COMMERCIAL IN CONFIDENCE

ROCKET EXAMPLE

HOOD Toolset Test Data

HOOD/ROCKET/EXAMPLE

Issue 1.2

Dated 20th January 1999

N.P.Wall et. al.

Review Approval CM Approval

Applies-to: HOOD-V5.1

Distribution: None

Abstract: Design is used as test data for HOOD Toolset Test Specification.

ROCKET EXAMPLE HOOD Toolset Test Data	Reference: ROCKET/EG
	Issue: 1.2
	Page: 1

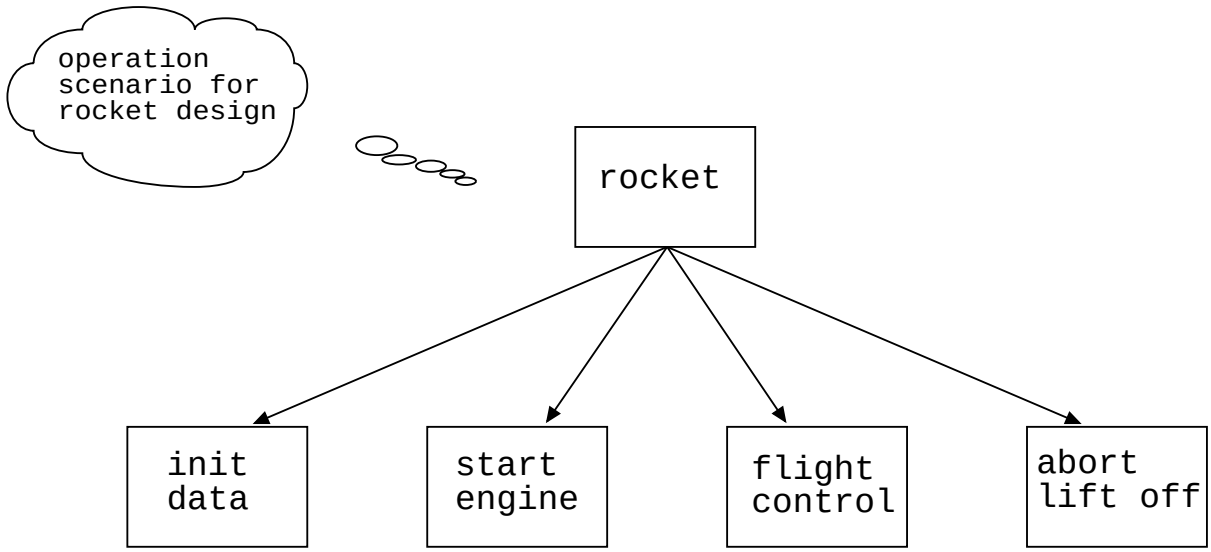
1. INTRODUCTION

This is the Introduction unit of the design rocket.

The design used is an extremely trivial one used to emulate a rocket system. The design exercises most of the features of the HOOD method.

The starting point of the system is the procedure start of the rocket root object which is the start_engine operation of the pilot. The system will then run ad infinitum/nauseam until it recieves the abort_lift_off call.

This operation sequence can be depicted diagrammatically as follows:



This is the design level DFD
of the design "rocket" and is called
"new_dfd"

2. SYSTEM DESIGN

Numbered Object Hierarchy

```
1 ........ rocket  (ACTIVE)
1.1 ........ clock  (ACTIVE)
1.2 ........ display
1.2.1 ........ display_manager
1.2.2 ........ screen
1.3 ........ engine
1.4 ........ fuel
1.5 ........ navigation_system  (ACTIVE)
1.6 ........ pilot  (ACTIVE)
2 ........ standard_io
```

2.1. (1H) Object rocket

2.1.1. (1H1) Problem Definition

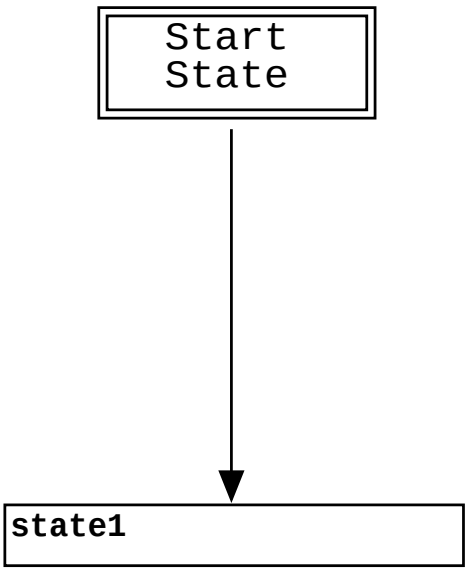
2.1.1.1. (1H1.1) Statement of the Problem

This is the Problem Statement unit of the object rocket in the design rocket.

The rocket system will take several user inputs and produce several outputs.

This is the

ROCKET STD



2.1.1.2. (1H1.2) Analysis Of Requirements

This is the Requirements Analysis unit of the object rocket in the design rocket.

The pilot will input the required altitude, initial fuel, required thrust, and payload mass. The navigation system will then calculate the current acceleration, velocity, and altitude upon receipt of a time update from the clock. Using the required thrust as a guide the engine will calculate the fuel used which it will pass to the fuel control system which will then calculate the remaining fuel.

2.1.2. (1H2) Informal Solution Strategy

This is the informal solution for the object rocket.

The pilot will input the required altitude, initial fuel, required thrust, and payload mass. The navigation system will then calculate the current acceleration, velocity, and altitude upon receipt of a time update from the clock. Using the required thrust as a guide the engine will calculate the fuel used which it will pass to the fuel control system which will then calculate the remaining fuel.

2.1.3. (1H3) Formalisation of the Strategy

2.1.3.1. (1H3.1) Identification of Objects

The potential set of objects are as follows:

rocket
pilot
navigation system
clock
engine
fuel control

2.1.3.2. (1H3.2) Identification of Operations

The potential operations are :

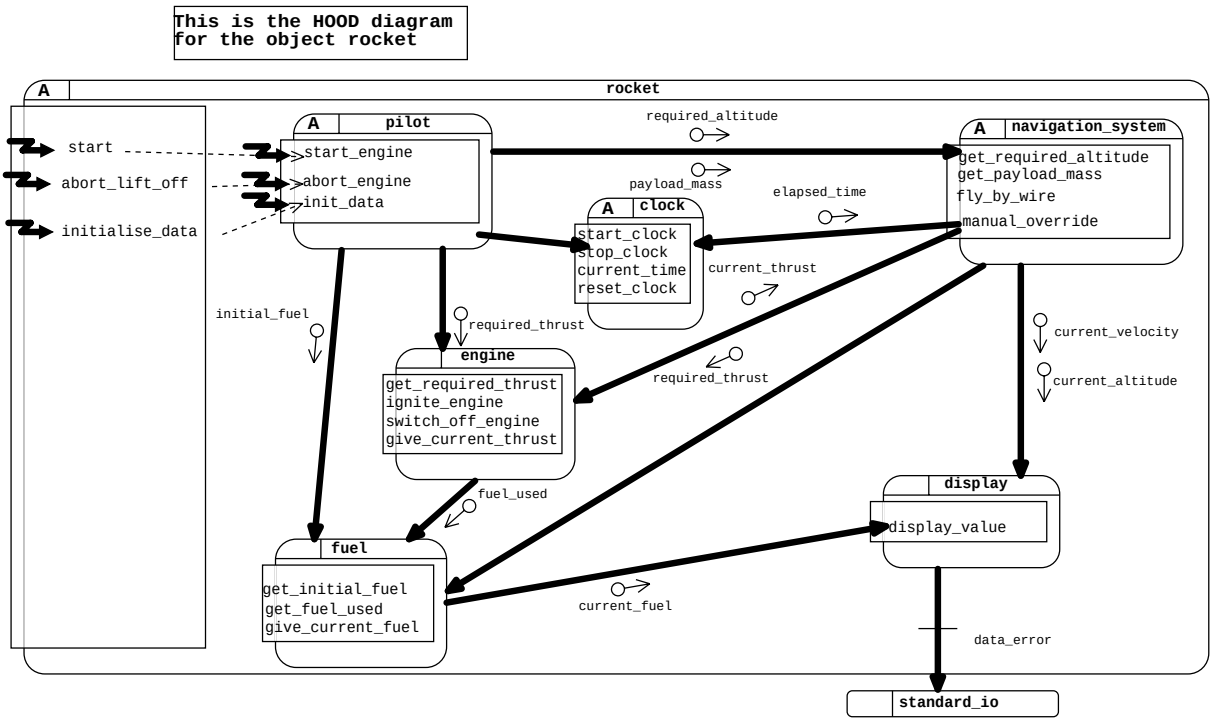
start
start engine
get required altitude
get initial fuel
get required thrust
ignite engine
get payload mass
update time
display current acceleration
display current velocity,
display current altitude
show current fuel

2.1.3.3. (1H3.3) Grouping Operations and Objects

```
OBJECT  OPERATION                PARAMETERS       USER
------  ---------                ----------       ----

rocket
        start

pilot
        start engine

navigation system
        get required altitude    altitude         pilot
        get payload mass         payload_mass     pilot

clock
        update time              elapsed_time     navigation_system
```

```
engine
        get required thrust        thrust           navigation_system
        ignite engine                               pilot

fuel
        show current fuel          remaining_fuel   display
        get initial fuel           fuel_mass        pilot
        get_fuel_used              fuel_used        engine
```

2.1.3.4. (1H3.4) Graphical Description



2.1.3.5. (1H3.5) Justification of Design Decisions

This is the Justification of Design Decisions unit of the object "rocket" in the design "rocket".

The main factor in the design is to assist in the testing and demonstration of the HOOD toolset. This is an example design only.

2.1.4. (1H4) Formalisation of the Solution

```
OBJECT  rocket IS ACTIVE

   PRAGMA EXCEPTION LOG (NO)
   PRAGMA USE CLAUSES (YES)
```

DESCRIPTION
The object rocket simulates a satellite launcher. The user can start the engine, abort the engine and display the outputs of various dataflows. The top level object rocket contains the objects pilot, fuel, engine, navigation_system, clock, and display.

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
NONE

REQUIREMENT_REFERENCES
XR01 (xref_requirement_1) ;
XR02 (xref_requirement_2) ;

PROVIDED_INTERFACE
CONSTANTS
NONE
TYPES
NONE
DATA
NONE
DECLARATIONS
NONE
OPERATIONS
start;
abort_lift_off;
initialise_data (
req_thrust : IN Float;
req_alt : IN Float;
cargo_mass : IN Float;
init_fuel : IN Float);
OPERATION_SETS
NONE
EXCEPTIONS
fatal_error;

REQUIRED_INTERFACE
OBJECTS
standard_io;
ENVIRONMENT_OBJECTS
NONE
CLASS_OBJECTS
NONE
CONSTANTS
NONE
TYPES
standard_io.file_type;
DATA
NONE
OPERATIONS
standard_io.open;
standard_io.put_line;
standard_io.close;
EXCEPTIONS
standard_io.data_error;

DATAFLOWS

NONE

OBJECT_CONTROL_STRUCTURE
PRAGMA CODE_BODY (EMBEDDED)
PRAGMA CODE_SPEC (HIDDEN)
DESCRIPTION
The rocket system is started by an asynchronous interrupt
to the operation start. It can be halted at any time by
an asynchronous interrupt to the operation abort_engine.
Whilst running it continuously displays several items of data.
CONSTRAINED_OPERATIONS
start;
abort_lift_off;
initialise_data;
REQUIREMENT_REFERENCES
NONE
USED_OPERATIONS
NONE
EXCEPTIONS
NONE
IMPLEMENTED_BY
pilot;

PRIVATE
CONSTANTS
NONE
TYPES
NONE
DATA
NONE

INTERNALS
OBJECTS
pilot;
fuel;
engine;
navigation_system;
clock;
display;
ENVIRONMENT_OBJECTS
NONE
CLASS_OBJECTS
NONE
DECLARATIONS
NONE
OPERATIONS
NONE
EXCEPTIONS
NONE

OPERATION_CONTROL_STRUCTURE
start IS
PRAGMA CODE_BODY (SEPARATE)
PRAGMA CODE_IMPL (CALL)
DESCRIPTION
The start operation of the object rocket is directly

```
   implemented by the start_engine operation of the object
   pilot.
REQUIREMENT_REFERENCES
   NONE
USED_OPERATIONS
   NONE
EXCEPTIONS
   NONE
IMPLEMENTED_BY
   pilot.start_engine;
END_OPERATION start;

 abort_lift_off IS
   PRAGMA CODE_BODY  (EMBEDDED)
   PRAGMA CODE_IMPL  (CALL)
DESCRIPTION
   The operation abort_lift_off of the root object rocket is
   directly implemented by the operation abort_engine of the
   object pilot.
REQUIREMENT_REFERENCES
   NONE
USED_OPERATIONS
   NONE
EXCEPTIONS
   NONE
IMPLEMENTED_BY
   pilot.abort_engine;
END_OPERATION abort_lift_off;

 initialise_data (
        req_thrust : IN Float;
        req_alt : IN Float;
        cargo_mass : IN Float;
        init_fuel : IN Float ) IS
   PRAGMA CODE_BODY  (EMBEDDED)
   PRAGMA CODE_IMPL  (CALL)
DESCRIPTION
   NONE
REQUIREMENT_REFERENCES
   NONE
USED_OPERATIONS
   NONE
EXCEPTIONS
   NONE
IMPLEMENTED_BY
   pilot.init_data;
END_OPERATION initialise_data;


END_OBJECT rocket
```

2.2. (1.1H) Object clock

2.2.1. (1.1H1) Problem Definition

2.2.1.1. (1.1H1.1) Statement of the Problem

The object clock should emit a time update every 0.2 seconds, the output being the time elapsed since the start of the system.

