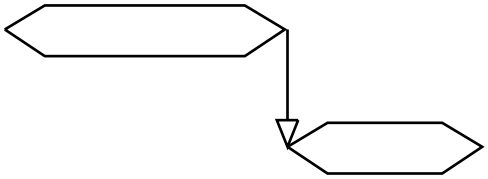
3.1. PASSIVE

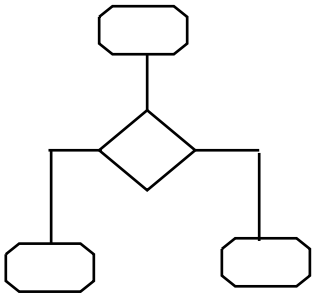
3.2. DESCRIPTION

3.2.1. PROBLEM

3.2.1.1. Sketch of the Problem







3.2.1.2. Analysis of Requirements

3.2.1.3. Local Environment

3.2.2. SOLUTION

3.2.2.1. Structural Description

3.2.2.2. Functional Description

3.2.2.3. Behavioural Description

3.3. PROVIDED_INTERFACE

3.3.1. TYPES

3.3.1.1. modes

3.3.1.1.1. type description (text)

The type "modes" describes the various possible states for the motor. (cf.motor_speed)

The C code for this enumerated type will be automatically generated from the provided list of elements.

That's why the "type definition" section is empty.

3.3.1.1.2. type properties (hood)

```
- class : no
```

3.3.1.1.3. type attributes (hood)

```
ATTRIBUTES NONE
```

3.3.1.1.4. type enumeration (hood)

```
ENUMERATION stopped, low, high
```

3.3.2. CONSTANTS

3.3.3. OPERATION_SETS

3.3.4. OPERATIONS

3.3.4.1. switch_off

3.3.4.1.1. operation spec. description (text)

This operation is called by the control panel to switch the motor off.

3.3.4.1.2. operation declaration (hood)

```
switch_off;
```

3.3.4.1.3. operation properties (hood)

```
- trigger label : STATE
- abstract : no
- inherited : no
```

3.3.4.1.4. real time attributes (hood)

```
WCET
```

3.3.4.2. switch_low

3.3.4.2.1. operation spec. description (text)

This operation is called by the control panel to switch the motor to low speed.

It is constrained by STATE, as it isn't active for all the states of the motor.

3.3.4.2.2. operation declaration (hood)

```
switch_low;
```

3.3.4.2.3. operation properties (hood)

```
- trigger label : STATE
- abstract : no
- inherited : no
```

3.3.4.2.4. real time attributes (hood)

```
WCET
```

3.3.4.3. switch_high

3.3.4.3.1. operation spec. description (text)

This operation is called by the control panel to switch the motor to high speed.

It is constrained by STATE, as it isn't active for all the states of the motor.

3.3.4.3.2. operation declaration (hood)

```
switch_high;
```

3.3.4.3.3. operation properties (hood)

```
- trigger label : STATE
- abstract : no
- inherited : no
```

3.3.4.3.4. real time attributes (hood)

```
WCET
```

3.3.4.4. get_status

3.3.4.4.1. operation spec. description (text)

This operation is called by the control panel to know about the current state of the motor.

3.3.4.4.2. operation declaration (hood)

```
get_status return motor.modes;
```

3.3.4.4.3. operation properties (hood)

```
- trigger label :
- abstract : no
- inherited : no
```

3.3.4.4.4. real time attributes (hood)

```
WCET
```

3.3.5. EXCEPTIONS

3.4. OBJECT_CONTROL_STRUCTURE

3.4.1. constrained operations

```
switch_off CONSTRAINED_BY STATE;
switch_low CONSTRAINED_BY STATE;
switch_high CONSTRAINED_BY STATE;
```

3.5. INTERNALS

3.5.1. TYPES

3.5.2. CONSTANTS

3.5.3. OPERATIONS

3.5.4. DATA

3.5.4.1. status

3.5.4.1.1. data description (text)

This data element stores the current state of the motor.

3.5.4.1.2. data declaration (c)

```
motor__modes status = stopped;
```

3.5.4.1.3. data access from pseudo_code

```
(da) motor.status IS USED BY NONE
```

3.5.4.1.4. data access from Ada code

```
(da) motor.status IS USED BY NONE
```

3.5.4.1.5. data access from C code

```
(da) motor.status IS USED BY
    (op) motor.get_status [R]
```

3.5.4.1.6. data access from C++ code

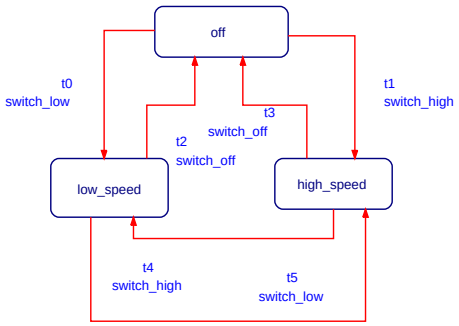
```
(da) motor.status IS USED BY NONE
```

3.5.5. OBJECT_CONTROL_STRUCTURE

3.5.5.1. obcs body description (text)

The state transition model for the motor is composed of three states (off, low_speed and high_speed), and six transitions tri
switch_off, switch_low and switch_high.