2.2.1.2. (1.1H1.2) Analysis Of Requirements

The object clock should emit a time update every 0.2 seconds, the output being the time elapsed since the start of the system.

2.2.2. (1.1H2) Informal Solution Strategy

The clock will be a 'leaf' object (i.e. subject to no further decomposition). Its control structure will loop untill stopped by the pilot (stop_clock).

2.2.3. (1.1H3) Formalisation of the Strategy

2.2.3.1. (1.1H3.1) Identification of Objects

-- NOT APPLICABLE - LEAF OBJECT--

2.2.3.2. (1.1H3.2) Identification of Operations

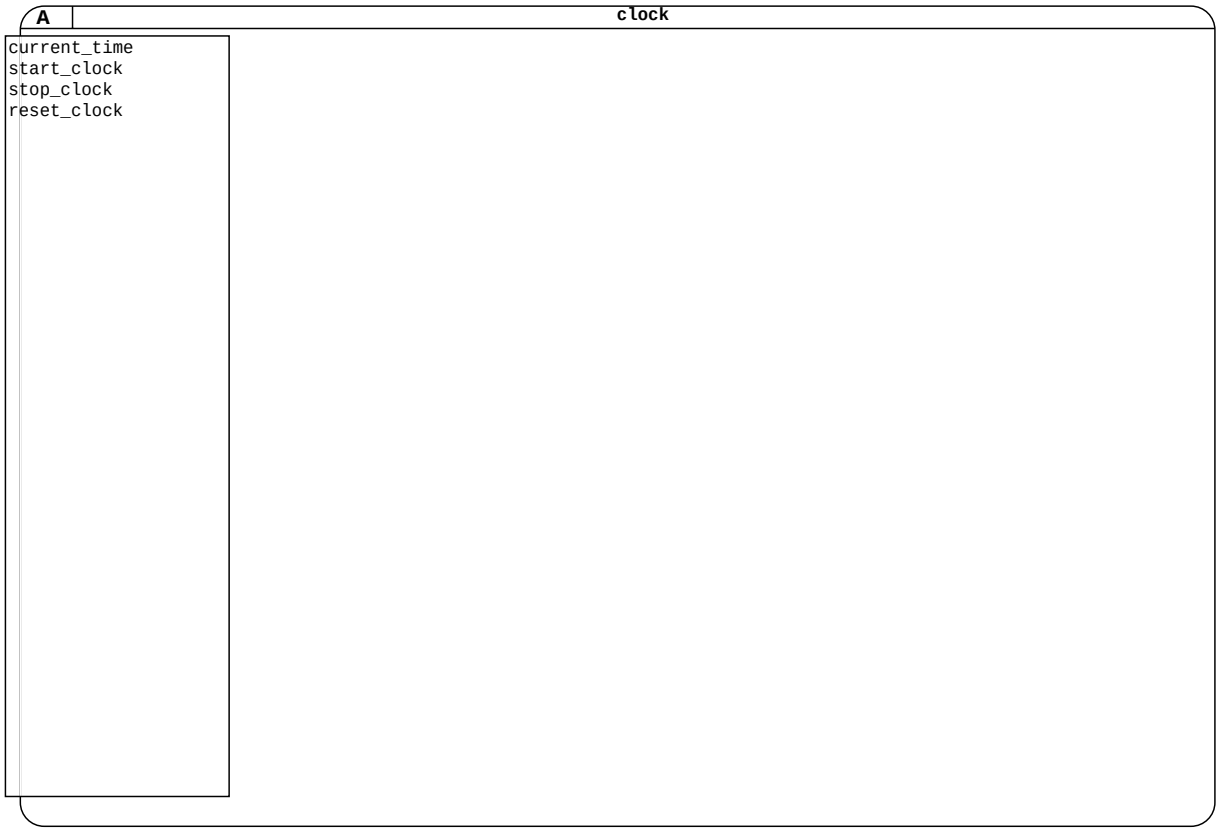
This object will provide the external operations :

start_clock;
update_time; and
stop_clock.

2.2.3.3. (1.1H3.3) Grouping Operations and Objects

-- ENTFAELLT - BLAETTERKNOTE --

2.2.3.4. (1.1H3.4) Graphical Description



2.2.3.5. (1.1H3.5) Justification of Design Decisions

(Not Available)

2.2.4. (1.1H4) Formalisation of the Solution

OBJECT clock IS ACTIVE

PRAGMA EXCEPTION LOG (NO)
PRAGMA USE CLAUSES (YES)

DESCRIPTION
This object provides a mechanism for starting, obtaining and stopping (and hence resetting) the elapsed time of the system. The clock starts running when the pilot object calls start_clock and updates the time every 0.2 second until it receives a call to stop_clock.

IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
NONE

REQUIREMENT_REFERENCES

```
   NONE

PROVIDED_INTERFACE
 CONSTANTS
  NONE
 TYPES
  SUBTYPE second IS Float;
 DATA
  NONE
 DECLARATIONS
  NONE
 OPERATIONS
    current_time (
       current_time : OUT second );
    start_clock;
    stop_clock;
    reset_clock;
 OPERATION_SETS
  NONE
 EXCEPTIONS
  NONE

REQUIRED_INTERFACE
 OBJECTS
  NONE
 ENVIRONMENT_OBJECTS
  NONE
 CLASS_OBJECTS
  NONE
 CONSTANTS
  NONE
 TYPES
  NONE
 DATA
  NONE
 OPERATIONS
  NONE
 EXCEPTIONS
  NONE

DATAFLOWS
   NONE

OBJECT_CONTROL_STRUCTURE
   PRAGMA CODE_BODY (EMBEDDED)
   PRAGMA CODE_SPEC (HIDDEN)
DESCRIPTION
   While the clock is running loop updating the elapsed time
   by 0.2 of a second.
CONSTRAINED_OPERATIONS
   NONE
REQUIREMENT_REFERENCES
   NONE
USED_OPERATIONS
   NONE
EXCEPTIONS
```

```
   NONE
DECLARATIONS
   NONE
PSEUDO_CODE
   NONE
CODE
   LOOP
     SELECT
       WHEN clock_running = FALSE  =>
         ACCEPT start_clock;
              -- sets clock_running := TRUE;
              LOOP
                update_time();
              END LOOP;
     OR
       WHEN clock_running =>
         ACCEPT current_time;
     OR
       WHEN clock_running =>
         ACCEPT stop_clock;
         -- sets clock_running := FALSE;
         -- and elapsed_time := 0.0;
     OR
       WHEN clock_running =>
         ACCEPT reset_clock; -- sets elapsed_time to 0.0
     END SELECT;
   END LOOP;

EXCEPTION_HANDLER
   NONE

PRIVATE
 CONSTANTS
   NONE
 TYPES
   NONE
 DATA
   NONE

INTERNALS
 OBJECTS
   NONE
 ENVIRONMENT_OBJECTS
   NONE
 CLASS_OBJECTS
   NONE
 DECLARATIONS
   elapsed_time : second;
   clock_running : BOOLEAN := FALSE;
 OPERATIONS
    update_time;
 EXCEPTIONS
   NONE

OPERATION_CONTROL_STRUCTURE
 current_time (
```

```
        current_time : OUT second ) IS
   PRAGMA CODE_BODY  (SEPARATE)
   PRAGMA CODE_IMPL  (CALL)
DESCRIPTION
   Returns the currently elapsed time
REQUIREMENT_REFERENCES
   NONE
USED_OPERATIONS
   NONE
EXCEPTIONS
   NONE
DECLARATIONS
   NONE
PSEUDO_CODE
   NONE
CODE
   CONTROL.current_time;
   current_time := elapsed_time;
EXCEPTION_HANDLER
   NONE
END_OPERATION current_time;

 start_clock IS
   PRAGMA CODE_BODY  (EMBEDDED)
   PRAGMA CODE_IMPL  (CALL)
DESCRIPTION
   This operation starts the clock by setting the boolean
   clock_running to true.
REQUIREMENT_REFERENCES
   NONE
USED_OPERATIONS
   NONE
EXCEPTIONS
   NONE
DECLARATIONS
   NONE
PSEUDO_CODE
   NONE
CODE
   CONTROL.start_clock;
   clock_running := true;
EXCEPTION_HANDLER
   NONE
END_OPERATION start_clock;

 stop_clock IS
   PRAGMA CODE_BODY  (EMBEDDED)
   PRAGMA CODE_IMPL  (CALL)
DESCRIPTION
   This operation stops and resets the clock by setting the
   boolean clock_running to false (see OBCS for object clock).
REQUIREMENT_REFERENCES
   NONE
USED_OPERATIONS
   NONE
EXCEPTIONS
```

```
   NONE
DECLARATIONS
   NONE
PSEUDO_CODE
   NONE
CODE
   CONTROL.stop_clock;
   clock_running := FALSE;
   elapsed_time := 0.0;
EXCEPTION_HANDLER
   NONE
END_OPERATION stop_clock;

 reset_clock IS
   PRAGMA CODE_BODY  (EMBEDDED)
   PRAGMA CODE_IMPL  (CALL)
DESCRIPTION
   NONE
REQUIREMENT_REFERENCES
   NONE
USED_OPERATIONS
   NONE
EXCEPTIONS
   NONE
DECLARATIONS
   NONE
PSEUDO_CODE
   NONE
CODE
   CONTROL.reset_clock;
   elapsed_time := 0.0;
EXCEPTION_HANDLER
   NONE
END_OPERATION reset_clock;

 update_time IS
   PRAGMA CODE_BODY  (EMBEDDED)
DESCRIPTION
   This operation simply updates the current value of the
   variable elapsed_time.
REQUIREMENT_REFERENCES
   NONE
USED_OPERATIONS
   NONE
EXCEPTIONS
   NONE
DECLARATIONS
   NONE
PSEUDO_CODE
   NONE
CODE
      sleep(0.2); -- or something similar;
      elapsed_time := elapsed_time + 0.2;
EXCEPTION_HANDLER
   NONE
END_OPERATION update_time;
```

END_OBJECT clock

2.3. (1.2H) Object display

2.3.1. (1.2H1) Problem Definition

2.3.1.1. (1.2H1.1) Statement of the Problem

This is the Problem Statement unit of the object display in the design rocket.

This object is repsonsible for displaying the current values of altitude, velocity, acceleration, and fuel.

2.3.1.2. (1.2H1.2) Analysis Of Requirements

This is the Requirements Analysis unit of the object display in the design rocket.

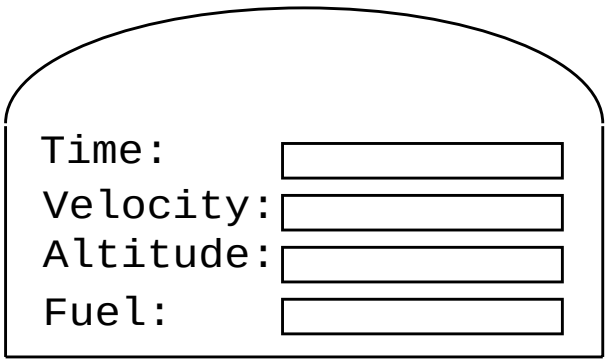
This object needs to display current altitude, current velocity, and current fuel in some way.

2.3.2. (1.2H2) Informal Solution Strategy

This is the Informal Solution unit of the object display in the design rocket.

The outputs will be displayed as strip_charts of value against elapsed time.

A possible layout might be as follows.



2.3.3. (1.2H3) Formalisation of the Strategy

2.3.3.1. (1.2H3.1) Identification of Objects

This is the Object Identification unit of the object display in the design rocket.

-- NONE --

2.3.3.2. (1.2H3.2) Identification of Operations

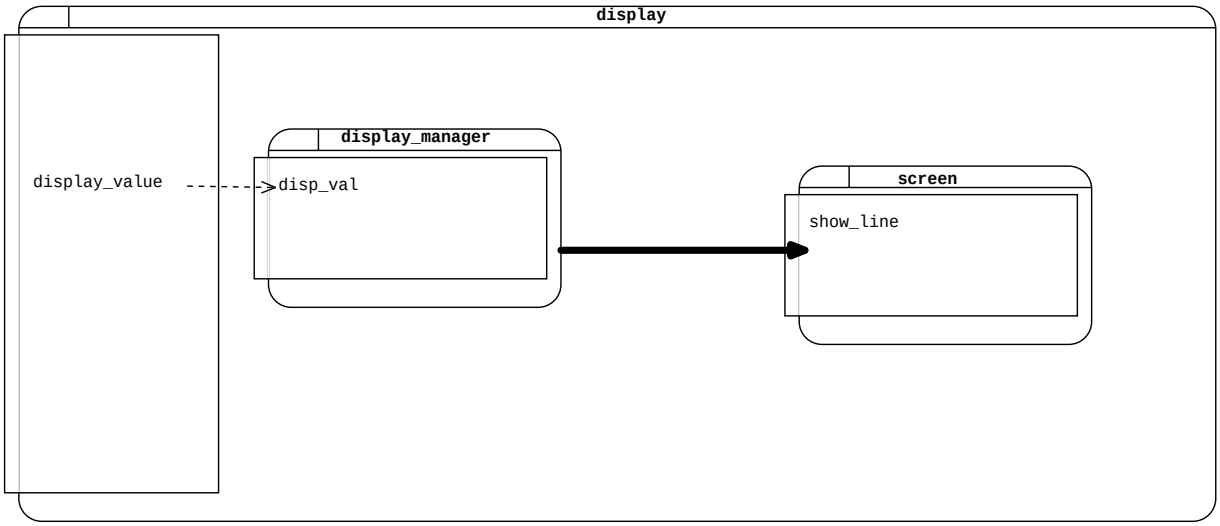
This is the Operation Identification unit of the object display in the design rocket.

-- none - leaf node.

2.3.3.3. (1.2H3.3) Grouping Operations and Objects

-- NONE --

2.3.3.4. (1.2H3.4) Graphical Description



2.3.3.5. (1.2H3.5) Justification of Design Decisions

This is the Justification of Design Decision unit of the object "display" in the design "rocket".

2.3.4. (1.2H4) Formalisation of the Solution

```
 OBJECT  display IS PASSIVE

    PRAGMA EXCEPTION LOG (NO)
    PRAGMA USE CLAUSES (YES)

 DESCRIPTION
    The object display handles the displaying of the current fuel,
    the current acceleration and the calculated current velocity
    value.
 IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
    NONE

 REQUIREMENT_REFERENCES
     XR01 ( xref_requirement_1 )  ;
    **** W: Requirement XR01 is not fulfilled by any operation or obcs
     XR04 ( xref_requirement_4 )  ;
    **** W: Requirement XR04 is not fulfilled by any operation or obcs
```

```
PROVIDED_INTERFACE
  CONSTANTS
    NONE
  TYPES
    NONE
  DATA
    NONE
  DECLARATIONS
    NONE
  OPERATIONS
      display_value (
          value : IN Float );
  OPERATION_SETS
    NONE
  EXCEPTIONS
    fatal_error;

REQUIRED_INTERFACE
  OBJECTS
     standard_io;
    **** W: object is not used by display
  ENVIRONMENT_OBJECTS
    NONE
  CLASS_OBJECTS
    NONE
  CONSTANTS
    NONE
  TYPES
    standard_io.file_type;
  DATA
    NONE
  OPERATIONS
     standard_io.put_line;
  EXCEPTIONS
    standard_io.data_error;

DATAFLOWS
    NONE

OBJECT_CONTROL_STRUCTURE
    NONE

PRIVATE
  CONSTANTS
    NONE
  TYPES
    NONE
  DATA
    NONE

INTERNALS
  OBJECTS
     display_manager;
    **** I: object not used
     screen;
    **** I: object not used
```

```
  ENVIRONMENT_OBJECTS
    NONE
  CLASS_OBJECTS
    NONE
  DECLARATIONS
    NONE
  OPERATIONS
    NONE
  EXCEPTIONS
    NONE

OPERATION_CONTROL_STRUCTURE
 display_value (
         value : IN Float ) IS
    PRAGMA CODE_BODY  (SEPARATE)
    PRAGMA CODE_IMPL  (RENAME)
DESCRIPTION
    This operation takes the data it receives and prints it
    out in some way.
REQUIREMENT_REFERENCES
    NONE
USED_OPERATIONS
     standard_io.put_line;
EXCEPTIONS
    display.fatal_error;
DECLARATIONS
    NONE
PSEUDO_CODE
    NONE
CODE
    standard_io.put_line (value);
EXCEPTION_HANDLER
  WHEN standard_io.data_error
  DESCRIPTION
    NONE
  PSEUDO_CODE
    NONE
  CODE
    -- panic
    RAISE fatal_error;

    **** W: operation is implemented by "display_manager.disp_val" in dictionary
END_OPERATION display_value;


END_OBJECT display
```

2.4. (1.2.1H) Object display_manager

2.4.1. (1.2.1H1) Problem Definition

2.4.1.1. (1.2.1H1.1) Statement of the Problem

(Not Available)

2.4.1.2. (1.2.1H1.2) Analysis Of Requirements

(Not Available)

2.4.2. (1.2.1H2) Informal Solution Strategy

(Not Available)

2.4.3. (1.2.1H3) Formalisation of the Strategy

2.4.3.1. (1.2.1H3.1) Identification of Objects

(Not Available)

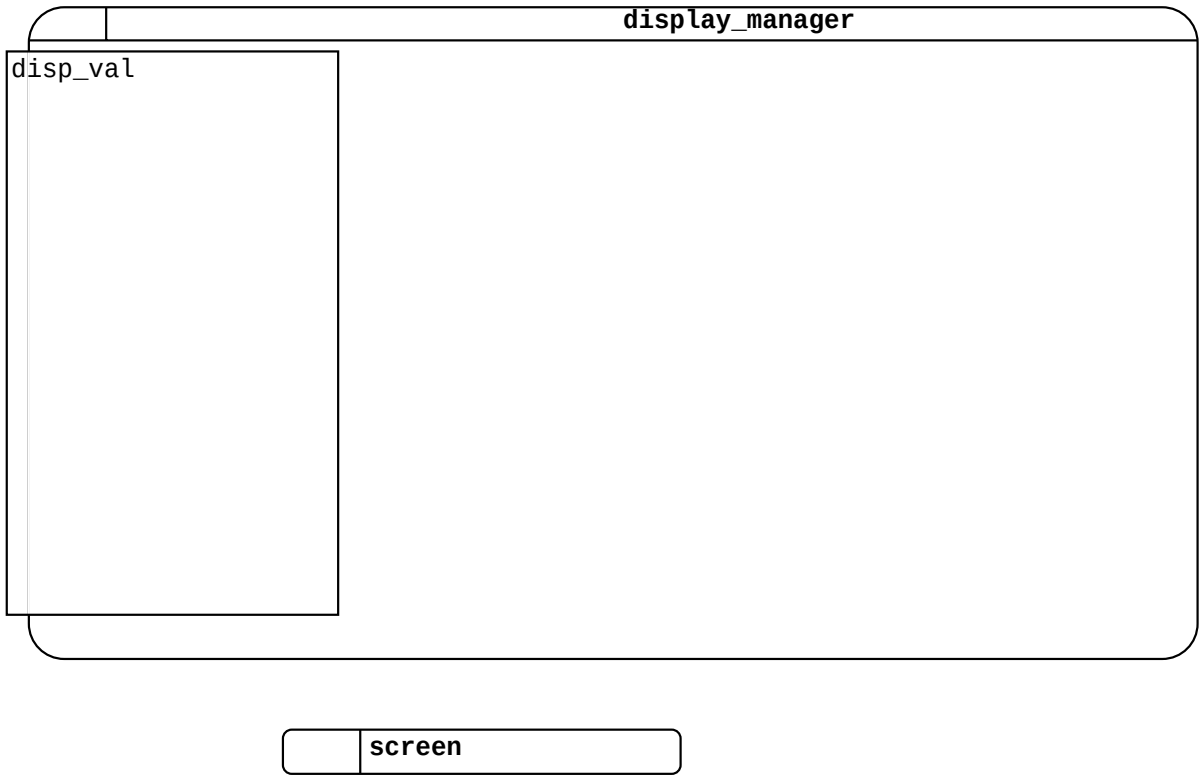
2.4.3.2. (1.2.1H3.2) Identification of Operations

(Not Available)

2.4.3.3. (1.2.1H3.3) Grouping Operations and Objects

(Not Available)

2.4.3.4. (1.2.1H3.4) Graphical Description



2.4.3.5. (1.2.1H3.5) Justification of Design Decisions

(Not Available)

2.4.4. (1.2.1H4) Formalisation of the Solution

```
 OBJECT  display_manager IS PASSIVE

   PRAGMA EXCEPTION LOG (NO)
   PRAGMA USE CLAUSES (YES)

 DESCRIPTION
   NONE
 IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   NONE

 REQUIREMENT_REFERENCES
   NONE

 PROVIDED_INTERFACE
  CONSTANTS
   NONE
```

```
  TYPES
    NONE
  DATA
    NONE
  DECLARATIONS
    NONE
  OPERATIONS
     disp_val;
  OPERATION_SETS
    NONE
  EXCEPTIONS
    NONE

REQUIRED_INTERFACE
  OBJECTS
    NONE
  ENVIRONMENT_OBJECTS
    NONE
  CLASS_OBJECTS
    NONE
  CONSTANTS
    NONE
  TYPES
    NONE
  DATA
    NONE
  OPERATIONS
    NONE
  EXCEPTIONS
    NONE

DATAFLOWS
    NONE

OBJECT_CONTROL_STRUCTURE
    NONE

PRIVATE
  CONSTANTS
    NONE
  TYPES
    NONE
  DATA
    NONE

INTERNALS
  OBJECTS
    NONE
  ENVIRONMENT_OBJECTS
    NONE
  CLASS_OBJECTS
    NONE
  DECLARATIONS
    NONE
  OPERATIONS
    NONE
```

```
  EXCEPTIONS
    NONE

OPERATION_CONTROL_STRUCTURE
 disp_val IS
    PRAGMA CODE_BODY  (EMBEDDED)
    PRAGMA CODE_IMPL  (CALL)
DESCRIPTION
    NONE
REQUIREMENT_REFERENCES
    NONE
USED_OPERATIONS
    NONE
EXCEPTIONS
    NONE
IMPLEMENTED_BY
    <Undefined_Object>.<Identifier>;
END_OPERATION disp_val;


END_OBJECT display_manager
```