3.5.5.2. state transition diagram



3.5.5.3. STATES

3.5.5.4. off

3.5.5.4.1. entering transitions

```
t2, t3
```

3.5.5.4.2. exiting transitions

```
t0, t1
```

3.5.5.4.3. state assignment (c)

```
status = stopped;
```

3.5.5.4.4. state test (c)

```
status == stopped
```

3.5.5.5. low_speed

3.5.5.5.1. entering transitions

```
t0, t5
```

3.5.5.5.2. exiting transitions

```
t2, t4
```

3.5.5.5.3. state assignment (c)

```
status = low;
```

3.5.5.5.4. state test (c)

```
status == low
```

3.5.5.6. high_speed

3.5.5.6.1. entering transitions

```
t1, t4
```

3.5.5.6.2. exiting transitions

```
t3, t5
```

3.5.5.6.3. state assignment (c)

```
status = high;
```

3.5.5.6.4. state test (c)

```
status == high
```

3.5.5.7. TRANSITIONS

3.5.5.8. t0

3.5.5.8.1. transition event

```
switch_low
```

3.5.5.8.2. transition from

```
off
```

3.5.5.8.3. transition to

```
low_speed
```

3.5.5.9. t1

3.5.5.9.1. transition event

```
switch_high
```

3.5.5.9.2. transition from

```
off
```

3.5.5.9.3. transition to

```
high_speed
```

3.5.5.10. t2

3.5.5.10.1. transition event

```
switch_off
```

3.5.5.10.2. transition from

```
low_speed
```

3.5.5.10.3. transition to

```
off
```

3.5.5.11. t3

3.5.5.11.1. transition event

```
switch_off
```

3.5.5.11.2. transition from

```
high_speed
```

3.5.5.11.3. transition to

```
off
```

3.5.5.12. t4

3.5.5.12.1. transition event

```
switch_high
```

3.5.5.12.2. transition from

```
low_speed
```

3.5.5.12.3. transition to

high_speed

3.5.5.13. t5

3.5.5.13.1. transition event

switch_low

3.5.5.13.2. transition from

high_speed

3.5.5.13.3. transition to

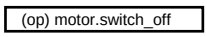
low_speed

3.5.5.14. OBCS CODE

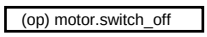
3.5.6. OPERATION_CONTROL_STRUCTURES

3.5.6.1. OPERATION switch_off IS

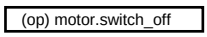
3.5.6.1.1. call tree from pseudo_code



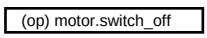
3.5.6.1.2. call tree from Ada code



3.5.6.1.3. call tree from C code

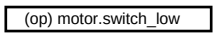


3.5.6.1.4. call tree from C++ code

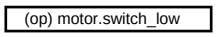


3.5.6.2. OPERATION switch_low IS

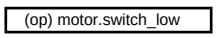
3.5.6.2.1. call tree from pseudo_code



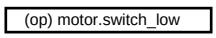
3.5.6.2.2. call tree from Ada code



3.5.6.2.3. call tree from C code

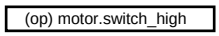


3.5.6.2.4. call tree from C++ code

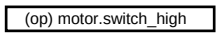


3.5.6.3. OPERATION switch_high IS

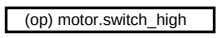
3.5.6.3.1. call tree from pseudo_code



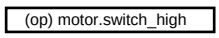
3.5.6.3.2. call tree from Ada code



3.5.6.3.3. call tree from C code



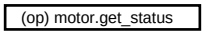
3.5.6.3.4. call tree from C++ code



3.5.6.4. OPERATION get_status IS

3.5.6.4.1. call tree from pseudo_code

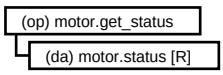
3.5.6.4.2. call tree from Ada code



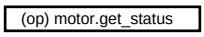
3.5.6.4.3. operation code (c)

```
{ return status; }
```

3.5.6.4.4. call tree from C code



3.5.6.4.5. call tree from C++ code



4. OBJECT heater IS

4.1. PASSIVE

4.2. DESCRIPTION

4.2.1. PROBLEM

4.2.1.1. Analysis of Requirements

4.2.1.2. Local Environment

4.2.2. SOLUTION

4.2.2.1. Structural Description

4.2.2.2. Functional Description

4.2.2.3. Behavioural Description

4.3. PROVIDED_INTERFACE

4.3.1. TYPES

4.3.1.1. modes

4.3.1.1.1. type description (text)

The type "modes" describes the various possible states for the heater.

The C code for this enumerated type will be automatically generated from the provided list of elements.

That's why the "type definition" section is empty.

4.3.1.1.2. type properties (hood)

```
- class : no
```

4.3.1.1.3. type attributes (hood)

```
ATTRIBUTES NONE
```

4.3.1.1.4. type enumeration (hood)

```
ENUMERATION on, off
```

4.3.2. CONSTANTS

4.3.3. OPERATION_SETS

4.3.4. OPERATIONS

4.3.4.1. turn_on

4.3.4.1.1. operation spec. description (text)

This operation is called by the control panel to switch the heater on.

It is constrained by STATE, as it isn't active for all the states of the heater.

4.3.4.1.2. operation declaration (hood)

```
turn_on;
```

4.3.4.1.3. operation properties (hood)

```
- trigger label : STATE
- abstract : no
- inherited : no
```

4.3.4.1.4. real time attributes (hood)

```
WCET
```

4.3.4.2. turn_off

4.3.4.2.1. operation spec. description (text)

This operation is called by the control panel to switch the heater off.

It is constrained by STATE, as it isn't active for all the states of the heater.

4.3.4.2.2. operation declaration (hood)

```
turn_off;
```

4.3.4.2.3. operation properties (hood)

```
- trigger label : STATE
- abstract : no
- inherited : no
```

4.3.4.2.4. real time attributes (hood)

```
WCET
```

4.3.4.3. get_status

4.3.4.3.1. operation spec. description (text)

This operation is called by the control panel to know about the current state of the heater.

4.3.4.3.2. operation declaration (hood)

```
get_status return heater.modes;
```

4.3.4.3.3. operation properties (hood)

```
- trigger label :
- abstract : no
- inherited : no
```

4.3.4.3.4. real time attributes (hood)

```
WCET
```

4.3.5. EXCEPTIONS

4.4. OBJECT_CONTROL_STRUCTURE

4.4.1. constrained operations

```
turn_on CONSTRAINED_BY STATE;
turn_off CONSTRAINED_BY STATE;
```

4.5. INTERNALS

4.5.1. TYPES

4.5.2. CONSTANTS

4.5.3. OPERATIONS

4.5.4. DATA

4.5.4.1. status

4.5.4.1.1. data description (text)

This data element stores the current state of the heater.

4.5.4.1.2. data declaration (c)

```
heater__modes status = off;
```

4.5.4.1.3. data access from pseudo_code

```
(da) heater.status IS USED BY NONE
```

4.5.4.1.4. data access from Ada code

```
(da) heater.status IS USED BY NONE
```

4.5.4.1.5. data access from C code

```
(da) heater.status IS USED BY NONE
```

4.5.4.1.6. data access from C++ code

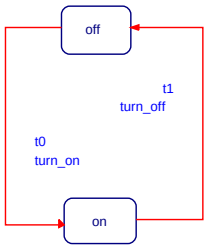
```
(da) heater.status IS USED BY NONE
```

4.5.5. OBJECT_CONTROL_STRUCTURE

4.5.5.1. obcs body description (text)

The internal state model of the heater is just composed of two states (on and off) and two transitions triggered by the operat

4.5.5.2. state transition diagram



4.5.5.3. STATES

4.5.5.4. off

4.5.5.4.1. entering transitions

```
t1
```

4.5.5.4.2. exiting transitions

```
t0
```

4.5.5.4.3. state description (text)

tyugyuzsysq

jhjhqh

4.5.5.4.4. state assignment (c)

```
status = off;
```

4.5.5.4.5. state test (c)

```
status == off
```

4.5.5.5. on

4.5.5.5.1. entering transitions

```
t0
```

4.5.5.5.2. exiting transitions

```
t1
```

4.5.5.5.3. state assignment (c)

```
status = on;
```

4.5.5.5.4. state test (c)

```
status == on
```

4.5.5.6. TRANSITIONS

4.5.5.7. t0

4.5.5.7.1. transition event

```
turn_on
```

4.5.5.7.2. transition from

```
off
```

4.5.5.7.3. transition to

```
on
```

4.5.5.7.4. trans description (text)

JIUJN

4.5.5.8. t1

4.5.5.8.1. transition event

```
turn_off
```

4.5.5.8.2. transition from

```
on
```

4.5.5.8.3. transition to

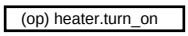
```
off
```

4.5.5.9. OBCS CODE

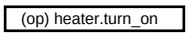
4.5.6. OPERATION_CONTROL_STRUCTURES

4.5.6.1. OPERATION turn_on IS

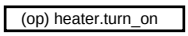
4.5.6.1.1. call tree from pseudo_code



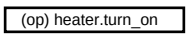
4.5.6.1.2. call tree from Ada code



4.5.6.1.3. call tree from C code



4.5.6.1.4. call tree from C++ code



4.5.6.2. OPERATION turn_off IS

4.5.6.2.1. call tree from pseudo_code

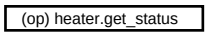
4.5.6.2.2. call tree from Ada code

4.5.6.2.3. call tree from C code

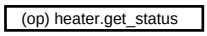
4.5.6.2.4. call tree from C++ code

4.5.6.3. OPERATION get_status IS

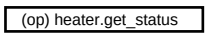
4.5.6.3.1. call tree from pseudo_code



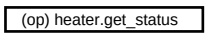
4.5.6.3.2. call tree from Ada code



4.5.6.3.3. call tree from C code



4.5.6.3.4. call tree from C++ code



5. OBJECT control_panel IS

5.1. PASSIVE

5.2. DESCRIPTION

5.2.1. PROBLEM

5.2.1.1. Statement of the Problem (text)

The control panel is in charge of handling the user interface for the EHD simulator, and also manages the motor and the hea