2.5. (1.2.2H) Object screen

2.5.1. (1.2.2H1) Problem Definition

2.5.1.1. (1.2.2H1.1) Statement of the Problem

(Not Available)

2.5.1.2. (1.2.2H1.2) Analysis Of Requirements

(Not Available)

2.5.2. (1.2.2H2) Informal Solution Strategy

(Not Available)

2.5.3. (1.2.2H3) Formalisation of the Strategy

2.5.3.1. (1.2.2H3.1) Identification of Objects

(Not Available)

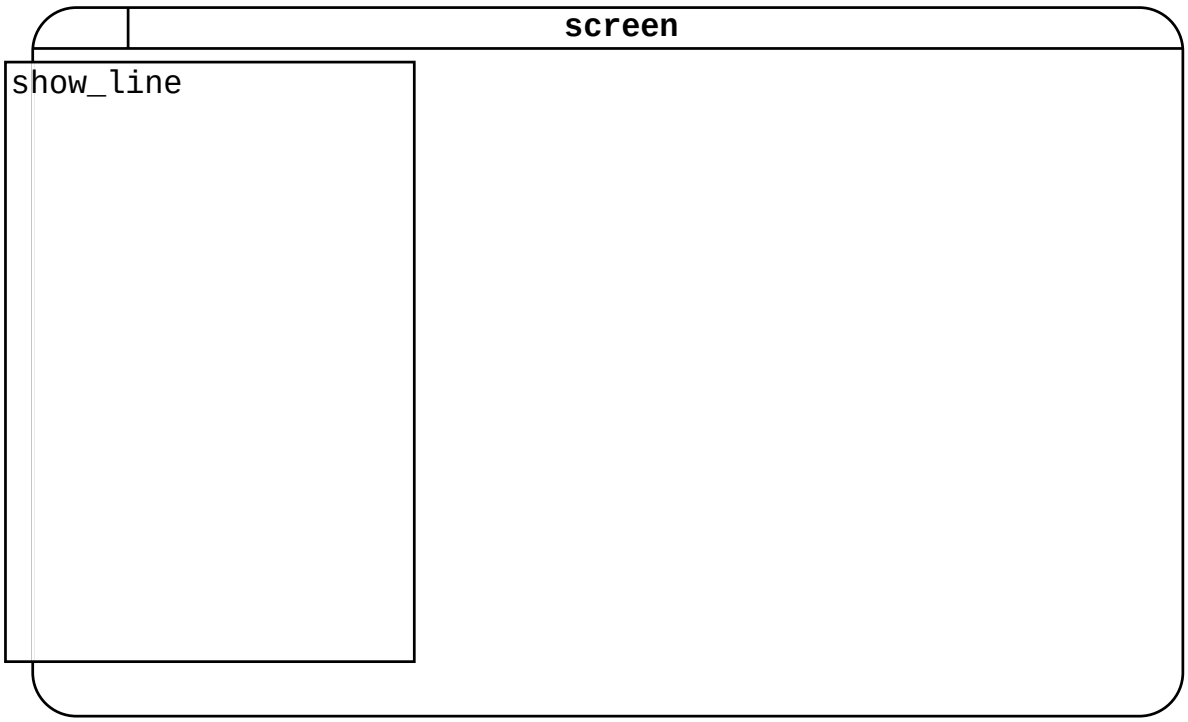
2.5.3.2. (1.2.2H3.2) Identification of Operations

(Not Available)

2.5.3.3. (1.2.2H3.3) Grouping Operations and Objects

(Not Available)

2.5.3.4. (1.2.2H3.4) Graphical Description



2.5.3.5. (1.2.2H3.5) Justification of Design Decisions

(Not Available)

2.5.4. (1.2.2H4) Formalisation of the Solution

```
 OBJECT  screen IS PASSIVE

    PRAGMA EXCEPTION LOG (NO)
    PRAGMA USE CLAUSES (YES)

 DESCRIPTION
    NONE
 IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
    NONE

 REQUIREMENT_REFERENCES
    NONE

 PROVIDED_INTERFACE
  CONSTANTS
    NONE
  TYPES
    NONE
  DATA
    NONE
```

```
  DECLARATIONS
    NONE
  OPERATIONS
     show_line;
  OPERATION_SETS
    NONE
  EXCEPTIONS
    NONE

REQUIRED_INTERFACE
  OBJECTS
    NONE
  ENVIRONMENT_OBJECTS
    NONE
  CLASS_OBJECTS
    NONE
  CONSTANTS
    NONE
  TYPES
    NONE
  DATA
    NONE
  OPERATIONS
    NONE
  EXCEPTIONS
    NONE

DATAFLOWS
    NONE

OBJECT_CONTROL_STRUCTURE
    NONE

PRIVATE
  CONSTANTS
    NONE
  TYPES
    NONE
  DATA
    NONE

INTERNALS
  OBJECTS
    NONE
  ENVIRONMENT_OBJECTS
    NONE
  CLASS_OBJECTS
    NONE
  DECLARATIONS
    NONE
  OPERATIONS
    NONE
  EXCEPTIONS
    NONE

OPERATION_CONTROL_STRUCTURE
```

```
 show_line IS
   PRAGMA CODE_BODY  (EMBEDDED)
   PRAGMA CODE_IMPL  (CALL)
DESCRIPTION
   NONE
REQUIREMENT_REFERENCES
   NONE
USED_OPERATIONS
   NONE
EXCEPTIONS
   NONE
IMPLEMENTED_BY
   <Undefined_Object>.<Identifier>;
END_OPERATION show_line;


END_OBJECT screen
```

2.6. (1.3H) Object engine

2.6.1. (1.3H1) Problem Definition

2.6.1.1. (1.3H1.1) Statement of the Problem

The task of the engine is to take the required thrust and set the current thrust accordingly.

2.6.1.2. (1.3H1.2) Analysis Of Requirements

The engine will ignite upon receipt of instruction to do so from the pilot, get the required thrust and set the current thrust accordingly. It will switch off upon receipt of the call to switch off engine from the pilot.

2.6.2. (1.3H2) Informal Solution Strategy

The engine will be a 'leaf' object of the system. It will provide functionality to ignite the engine, switch it off ascertain the required thrust and set the current thrust.

2.6.3. (1.3H3) Formalisation of the Strategy

2.6.3.1. (1.3H3.1) Identification of Objects

-- NO CHILD OBJECTS --

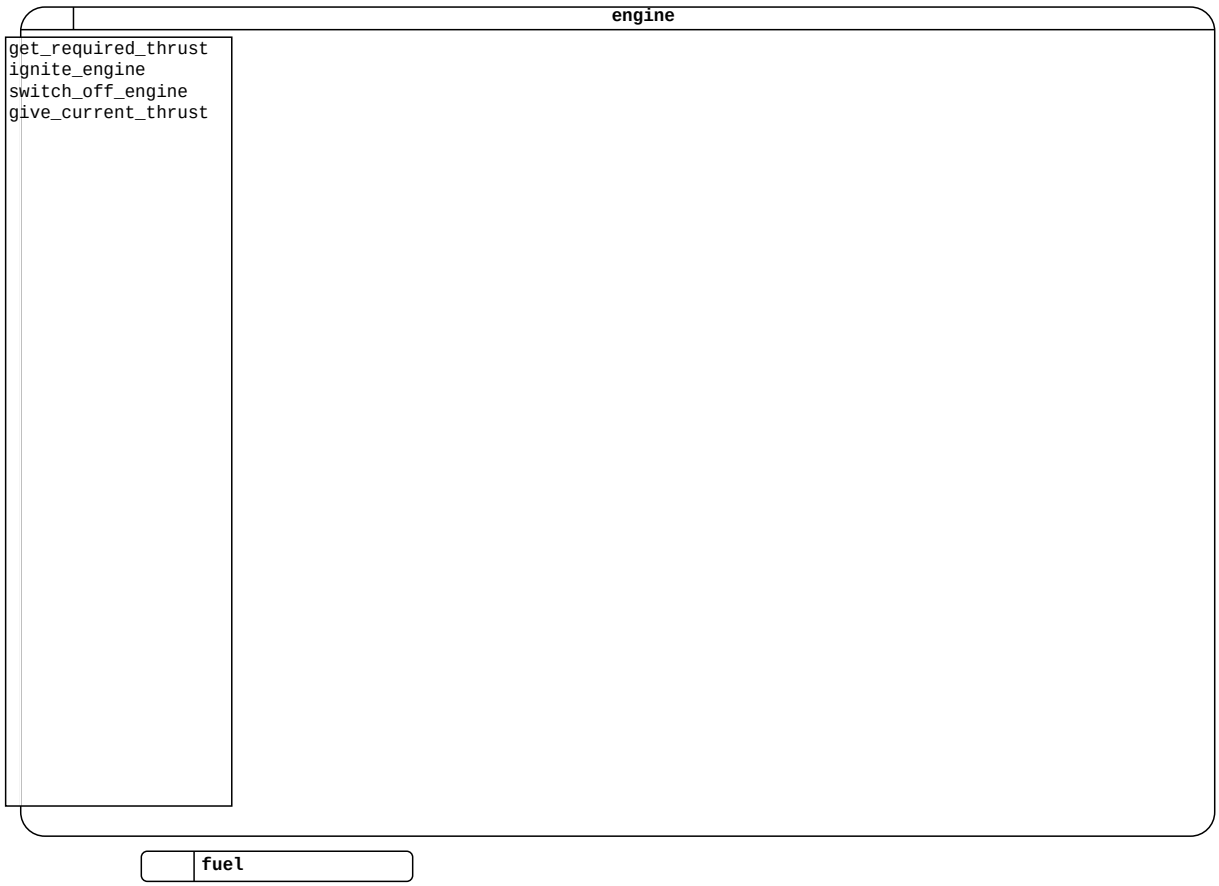
2.6.3.2. (1.3H3.2) Identification of Operations

-- NO CHILD OBJECTS/OPERATIONS --

2.6.3.3. (1.3H3.3) Grouping Operations and Objects

-- NO CHILD OBJECTS/OPERATIONS - the functionality of this object
will be expresed through its external and internal operations.--

2.6.3.4. (1.3H3.4) Graphical Description



2.6.3.5. (1.3H3.5) Justification of Design Decisions

(Not Available)