5.2.1.2. Analysis of Requirements

5.2.1.3. Local Environment

5.2.2. SOLUTION

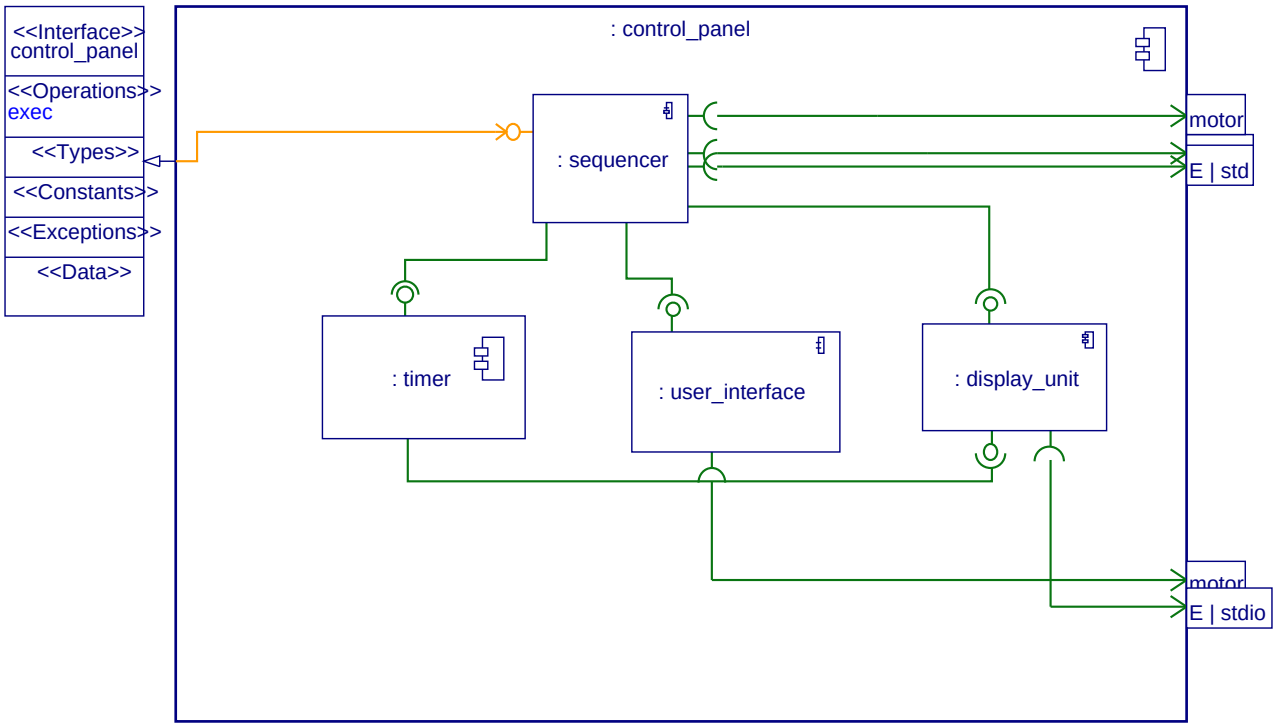
5.2.2.1. Identification of Child Modules (text)

The control panel component is composed of:

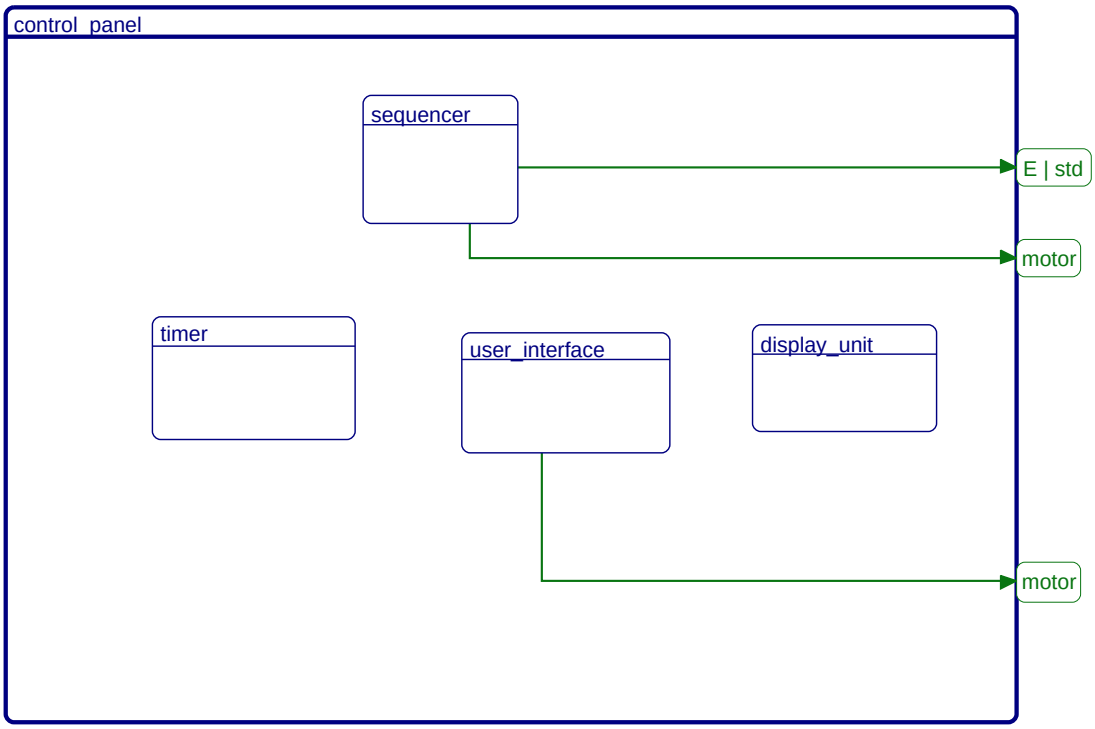
- a sequencer subcomponent.
- a user interface subcomponent to get instructions from the user.
- a display unit subcomponent to show various parameters of the system.
- a timer subcomponent to be used by the sequencer.

5.2.2.2. Structural Description

5.2.2.2.1. UML Structure Diagram

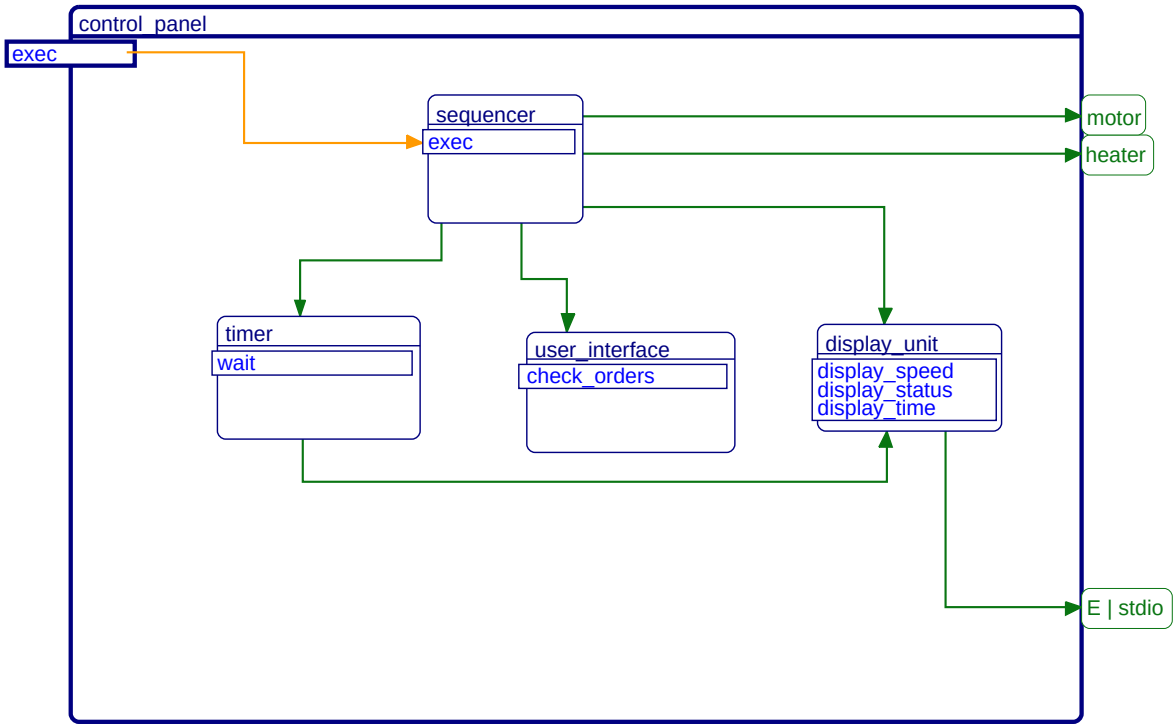


5.2.2.2.2. Structural (types) Diagram



5.2.2.3. Functional Description

5.2.2.3.1. Functional (oper.) Diagram



5.2.2.4. Behavioural Description

5.3. PROVIDED_INTERFACE

5.3.1. TYPES

5.3.2. CONSTANTS

5.3.3. OPERATION_SETS

5.3.4. OPERATIONS

5.3.4.1. exec

5.3.4.1.1. operation declaration (hood)

```
exec;
```

5.3.4.1.2. operation properties (hood)

```
- trigger label :
- abstract : no
- inherited : no
```

5.3.4.1.3. real time attributes (hood)

```
WCET
```

5.3.5. EXCEPTIONS

5.4. OBJECT_CONTROL_STRUCTURE

5.5. REQUIRED_INTERFACE

```
OBJECT heater;
    TYPES
        NONE
    CONSTANTS
        NONE
    OPERATION_SETS
        NONE
    OPERATIONS
        turn_off; turn_on;
    EXCEPTIONS
        NONE
OBJECT motor;
    TYPES
        modes;
    CONSTANTS
        NONE
    OPERATION_SETS
        NONE
    OPERATIONS
        switch_off; switch_low; switch_high;
    EXCEPTIONS
        NONE
OBJECT std;
    TYPES
        char; int;
    CONSTANTS
        NONE
    OPERATION_SETS
        NONE
    OPERATIONS
        NONE
    EXCEPTIONS
        NONE
OBJECT stdio;
    TYPES
        NONE
```

```
CONSTANTS
    NONE
OPERATION_SETS
    NONE
OPERATIONS
    printf;
EXCEPTIONS
    NONE
```