2.6.4. (1.3H4) Formalisation of the Solution

```
 OBJECT  engine IS PASSIVE

   PRAGMA EXCEPTION LOG (NO)
   PRAGMA USE CLAUSES (YES)

 DESCRIPTION
   NONE
IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   NONE

 REQUIREMENT_REFERENCES
   NONE
```

```
PROVIDED_INTERFACE
  CONSTANTS
    NONE
  TYPES
    NONE
  DATA
    NONE
  DECLARATIONS
    NONE
  OPERATIONS
      get_required_thrust (
         required_thrust : IN Float );
      ignite_engine;
      switch_off_engine;
      give_current_thrust (
         current_thrust : IN Float );
  OPERATION_SETS
    NONE
  EXCEPTIONS
    NONE

REQUIRED_INTERFACE
  OBJECTS
     fuel;
  ENVIRONMENT_OBJECTS
    NONE
  CLASS_OBJECTS
    NONE
  CONSTANTS
    NONE
  TYPES
    NONE
  DATA
    NONE
  OPERATIONS
     fuel.get_fuel_used;
  EXCEPTIONS
    NONE

DATAFLOWS
     fuel_used => fuel;

OBJECT_CONTROL_STRUCTURE
    NONE

PRIVATE
  CONSTANTS
    NONE
  TYPES
    NONE
  DATA
    NONE

INTERNALS
  OBJECTS
    NONE
```

```
  ENVIRONMENT_OBJECTS
    NONE
  CLASS_OBJECTS
    NONE
  DECLARATIONS
    engine_running : BOOLEAN;
    req_thrust : real;
  OPERATIONS
    NONE
  EXCEPTIONS
    NONE

OPERATION_CONTROL_STRUCTURE
 get_required_thrust (
        required_thrust : IN Float ) IS
    PRAGMA CODE_BODY  (EMBEDDED)
    PRAGMA CODE_IMPL  (CALL)
DESCRIPTION
    this routine just sets the data type req_thrust to the value
    supplied.
REQUIREMENT_REFERENCES
    NONE
USED_OPERATIONS
    NONE
EXCEPTIONS
    NONE
DECLARATIONS
    NONE
PSEUDO_CODE
    NONE
CODE
    req_thrust := required_thrust;
EXCEPTION_HANDLER
    NONE
END_OPERATION get_required_thrust;

 ignite_engine IS
    PRAGMA CODE_BODY  (EMBEDDED)
    PRAGMA CODE_IMPL  (CALL)
DESCRIPTION
    -- start the engine running.
REQUIREMENT_REFERENCES
    NONE
USED_OPERATIONS
    NONE
EXCEPTIONS
    NONE
DECLARATIONS
    NONE
PSEUDO_CODE
    NONE
CODE
    engine_running := TRUE;
EXCEPTION_HANDLER
    NONE
END_OPERATION ignite_engine;
```

```
 switch_off_engine IS
   PRAGMA CODE_BODY  (EMBEDDED)
   PRAGMA CODE_IMPL  (CALL)
DESCRIPTION
   -- stop the engine.
REQUIREMENT_REFERENCES
   NONE
USED_OPERATIONS
   NONE
EXCEPTIONS
   NONE
DECLARATIONS
   NONE
PSEUDO_CODE
   NONE
CODE
   engine_running := FALSE;
EXCEPTION_HANDLER
   NONE
END_OPERATION switch_off_engine;

 give_current_thrust (
        current_thrust : IN Float ) IS
   PRAGMA CODE_BODY  (EMBEDDED)
   PRAGMA CODE_IMPL  (CALL)
DESCRIPTION
   This routine calculates the current thrust from the fuel used.
REQUIREMENT_REFERENCES
   NONE
USED_OPERATIONS
    fuel.get_fuel_used;
EXCEPTIONS
   NONE
DECLARATIONS
   NONE
PSEUDO_CODE
   NONE
CODE
   fuel_used : REAL;
   fuel_usage_rate : CONSTANT := 0.009;
   thrust_fuel_ratio : CONSTANT := 10000.0;

   fuel_used := req_thrust * fuel_usage_rate;

   fuel.get_fuel_used (fuel_used);

   current_thrust := req_thrust * fuel_used * thrust_fuel_ratio;
EXCEPTION_HANDLER
   NONE
END_OPERATION give_current_thrust;


END_OBJECT engine
```

2.7. (1.4H) Object fuel

2.7.1. (1.4H1) Problem Definition

2.7.1.1. (1.4H1.1) Statement of the Problem

The fuel system maintains a record of the current fuel levels.

2.7.1.2. (1.4H1.2) Analysis Of Requirements

The fuel system should continuously display the current fuel level.

2.7.2. (1.4H2) Informal Solution Strategy

The fuel system will subtract the fuel used by the engine from the initial fuel and display the current fuel.

2.7.3. (1.4H3) Formalisation of the Strategy

2.7.3.1. (1.4H3.1) Identification of Objects

-- NONE --

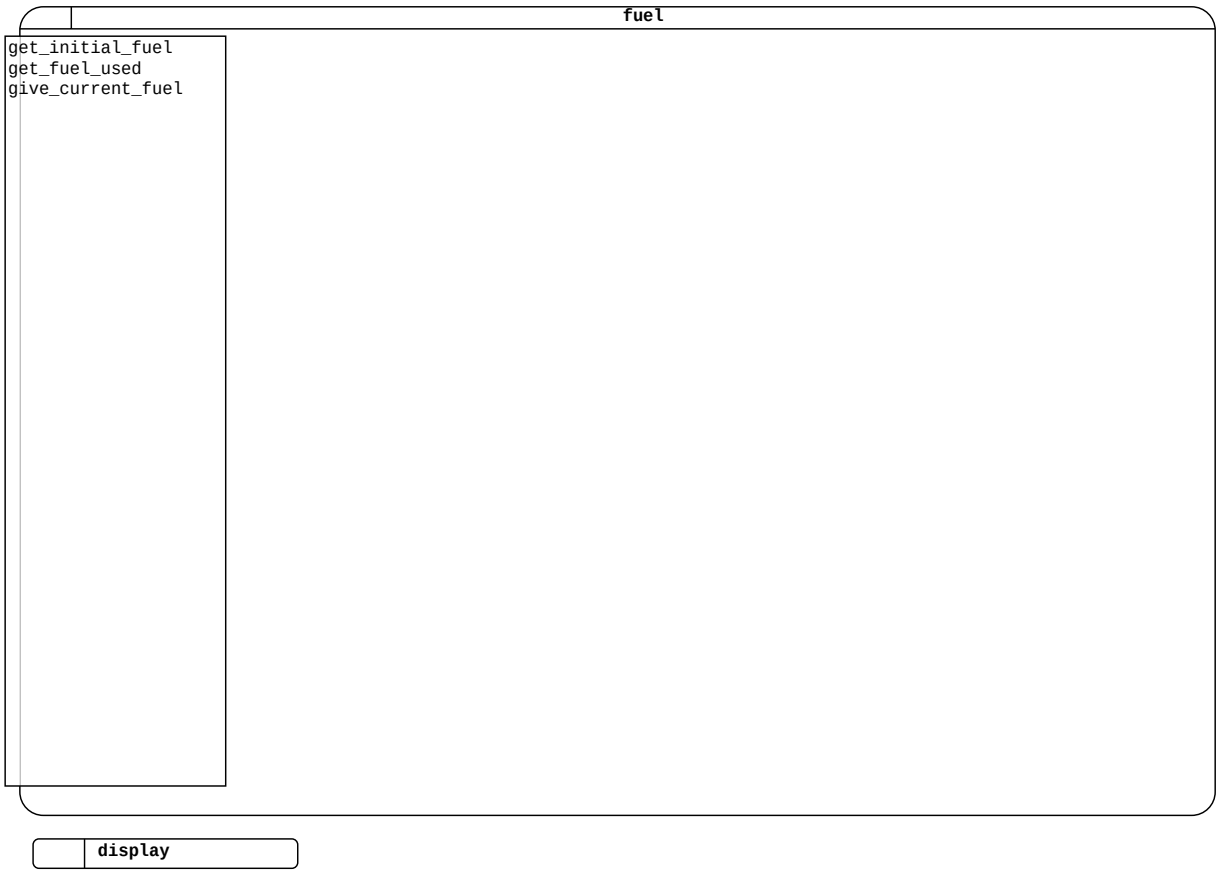
2.7.3.2. (1.4H3.2) Identification of Operations

-- NONE --

2.7.3.3. (1.4H3.3) Grouping Operations and Objects

-- NONE --

2.7.3.4. (1.4H3.4) Graphical Description



2.7.3.5. (1.4H3.5) Justification of Design Decisions

(Not Available)

2.7.4. (1.4H4) Formalisation of the Solution

OBJECT fuel IS PASSIVE

PRAGMA EXCEPTION LOG (NO)
PRAGMA USE CLAUSES (YES)

DESCRIPTION
Maintains the current fuel level.
IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
NONE

REQUIREMENT_REFERENCES
NONE

```
PROVIDED_INTERFACE
  CONSTANTS
    NONE
  TYPES
    NONE
  DATA
    NONE
  DECLARATIONS
    NONE
  OPERATIONS
      get_initial_fuel (
          initial_fuel : IN Float );
      get_fuel_used (
          used_fuel : IN Float );
      give_current_fuel (
          current_fuel : OUT Float );
  OPERATION_SETS
    NONE
  EXCEPTIONS
    fatal_error;

REQUIRED_INTERFACE
  OBJECTS
     display;
  ENVIRONMENT_OBJECTS
    NONE
  CLASS_OBJECTS
    NONE
  CONSTANTS
    NONE
  TYPES
    NONE
  DATA
    NONE
  OPERATIONS
     display.display_value;
  EXCEPTIONS
    NONE

DATAFLOWS
     current_fuel => display;

OBJECT_CONTROL_STRUCTURE
    NONE

PRIVATE
  CONSTANTS
    NONE
  TYPES
    NONE
  DATA
    NONE

INTERNALS
  OBJECTS
    NONE
```

```
  ENVIRONMENT_OBJECTS
    NONE
  CLASS_OBJECTS
    NONE
  DECLARATIONS
    init_fuel : Float;
    fuel_used : Float;
    current_fuel : Float;
  OPERATIONS
    NONE
  EXCEPTIONS
    NONE

OPERATION_CONTROL_STRUCTURE
 get_initial_fuel (
        initial_fuel : IN Float ) IS
    PRAGMA CODE_BODY  (EMBEDDED)
    PRAGMA CODE_IMPL  (CALL)
DESCRIPTION
    NONE
REQUIREMENT_REFERENCES
    NONE
USED_OPERATIONS
    NONE
EXCEPTIONS
    NONE
DECLARATIONS
    NONE
PSEUDO_CODE
    NONE
CODE
    init_fuel := initial_fuel;
EXCEPTION_HANDLER
    NONE
END_OPERATION get_initial_fuel;

 get_fuel_used (
        used_fuel : IN Float ) IS
    PRAGMA CODE_BODY  (SEPARATE)
    PRAGMA CODE_IMPL  (CALL)
DESCRIPTION
    NONE
REQUIREMENT_REFERENCES
    NONE
USED_OPERATIONS
    NONE
EXCEPTIONS
    NONE
DECLARATIONS
    NONE
PSEUDO_CODE
    NONE
CODE
    fuel_used := used_fuel;
EXCEPTION_HANDLER
    NONE
```

```
END_OPERATION get_fuel_used;

 give_current_fuel (
        current_fuel : OUT Float ) IS
   PRAGMA CODE_BODY  (EMBEDDED)
   PRAGMA CODE_IMPL  (RENAME)
DESCRIPTION
   This operation calculates the current fuel and displays it.
REQUIREMENT_REFERENCES
   NONE
USED_OPERATIONS
    display.display_value;
EXCEPTIONS
   fuel.fatal_error;
DECLARATIONS
   NONE
PSEUDO_CODE
   NONE
CODE
   current_fuel := init_fuel - fuel_used;
   if current_fuel < 0.0 then
     raise fatal_error;
   end if;
   display.display_value (current_fuel);
EXCEPTION_HANDLER
   NONE
END_OPERATION give_current_fuel;


END_OBJECT fuel
```

2.8. (1.5H) Object navigation_system

2.8.1. (1.5H1) Problem Definition

2.8.1.1. (1.5H1.1) Statement of the Problem

The navigation system calculates the present altitude and velocity for every pulse of the clock.

2.8.1.2. (1.5H1.2) Analysis Of Requirements

Calculates the current altitude and velocity. Gets the required altitude and payload mass, current thrust and elapsed time as inputs.

2.8.2. (1.5H2) Informal Solution Strategy

Calculates the current altitude and velocity from the required altitude and payload mass, current thrust and elapsed time.