5.6. INTERNALS

5.6.1. OBJECTS

```
display_unit;
user_interface;
sequencer;
timer;
```

5.6.2. TYPES

5.6.3. CONSTANTS

5.6.4. OPERATION_SETS

5.6.5. OPERATIONS

5.6.5.1. exec

5.6.5.1.1. implemented_by

```
sequencer.exec
```

5.6.6. EXCEPTIONS

5.6.7. OBJECT_CONTROL_STRUCTURE

5.6.7.1. implemented_by

```
sequencer;
```

6. OBJECT display_unit IS

6.1. PASSIVE

6.2. DESCRIPTION

6.2.1. PROBLEM

6.2.1.1. Analysis of Requirements

6.2.1.2. Local Environment

6.2.2. SOLUTION

6.2.2.1. Structural Description

6.2.2.2. Functional Description

6.2.2.3. Behavioural Description

6.3. PROVIDED_INTERFACE

6.3.1. TYPES

6.3.2. CONSTANTS

6.3.3. OPERATION_SETS

6.3.4. OPERATIONS

6.3.4.1. display_speed

6.3.4.1.1. operation spec. description (text)

The operation "display_speed" shows the current status of the motor on the control panel.(cf.display_speed)

6.3.4.1.2. operation declaration (hood)

```
display_speed(text : in char*);
```

6.3.4.1.3. operation properties (hood)

```
- trigger label :
- abstract : no
- inherited : no
```

6.3.4.1.4. real time attributes (hood)

```
WCET
```

6.3.4.2. display_status

6.3.4.2.1. operation spec. description (text)

The operation "display_status" shows the current status of the heater on the control panel.(cf.display_status)

6.3.4.2.2. operation declaration (hood)

```
display_status(text : in char*);
```

6.3.4.2.3. operation properties (hood)

```
- trigger label :
- abstract : no
- inherited : no
```

6.3.4.2.4. real time attributes (hood)

```
WCET
```

6.3.4.3. display_time

6.3.4.3.1. operation spec. description (text)

The operation "display_time" shows the ellapsed time of the timer on the control panel.(cf.display_time)

6.3.4.3.2. operation declaration (hood)

```
display_time(value : in int);
```

6.3.4.3.3. operation properties (hood)

```
- trigger label :
- abstract : no
```

```
- inherited : no
```

6.3.4.3.4. real time attributes (hood)

```
WCET
```

6.3.5. EXCEPTIONS

6.4. OBJECT_CONTROL_STRUCTURE

6.5. REQUIRED_INTERFACE

```
OBJECT std;
    TYPES
        char; int;
    CONSTANTS
        NONE
    OPERATION_SETS
        NONE
    OPERATIONS
        NONE
    EXCEPTIONS
        NONE
OBJECT stdio;
    TYPES
        NONE
    CONSTANTS
        NONE
    OPERATION_SETS
        NONE
    OPERATIONS
        printf;
    EXCEPTIONS
        NONE
```

6.6. INTERNALS

6.6.1. TYPES

6.6.2. CONSTANTS

6.6.3. OPERATIONS

6.6.4. DATA

6.6.5. OBJECT_CONTROL_STRUCTURE

6.6.5.1. STATES

6.6.5.2. TRANSITIONS

6.6.5.3. OBCS CODE

6.6.6. OPERATION_CONTROL_STRUCTURES

6.6.6.1. OPERATION display_speed IS

6.6.6.1.1. used operations

```
stdio.printf
```

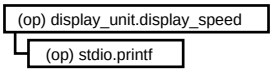
6.6.6.1.2. call tree from pseudo_code

6.6.6.1.3. call tree from Ada code

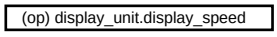
6.6.6.1.4. operation code (c)

```
{ printf("%s\n",text); }
```

6.6.6.1.5. call tree from C code



6.6.6.1.6. call tree from C++ code

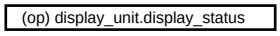


6.6.6.2. OPERATION display_status IS

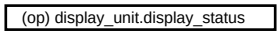
6.6.6.2.1. used operations

```
stdio.printf
```

6.6.6.2.2. call tree from pseudo_code



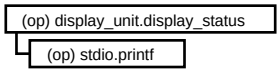
6.6.6.2.3. call tree from Ada code



6.6.6.2.4. operation code (c)

```
{ printf("%s\n",text); }
```

6.6.6.2.5. call tree from C code



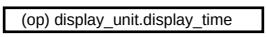
6.6.6.2.6. call tree from C++ code

6.6.6.3. OPERATION display_time IS

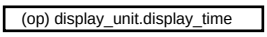
6.6.6.3.1. used operations

```
stdio.printf
```

6.6.6.3.2. call tree from pseudo_code



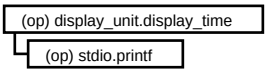
6.6.6.3.3. call tree from Ada code



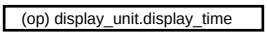
6.6.6.3.4. operation code (c)

```
{ printf("%d\n",value); }
```