2.8.3. (1.5H3) Formalisation of the Strategy

2.8.3.1. (1.5H3.1) Identification of Objects

-- None - leaf object --

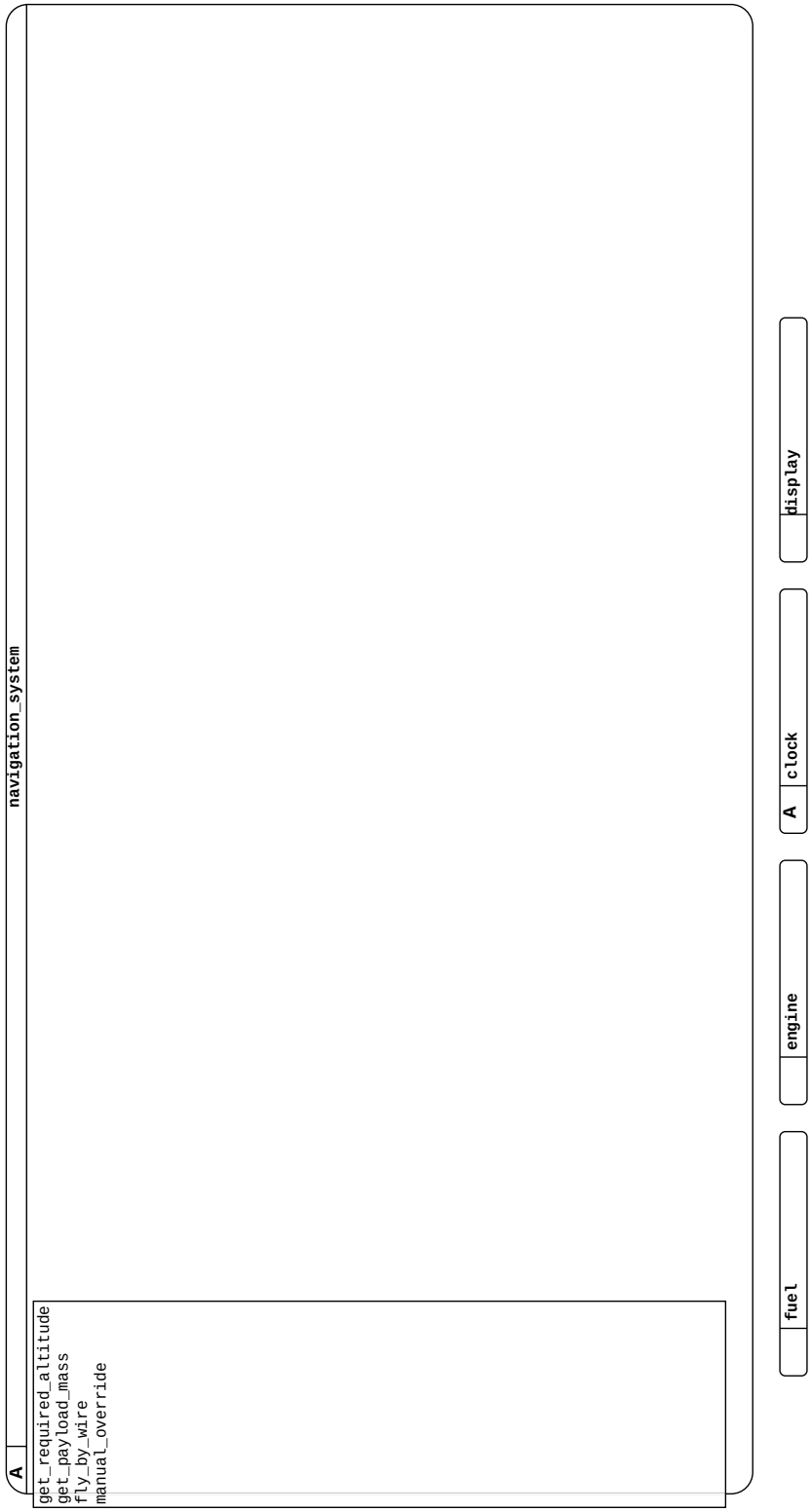
2.8.3.2. (1.5H3.2) Identification of Operations

-- None - leaf object --

2.8.3.3. (1.5H3.3) Grouping Operations and Objects

-- None - leaf object --

2.8.3.4. (1.5H3.4) Graphical Description



2.8.3.5. (1.5H3.5) Justification of Design Decisions

(Not Available)

2.8.4. (1.5H4) Formalisation of the Solution

```
 OBJECT  navigation_system IS ACTIVE

   PRAGMA EXCEPTION LOG (NO)
   PRAGMA USE CLAUSES (YES)

   **** W: active object has no constrained operations
DESCRIPTION
   NONE
IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   NONE

REQUIREMENT_REFERENCES
   NONE

PROVIDED_INTERFACE
  CONSTANTS
    NONE
  TYPES
    NONE
  DATA
    NONE
  DECLARATIONS
    NONE
  OPERATIONS
     get_required_altitude (
        req_alt : IN Float );
     get_payload_mass (
        payload_mass : IN Float );
     fly_by_wire (
        required_alt : IN Float;
        payload_mass : IN Float );
     manual_override;
  OPERATION_SETS
    NONE
  EXCEPTIONS
    NONE

REQUIRED_INTERFACE
  OBJECTS
    clock;
    engine;
    display;
    fuel;
  ENVIRONMENT_OBJECTS
    NONE
  CLASS_OBJECTS
    NONE
  CONSTANTS
    NONE
```

```
  TYPES
    clock.second;
  DATA
    NONE
  OPERATIONS
     clock.current_time;
     clock.reset_clock;
     engine.give_current_thrust;
     engine.get_required_thrust;
     display.display_value;
     fuel.give_current_fuel;
  EXCEPTIONS
    NONE

DATAFLOWS
     elapsed_time <= clock;
     current_thrust <= engine;
     required_thrust => engine;
     current_velocity => display;
     current_altitude => display;

OBJECT_CONTROL_STRUCTURE
    PRAGMA CODE_BODY (EMBEDDED)
    PRAGMA CODE_SPEC (HIDDEN)
DESCRIPTION
    The task for the navigation system has four entry points. The
    first, fly_by_wire kicks it off and is the only entry accepted
    until it is started (via pilot.start_engine).  Whilst flying
    the payload_mass and required_altitude values can be modified
    and the task can be halted by a call to manual override.
CONSTRAINED_OPERATIONS
    NONE
REQUIREMENT_REFERENCES
    NONE
USED_OPERATIONS
    NONE
EXCEPTIONS
    NONE
DECLARATIONS
    NONE
PSEUDO_CODE
    NONE
CODE
    loop
      select
        when not started =>
          accept fly_by_wire(req_alt : in Float;
                             pay_load_mess : in Float) do
            required_altitude := req_alt;
            payload_mass := pay_load_mass;
          end fly_by_wire;
          loop
            display_outputs();
          end loop;
      or
        when started =>
```

```
          accept get_required_altitude(req_alt : in Float) do
            required_altitude := req_alt;
          end get_required_altitude;
      or
        when started =>
          accept get_payload_mass(pay_load_mass : in Float) do
            payload_mass := pay_load_mass;
          end get_payload_mass;
      or
        when started =>
          accept manual_override;
          started := False;
      end select;
    end loop;
EXCEPTION_HANDLER
    NONE

PRIVATE
  CONSTANTS
    NONE
  TYPES
    NONE
  DATA
    NONE

INTERNALS
  OBJECTS
    NONE
  ENVIRONMENT_OBJECTS
    NONE
  CLASS_OBJECTS
    NONE
  DECLARATIONS
    started : Boolean := False;
    req_altitude : Float;
    payload_mass : Float;
    elapsed_time : Float;
    velocity : Float;
    acceleration : Float;
    altitude : Float;
    initial_altitude : Float := 0.0;
    thrust : Float;
    required_thrust : Float;
    mass : Float;
    current_fuel : Float;
    gravity : Float := 9.81;
    req_alt_attained : Boolean := False;
    rocket_mass : Float := 100.0;
    initial_velocity : Float := 0.0;
    --assume stationary start
  OPERATIONS
     display_outputs;
     show_current_altitude (
         altitude : IN Float );
     show_current_velocity (
         velocity : IN Float );
```

```
 EXCEPTIONS
   NONE

OPERATION_CONTROL_STRUCTURE
 get_required_altitude (
        req_alt : IN Float ) IS
   PRAGMA CODE_BODY  (EMBEDDED)
   PRAGMA CODE_IMPL  (CALL)
DESCRIPTION
   null procedure - the code is in the obcs rendezvous statement.
REQUIREMENT_REFERENCES
   NONE
USED_OPERATIONS
   NONE
EXCEPTIONS
   NONE
DECLARATIONS
   NONE
PSEUDO_CODE
   NONE
CODE
   NONE
EXCEPTION_HANDLER
   NONE
END_OPERATION get_required_altitude;

 get_payload_mass (
        payload_mass : IN Float ) IS
   PRAGMA CODE_BODY  (EMBEDDED)
   PRAGMA CODE_IMPL  (CALL)
DESCRIPTION
   this is a null procedure (entry point into object control
   structure) -- associated code can be seen in the obcs.
REQUIREMENT_REFERENCES
   NONE
USED_OPERATIONS
   NONE
EXCEPTIONS
   NONE
DECLARATIONS
   NONE
PSEUDO_CODE
   NONE
CODE
   NONE
EXCEPTION_HANDLER
   NONE
END_OPERATION get_payload_mass;

 fly_by_wire (
        required_alt : IN Float;
        payload_mass : IN Float ) IS
   PRAGMA CODE_BODY  (EMBEDDED)
   PRAGMA CODE_IMPL  (CALL)
DESCRIPTION
   null procedure - body is in obcs rendezvous statement.
```

REQUIREMENT_REFERENCES
NONE
USED_OPERATIONS
NONE
EXCEPTIONS
NONE
DECLARATIONS
NONE
PSEUDO_CODE
NONE
CODE
NONE
EXCEPTION_HANDLER
NONE
END_OPERATION fly_by_wire;

manual_override IS
PRAGMA CODE_BODY (EMBEDDED)
PRAGMA CODE_IMPL (CALL)
DESCRIPTION
null procedure - code in the obcs rendezvous statement.
REQUIREMENT_REFERENCES
NONE
USED_OPERATIONS
NONE
EXCEPTIONS
NONE
DECLARATIONS
NONE
PSEUDO_CODE
NONE
CODE
NONE
EXCEPTION_HANDLER
NONE
END_OPERATION manual_override;

display_outputs IS
PRAGMA CODE_BODY (EMBEDDED)
DESCRIPTION
The business end.
The procedure calculates the current altitude and current
velocity from the data passed to it.
REQUIREMENT_REFERENCES
NONE
USED_OPERATIONS
clock.current_time;
clock.reset_clock;
engine.give_current_thrust;
engine.get_required_thrust;
fuel.give_current_fuel;
EXCEPTIONS
NONE
DECLARATIONS
NONE
PSEUDO_CODE

```
   NONE
CODE
   clock.current_time (elapsed_time);
   engine.give_current_thrust (thrust);
   fuel.give_current_fuel (current_fuel);

   mass := rocket_mass + payload_mass + current_fuel;
   acceleration := (thrust/mass) - gravity;
   velocity := initial_velocity + acceleration * elapsed_time;
   altitude := initial_altitude +
               0.5 * acceleration * (elapsed_time**2);
   show_current_velocity (velocity);
   show_current_altitude (altitude);

   if altitude >= required_altitude then
      required_thrust := 0.0;
      initial_altitude := altitude;
      req_alt_attained := True; -- used in next branch - HACK
   else
      required_thrust := (acceleration + gravity) * mass;
      if req_alt_attained then
         -- after have reached req_alt initial_altitude will be
         -- starting altitude (will also need to reset clock)
         -- thus the clock will not be recording the time since
         -- the start of the rocket launch but the amount of time
         -- that the rocket has been going in any one direction.
         -- Hopefully this naive method should produce an effect
         -- the current altitude fluctuating either side of the
         -- required altitude marker.
         initial_altitude := altitude;
         clock.reset_clock;
      end if;
   end if;

   engine.get_required_thrust (required_thrust);
EXCEPTION_HANDLER
   NONE
END_OPERATION display_outputs;

 show_current_altitude (
        altitude : IN Float ) IS
   PRAGMA CODE_BODY  (EMBEDDED)
DESCRIPTION
   This procedure just calls the procedure display_value of object
   display with the current_altitude as the parameter.
REQUIREMENT_REFERENCES
   NONE
USED_OPERATIONS
    display.display_value;
EXCEPTIONS
   NONE
DECLARATIONS
   NONE
PSEUDO_CODE
   NONE
CODE
```

```
   display.display_value (current_altitude);
EXCEPTION_HANDLER
   NONE
END_OPERATION show_current_altitude;

 show_current_velocity (
        velocity : IN Float ) IS
   PRAGMA CODE_BODY  (EMBEDDED)
DESCRIPTION
   this procedure just calls the procedure display_value of the
   object display with the current_velocity as a parameter.
REQUIREMENT_REFERENCES
   NONE
USED_OPERATIONS
    display.display_value;
EXCEPTIONS
   NONE
DECLARATIONS
   NONE
PSEUDO_CODE
   NONE
CODE
   display.display_value (current_velocity);
EXCEPTION_HANDLER
   NONE
END_OPERATION show_current_velocity;


END_OBJECT navigation_system
```

2.9. (1.6H) Object pilot

2.9.1. (1.6H1) Problem Definition

2.9.1.1. (1.6H1.1) Statement of the Problem

The pilot is the main controller of the rocket. It starts the system running, kicking off the fly-by-wire system (navigation_ system) and starting the clock. It also provides the means for stopping the rocket (abort_engine).

2.9.1.2. (1.6H1.2) Analysis Of Requirements

-- see problem statement for this object.

2.9.2. (1.6H2) Informal Solution Strategy

The pilot starts the system running, kicking off the fly-by-wire system (navigation_system) and starting the clock. It also provides the means for stopping the rocket (abort_engine).

2.9.3. (1.6H3) Formalisation of the Strategy

2.9.3.1. (1.6H3.1) Identification of Objects

-- none - leaf object

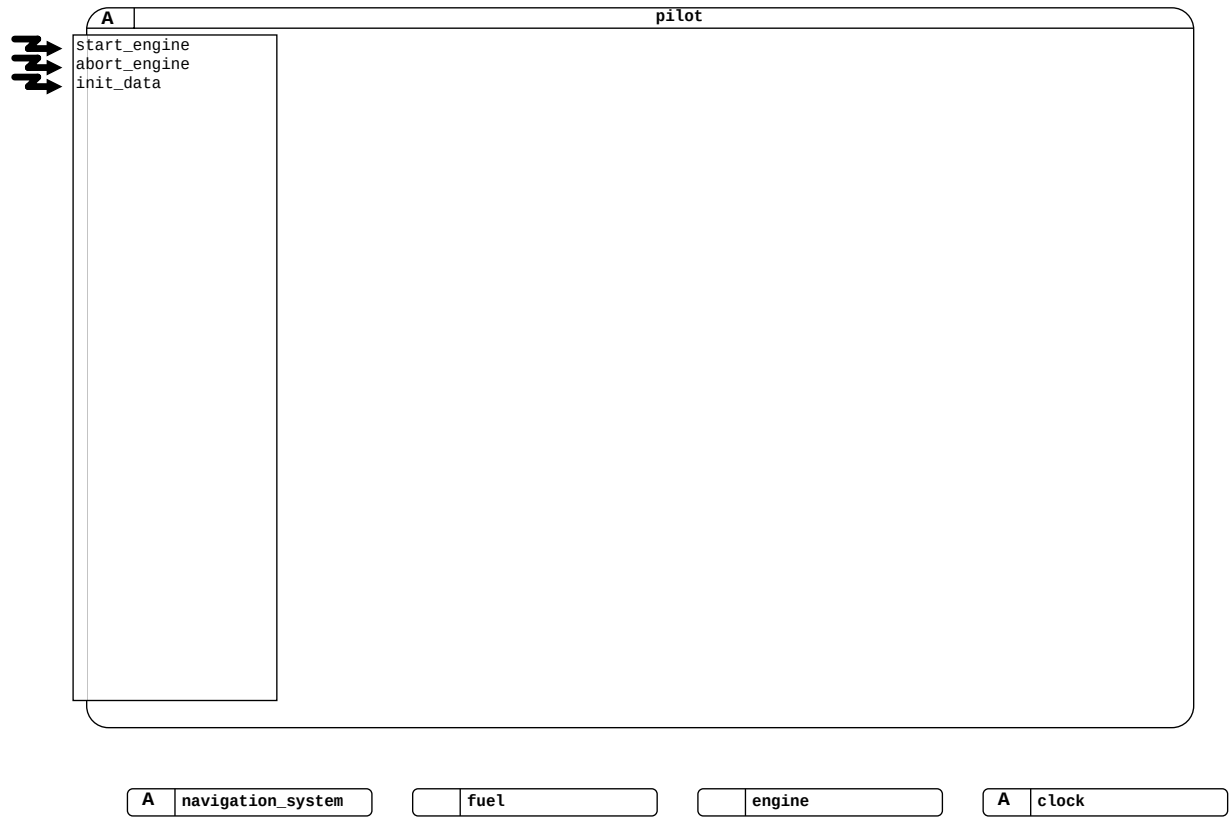
2.9.3.2. (1.6H3.2) Identification of Operations

-- none - leaf object

2.9.3.3. (1.6H3.3) Grouping Operations and Objects

-- none - leaf object

2.9.3.4. (1.6H3.4) Graphical Description



2.9.3.5. (1.6H3.5) Justification of Design Decisions

(Not Available)

2.9.4. (1.6H4) Formalisation of the Solution

```
 OBJECT  pilot IS ACTIVE

    PRAGMA EXCEPTION LOG (NO)
    PRAGMA USE CLAUSES (YES)

 DESCRIPTION
    NONE
 IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
    NONE

 REQUIREMENT_REFERENCES
     XR03 ( xref_requirement_3 )  ;

 PROVIDED_INTERFACE
  CONSTANTS
```

```
    NONE
  TYPES
    NONE
  DATA
    NONE
  DECLARATIONS
    NONE
  OPERATIONS
      start_engine;
      abort_engine;
      init_data (
          req_thrust : IN Float;
          req_alt : IN Float;
          cargo_mass : IN Float;
          init_fuel : IN Float );
  OPERATION_SETS
    NONE
  EXCEPTIONS
    NONE

REQUIRED_INTERFACE
  OBJECTS
     navigation_system;
     fuel;
     engine;
     clock;
  ENVIRONMENT_OBJECTS
    NONE
  CLASS_OBJECTS
    NONE
  CONSTANTS
    NONE
  TYPES
    NONE
  DATA
    NONE
  OPERATIONS
     navigation_system.fly_by_wire;
     navigation_system.get_required_altitude;
     navigation_system.get_payload_mass;
     navigation_system.manual_override;
     fuel.get_initial_fuel;
     engine.get_required_thrust;
     engine.ignite_engine;
     engine.switch_off_engine;
     clock.start_clock;
     clock.stop_clock;
  EXCEPTIONS
    NONE

DATAFLOWS
     initial_fuel => fuel;
     required_thrust => engine;
     required_altitude => navigation_system;
     payload_mass => navigation_system;
```

```
OBJECT_CONTROL_STRUCTURE
   PRAGMA CODE_BODY (EMBEDDED)
   PRAGMA CODE_SPEC (HIDDEN)
DESCRIPTION
   NONE
CONSTRAINED_OPERATIONS
   start_engine;
   abort_engine;
   init_data;
REQUIREMENT_REFERENCES
   NONE
USED_OPERATIONS
   NONE
EXCEPTIONS
   NONE
DECLARATIONS
   NONE
PSEUDO_CODE
   NONE
CODE
   started : Boolean := False;
   loop
     select
       when not started =>
         accept start_engine;
         started := True;
       when started =>
         accept init_data(req_thrust : in Float;
                          req_alt : in Float;
                          cargo_mass : in Float;
                          initial_fuel : in Float);
       when started =>
        accept abort_engine;
        started := False;
     end select;
   end loop;
EXCEPTION_HANDLER
   NONE

PRIVATE
 CONSTANTS
   NONE
 TYPES
   NONE
 DATA
   NONE

INTERNALS
 OBJECTS
   NONE
 ENVIRONMENT_OBJECTS
   NONE
 CLASS_OBJECTS
   NONE
 DECLARATIONS
   engine_running : Boolean;
```

```
  OPERATIONS
    NONE
  EXCEPTIONS
    NONE

OPERATION_CONTROL_STRUCTURE
 start_engine IS
    PRAGMA CODE_BODY  (EMBEDDED)
    PRAGMA CODE_IMPL  (CALL)
DESCRIPTION
    NONE
REQUIREMENT_REFERENCES
    NONE
USED_OPERATIONS
     navigation_system.fly_by_wire;
     engine.ignite_engine;
     clock.start_clock;
EXCEPTIONS
    NONE
DECLARATIONS
    NONE
PSEUDO_CODE
    NONE
CODE
    navigation_system.fly_by_wire;
    engine.ignite_engine;
    clock.start_clock;
EXCEPTION_HANDLER
    NONE
END_OPERATION start_engine;

 abort_engine IS
    PRAGMA CODE_BODY  (EMBEDDED)
    PRAGMA CODE_IMPL  (CALL)
DESCRIPTION
    NONE
REQUIREMENT_REFERENCES
    NONE
USED_OPERATIONS
     clock.stop_clock;
     navigation_system.manual_override;
     engine.switch_off_engine;
EXCEPTIONS
    NONE
DECLARATIONS
    NONE
PSEUDO_CODE
    NONE
CODE
    clock.stop_clock;
    navigation_system.manual_override;
    engine.switch_off_engine;
EXCEPTION_HANDLER
    NONE
END_OPERATION abort_engine;
```

```
 init_data (
        req_thrust : IN Float;
        req_alt : IN Float;
        cargo_mass : IN Float;
        init_fuel : IN Float ) IS
   PRAGMA CODE_BODY  (EMBEDDED)
   PRAGMA CODE_IMPL  (CALL)
DESCRIPTION
   NONE
REQUIREMENT_REFERENCES
   NONE
USED_OPERATIONS
    navigation_system.get_required_altitude;
    navigation_system.get_payload_mass;
    engine.get_required_thrust;
    fuel.get_initial_fuel;
EXCEPTIONS
   NONE
DECLARATIONS
   NONE
PSEUDO_CODE
   NONE
CODE
   navigation_system.get_required_altitude(req_alt);
   navigation_system.get_payload_mass(cargo_mass);
   fuel.get_initial_fuel(init_fuel);
   engine.get_required_thrust(req_thrust);
EXCEPTION_HANDLER
   NONE
END_OPERATION init_data;


END_OBJECT pilot
```

2.10. (2H) Object standard_io

2.10.1. (2H1) Problem Definition

2.10.1.1. (2H1.1) Statement of the Problem

(Not Available)

2.10.1.2. (2H1.2) Analysis Of Requirements

(Not Available)

2.10.2. (2H2) Informal Solution Strategy

(Not Available)

2.10.3. (2H3) Formalisation of the Strategy

2.10.3.1. (2H3.1) Identification of Objects

(Not Available)

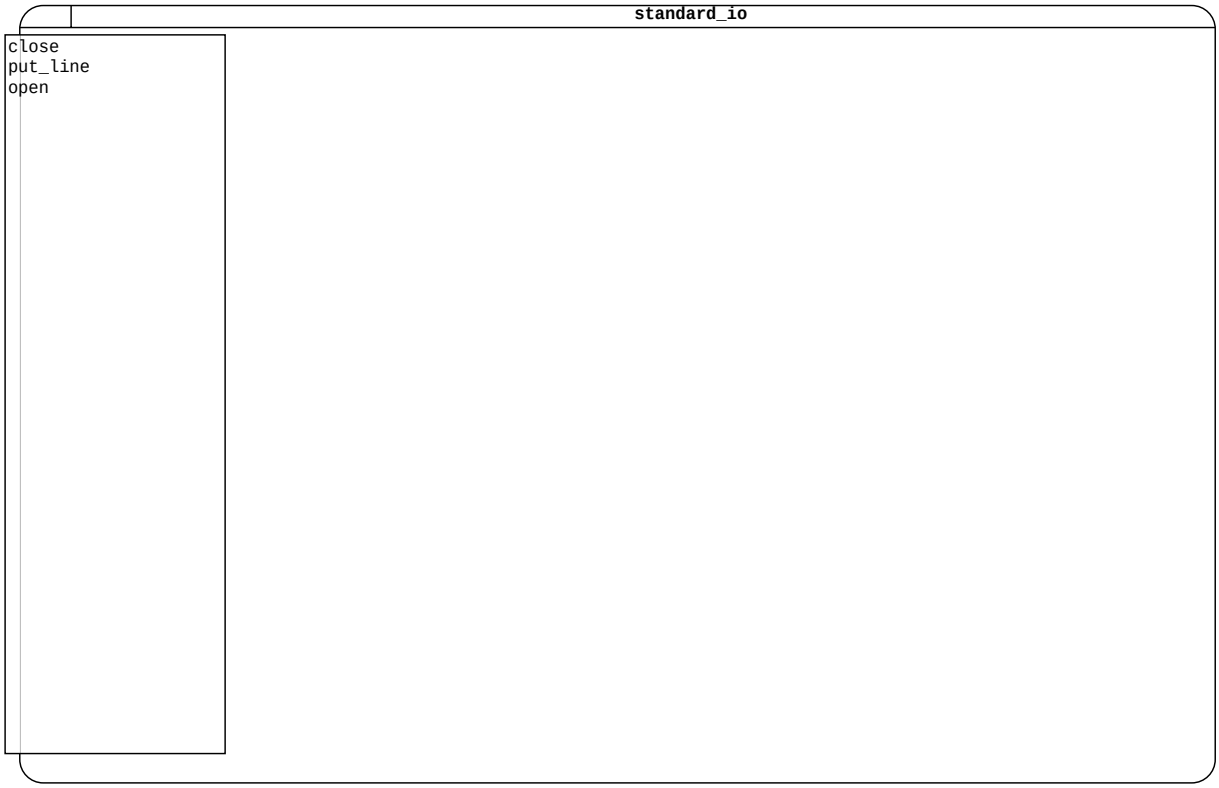
2.10.3.2. (2H3.2) Identification of Operations