6.6.6.3.5. call tree from C code



6.6.6.3.6. call tree from C++ code



7. OBJECT user_interface IS

7.1. PASSIVE

7.2. DESCRIPTION

7.2.1. PROBLEM

7.2.1.1. Analysis of Requirements

7.2.1.2. Local Environment

7.2.2. SOLUTION

7.2.2.1. Structural Description

7.2.2.2. Functional Description

7.2.2.3. Behavioural Description

7.3. PROVIDED_INTERFACE

7.3.1. TYPES

7.3.2. CONSTANTS

7.3.3. OPERATION_SETS

7.3.4. OPERATIONS

7.3.4.1. check_orders

7.3.4.1.1. operation declaration (hood)

```
check_orders return motor.modes;
```

7.3.4.1.2. operation properties (hood)

```
- trigger label :
- abstract : no
- inherited : no
```

7.3.4.1.3. real time attributes (hood)

```
WCET
```

7.3.5. EXCEPTIONS

7.4. OBJECT_CONTROL_STRUCTURE

7.5. REQUIRED_INTERFACE

```
OBJECT motor;
    TYPES
        modes;
    CONSTANTS
        NONE
    OPERATION_SETS
        NONE
    OPERATIONS
```

```
        NONE
    EXCEPTIONS
        NONE
```

7.6. INTERNALS

7.6.1. TYPES

7.6.2. CONSTANTS

7.6.3. OPERATIONS

7.6.4. DATA

7.6.5. OBJECT_CONTROL_STRUCTURE

7.6.5.1. STATES

7.6.5.2. TRANSITIONS

7.6.5.3. OBCS CODE

7.6.6. OPERATION_CONTROL_STRUCTURES

7.6.6.1. OPERATION check_orders IS

7.6.6.1.1. call tree from pseudo_code

7.6.6.1.2. call tree from Ada code

7.6.6.1.3. operation code (c)

```
{
  motor__modes order = stopped;
  /* user dialog to get next order */
  return order;
}
```

7.6.6.1.4. call tree from C code

7.6.6.1.5. call tree from C++ code

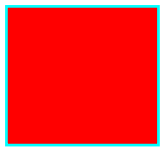
8. OBJECT sequencer IS

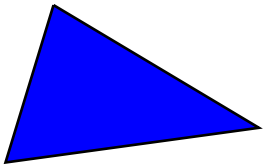
8.1. PASSIVE

8.2. DESCRIPTION

8.2.1. PROBLEM

8.2.1.1. Sketch of the Problem





texykjbhjsbvh
vqjhxb

texykjbhjsbvh
vqjhxb

8.2.1.2. Analysis of Requirements

8.2.1.3. Local Environment

8.2.2. SOLUTION

8.2.2.1. Structural Description

8.2.2.2. Functional Description

8.2.2.3. Behavioural Description

8.3. PROVIDED_INTERFACE

8.3.1. TYPES

8.3.2. CONSTANTS

8.3.3. OPERATION_SETS

8.3.4. OPERATIONS

8.3.4.1. exec

8.3.4.1.1. operation declaration (hood)

```
exec;
```

8.3.4.1.2. operation properties (hood)

```
- trigger label :
- abstract : no
- inherited : no
```

8.3.4.1.3. real time attributes (hood)

```
WCET
```

8.3.5. EXCEPTIONS

8.4. OBJECT_CONTROL_STRUCTURE

8.5. REQUIRED_INTERFACE

```
OBJECT display_unit;
    TYPES
        NONE
    CONSTANTS
        NONE
    OPERATION_SETS
        NONE
    OPERATIONS
        display_speed; display_status;
    EXCEPTIONS
        NONE
OBJECT heater;
    TYPES
        NONE
    CONSTANTS
        NONE
    OPERATION_SETS
        NONE
    OPERATIONS
        turn_off; turn_on;
    EXCEPTIONS
        NONE
OBJECT motor;
    TYPES
        modes;
    CONSTANTS
        NONE
    OPERATION_SETS
        NONE
    OPERATIONS
        switch_off; switch_low; switch_high;
    EXCEPTIONS
        NONE
OBJECT std;
    TYPES
        int;
    CONSTANTS
        NONE
    OPERATION_SETS
        NONE
    OPERATIONS
        NONE
    EXCEPTIONS
        NONE
OBJECT timer;
    TYPES
        NONE
```

```
    CONSTANTS
        NONE
    OPERATION_SETS
        NONE
    OPERATIONS
        wait;
    EXCEPTIONS
        NONE
OBJECT user_interface;
    TYPES
        NONE
    CONSTANTS
        NONE
    OPERATION_SETS
        NONE
    OPERATIONS
        check_orders;
    EXCEPTIONS
        NONE
```

8.6. INTERNALS

8.6.1. TYPES

8.6.2. CONSTANTS

8.6.3. OPERATIONS

8.6.3.1. cycle_heater

8.6.3.1.1. operation declaration (hood)

```
cycle_heater;
```

8.6.3.1.2. operation properties (hood)

```
- trigger label :
- abstract : no
- inherited : no
```

8.6.3.1.3. real time attributes (hood)

```
WCET
```

8.6.4. DATA

8.6.5. OBJECT_CONTROL_STRUCTURE

8.6.5.1. STATES

8.6.5.2. TRANSITIONS

8.6.5.3. OBCS CODE

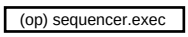
8.6.6. OPERATION_CONTROL_STRUCTURES

8.6.6.1. OPERATION exec IS

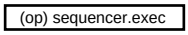
8.6.6.1.1. used operations

```
user_interface.check_orders
motor.switch_off
display_unit.display_speed
display_unit.display_status
heater.turn_off
motor.switch_low
sequencer.cycle_heater
motor.switch_high
```

8.6.6.1.2. call tree from pseudo_code



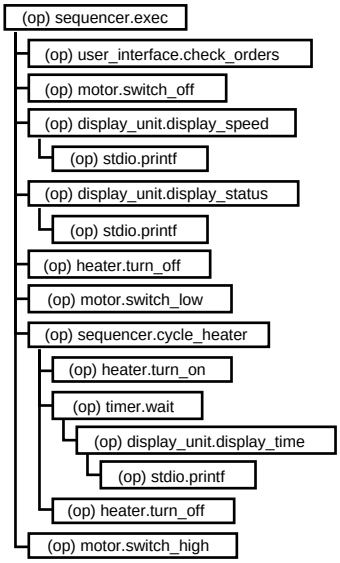
8.6.6.1.3. call tree from Ada code



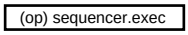
8.6.6.1.4. operation code (c)

```
{
  int running = 1;
  motor__modes current_order = stopped;
  while (running)
  {
    current_order = user_interface__check_orders();
    switch (current_order)
    {
      case stopped :
      {
        motor__switch_off();
        display_unit__display_speed("off");
        display_unit__display_status("off");
        heater__turn_off();
      }
      case low :
      {
        motor__switch_low();
        display_unit__display_speed("low");
        display_unit__display_status("on");
        cycle_heater();
      }
      case high:
      {
        motor__switch_high();
        display_unit__display_speed("high");
        display_unit__display_status("on");
        cycle_heater();
      }
    }
  }
}
```

8.6.6.1.5. call tree from C code



8.6.6.1.6. call tree from C++ code



8.6.6.2. OPERATION cycle_heater IS

8.6.6.2.1. used operations

```
heater.turn_on
timer.wait
heater.turn_off
```

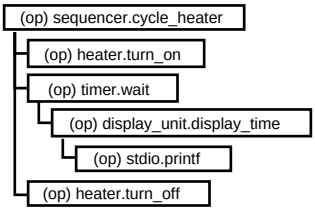
8.6.6.2.2. call tree from pseudo_code

8.6.6.2.3. call tree from Ada code

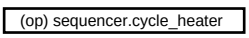
8.6.6.2.4. operation code (c)

```
{
  heater__turn_on();
  timer__wait(30);
  heater__turn_off();
}
```

8.6.6.2.5. call tree from C code



8.6.6.2.6. call tree from C++ code



9. OBJECT timer IS

9.1. PASSIVE

9.2. DESCRIPTION

9.2.1. PROBLEM

9.2.1.1. Analysis of Requirements

9.2.1.2. Local Environment

9.2.2. SOLUTION

9.2.2.1. Structural Description

9.2.2.2. Functional Description

9.2.2.3. Behavioural Description

9.3. PROVIDED_INTERFACE

9.3.1. TYPES

9.3.2. CONSTANTS

9.3.3. OPERATION_SETS

9.3.4. OPERATIONS

9.3.4.1. wait

9.3.4.1.1. operation declaration (hood)

```
wait(delay : in int);
```

9.3.4.1.2. operation properties (hood)

```
- trigger label :
- abstract : no
- inherited : no
```

9.3.4.1.3. real time attributes (hood)

```
WCET
```

9.3.5. EXCEPTIONS

9.4. OBJECT_CONTROL_STRUCTURE

9.5. REQUIRED_INTERFACE

```
OBJECT display_unit;
    TYPES
        NONE
    CONSTANTS
        NONE
    OPERATION_SETS
        NONE
    OPERATIONS
        display_time;
    EXCEPTIONS
        NONE
OBJECT std;
    TYPES
        int;
    CONSTANTS
```

```
    NONE
  OPERATION_SETS
    NONE
  OPERATIONS
    NONE
  EXCEPTIONS
    NONE
```

9.6. INTERNALS

9.6.1. TYPES

9.6.2. CONSTANTS

9.6.3. OPERATIONS

9.6.4. DATA

9.6.5. OBJECT_CONTROL_STRUCTURE

9.6.5.1. STATES

9.6.5.2. TRANSITIONS

9.6.5.3. OBCS CODE

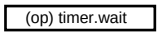
9.6.6. OPERATION_CONTROL_STRUCTURES

9.6.6.1. OPERATION wait IS

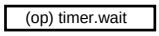
9.6.6.1.1. used operations

```
display_unit.display_time
```

9.6.6.1.2. call tree from pseudo_code



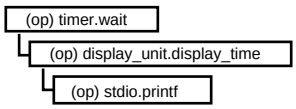
9.6.6.1.3. call tree from Ada code



9.6.6.1.4. operation code (c)

```
{
  while (delay)
  {
    delay--;
    display_unit__display_time(delay);
  }
}
```

9.6.6.1.5. call tree from C code



9.6.6.1.6. call tree from C++ code