(Not Available)

2.10.3.3. (2H3.3) Grouping Operations and Objects

(Not Available)

2.10.3.4. (2H3.4) Graphical Description



2.10.3.5. (2H3.5) Justification of Design Decisions

(Not Available)

2.10.4. (2H4) Formalisation of the Solution

```
 OBJECT  standard_io IS PASSIVE

   PRAGMA EXCEPTION LOG (NO)
   PRAGMA USE CLAUSES (YES)

DESCRIPTION
   NONE
IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
   NONE

REQUIREMENT_REFERENCES
   NONE

PROVIDED_INTERFACE
 CONSTANTS
   NONE
```

```
  TYPES
    file_type IS LIMITED PRIVATE;
    file_mode IS
       (In_file,
        Out_file);
  DATA
    NONE
  DECLARATIONS
    NONE
  OPERATIONS
      open (
         file : IN OUT file_type;
         name : IN String;
         mode : IN file_mode := Out_file );
      put_line (
         file : IN OUT file_type;
         line : IN String );
      close (
         file : IN OUT file_type );
  OPERATION_SETS
    NONE
  EXCEPTIONS
    data_error;

REQUIRED_INTERFACE
  OBJECTS
    NONE
  ENVIRONMENT_OBJECTS
    NONE
  CLASS_OBJECTS
    NONE
  CONSTANTS
    NONE
  TYPES
    NONE
  DATA
    NONE
  OPERATIONS
    NONE
  EXCEPTIONS
    NONE

DATAFLOWS
    NONE

OBJECT_CONTROL_STRUCTURE
    NONE

PRIVATE
  CONSTANTS
    NONE
  TYPES
    file_type IS <Type_Definition>;
  DATA
    NONE
```

```
INTERNALS
 OBJECTS
  NONE
 ENVIRONMENT_OBJECTS
  NONE
 CLASS_OBJECTS
  NONE
 DECLARATIONS
  Type file_type Is record
    -- FCB details
    id : Integer;
    buffer : String (1 .. 1024);
    -- etc.
  end record;
  --W: type is already declared
 OPERATIONS
  NONE
 EXCEPTIONS
  NONE

OPERATION_CONTROL_STRUCTURE
 open (
        file : IN OUT file_type;
        name : IN String;
        mode : IN file_mode := Out_file ) IS
   PRAGMA CODE_BODY  (EMBEDDED)
   PRAGMA CODE_IMPL  (CALL)
DESCRIPTION
   NONE
REQUIREMENT_REFERENCES
   NONE
USED_OPERATIONS
   NONE
EXCEPTIONS
   NONE
DECLARATIONS
   NONE
PSEUDO_CODE
   NONE
CODE
   NONE
EXCEPTION_HANDLER
   NONE
END_OPERATION open;

 put_line (
        file : IN OUT file_type;
        line : IN String ) IS
   PRAGMA CODE_BODY  (EMBEDDED)
   PRAGMA CODE_IMPL  (CALL)
DESCRIPTION
   NONE
REQUIREMENT_REFERENCES
   NONE
USED_OPERATIONS
   NONE
```

```
EXCEPTIONS
   standard_io.data_error;
DECLARATIONS
   NONE
PSEUDO_CODE
   NONE
CODE
   NONE
EXCEPTION_HANDLER
   NONE
END_OPERATION put_line;

 close (
        file : IN OUT file_type ) IS
   PRAGMA CODE_BODY  (EMBEDDED)
   PRAGMA CODE_IMPL  (CALL)
DESCRIPTION
   NONE
REQUIREMENT_REFERENCES
   NONE
USED_OPERATIONS
   NONE
EXCEPTIONS
   NONE
DECLARATIONS
   NONE
PSEUDO_CODE
   NONE
CODE
   NONE
EXCEPTION_HANDLER
   NONE
END_OPERATION close;


END_OBJECT standard_io
```

A. OPERATION/OBJECT CROSS-REFERENCE

Operation	Provided By
abort_engine	pilot
abort_lift_off	rocket
close	standard_io
current_time	clock
disp_val	display_manager
display_value	display
fly_by_wire	navigation_system
get_fuel_used	fuel
get_initial_fuel	fuel
get_payload_mass	navigation_system
get_required_altitude	navigation_system
get_required_thrust	engine
give_current_fuel	fuel
give_current_thrust	engine
ignite_engine	engine
init_data	pilot
initialise_data	rocket
manual_override	navigation_system
open	standard_io
put_line	standard_io
reset_clock	clock
show_line	screen
start	rocket
start_clock	clock
start_engine	pilot
stop_clock	clock
switch_off_engine	engine

B. REQUIREMENT/OBJECT CROSS-REFERENCE

Requirement	Fulfilled By	Qual.
XR01 (xref_requirement_1)	rocket display	
XR02 (xref_requirement_2)	rocket	
XR03 (xref_requirement_3)	pilot	
XR04 (xref_requirement_4)	display	

C. OBJECT/REQUIREMENT CROSS-REFERENCE

Object	Fulfills	Qual.
clock		
display	XR01 (xref_requirement_1) XR04 (xref_requirement_4)	
display_manager		
engine		
fuel		
navigation_system		
pilot	XR03 (xref_requirement_3)	
rocket	XR01 (xref_requirement_1) XR02 (xref_requirement_2)	
screen		
standard_io		

--- END OF DOCUMENT ---

ROCKET EXAMPLE HOOD Toolset Test Data	Reference: ROCKET/EG
	Issue: 1.2
	Page: i

Table of Contents

Chapter 1. INTRODUCTION 1
Chapter 2. SYSTEM DESIGN 2
2.1. (1H) Object rocket 3
2.1.1. (1H1) Problem Definition 3
2.1.1.1. (1H1.1) Statement of the Problem 3
2.1.1.2. (1H1.2) Analysis Of Requirements 3
2.1.2. (1H2) Informal Solution Strategy 3
2.1.3. (1H3) Formalisation of the Strategy 4
2.1.3.1. (1H3.1) Identification of Objects 4
2.1.3.2. (1H3.2) Identification of Operations 4
2.1.3.3. (1H3.3) Grouping Operations and Objects 4
2.1.3.4. (1H3.4) Graphical Description 5
2.1.3.5. (1H3.5) Justification of Design Decisions 5
2.1.4. (1H4) Formalisation of the Solution 5
2.2. (1.1H) Object clock 9
2.2.1. (1.1H1) Problem Definition 9
2.2.1.1. (1.1H1.1) Statement of the Problem 9
2.2.1.2. (1.1H1.2) Analysis Of Requirements 9
2.2.2. (1.1H2) Informal Solution Strategy 9
2.2.3. (1.1H3) Formalisation of the Strategy 9
2.2.3.1. (1.1H3.1) Identification of Objects 9
2.2.3.2. (1.1H3.2) Identification of Operations 9
2.2.3.3. (1.1H3.3) Grouping Operations and Objects 9
2.2.3.4. (1.1H3.4) Graphical Description 10
2.2.3.5. (1.1H3.5) Justification of Design Decisions 10
2.2.4. (1.1H4) Formalisation of the Solution 10
2.3. (1.2H) Object display 16
2.3.1. (1.2H1) Problem Definition 16
2.3.1.1. (1.2H1.1) Statement of the Problem 16
2.3.1.2. (1.2H1.2) Analysis Of Requirements 16
2.3.2. (1.2H2) Informal Solution Strategy 16
2.3.3. (1.2H3) Formalisation of the Strategy 16
2.3.3.1. (1.2H3.1) Identification of Objects 16
2.3.3.2. (1.2H3.2) Identification of Operations 16
2.3.3.3. (1.2H3.3) Grouping Operations and Objects 17
2.3.3.4. (1.2H3.4) Graphical Description 17
2.3.3.5. (1.2H3.5) Justification of Design Decisions 17
2.3.4. (1.2H4) Formalisation of the Solution 17
2.4. (1.2.1H) Object display_manager 20
2.4.1. (1.2.1H1) Problem Definition 20

2.4.1.1. (1.2.1H1.1) Statement of the Problem . . . 20
2.4.1.2. (1.2.1H1.2) Analysis Of Requirements . . . 20
2.4.2. (1.2.1H2) Informal Solution Strategy . . . 20
2.4.3. (1.2.1H3) Formalisation of the Strategy . . . 20
2.4.3.1. (1.2.1H3.1) Identification of Objects . . . 20
2.4.3.2. (1.2.1H3.2) Identification of Operations . . . 20
2.4.3.3. (1.2.1H3.3) Grouping Operations and Objects . . . 20
2.4.3.4. (1.2.1H3.4) Graphical Description . . . 21
2.4.3.5. (1.2.1H3.5) Justification of Design Decisions . . . 21
2.4.4. (1.2.1H4) Formalisation of the Solution . . . 21
2.5. (1.2.2H) Object screen . . . 24
2.5.1. (1.2.2H1) Problem Definition . . . 24
2.5.1.1. (1.2.2H1.1) Statement of the Problem . . . 24
2.5.1.2. (1.2.2H1.2) Analysis Of Requirements . . . 24
2.5.2. (1.2.2H2) Informal Solution Strategy . . . 24
2.5.3. (1.2.2H3) Formalisation of the Strategy . . . 24
2.5.3.1. (1.2.2H3.1) Identification of Objects . . . 24
2.5.3.2. (1.2.2H3.2) Identification of Operations . . . 24
2.5.3.3. (1.2.2H3.3) Grouping Operations and Objects . . . 24
2.5.3.4. (1.2.2H3.4) Graphical Description . . . 25
2.5.3.5. (1.2.2H3.5) Justification of Design Decisions . . . 25
2.5.4. (1.2.2H4) Formalisation of the Solution . . . 25
2.6. (1.3H) Object engine . . . 28
2.6.1. (1.3H1) Problem Definition . . . 28
2.6.1.1. (1.3H1.1) Statement of the Problem . . . 28
2.6.1.2. (1.3H1.2) Analysis Of Requirements . . . 28
2.6.2. (1.3H2) Informal Solution Strategy . . . 28
2.6.3. (1.3H3) Formalisation of the Strategy . . . 28
2.6.3.1. (1.3H3.1) Identification of Objects . . . 28
2.6.3.2. (1.3H3.2) Identification of Operations . . . 28
2.6.3.3. (1.3H3.3) Grouping Operations and Objects . . . 28
2.6.3.4. (1.3H3.4) Graphical Description . . . 29
2.6.3.5. (1.3H3.5) Justification of Design Decisions . . . 29
2.6.4. (1.3H4) Formalisation of the Solution . . . 29
2.7. (1.4H) Object fuel . . . 33
2.7.1. (1.4H1) Problem Definition . . . 33
2.7.1.1. (1.4H1.1) Statement of the Problem . . . 33
2.7.1.2. (1.4H1.2) Analysis Of Requirements . . . 33
2.7.2. (1.4H2) Informal Solution Strategy . . . 33
2.7.3. (1.4H3) Formalisation of the Strategy . . . 33
2.7.3.1. (1.4H3.1) Identification of Objects . . . 33
2.7.3.2. (1.4H3.2) Identification of Operations . . . 33
2.7.3.3. (1.4H3.3) Grouping Operations and Objects . . . 33

2.7.3.4. (1.4H3.4) Graphical Description . . . 34
2.7.3.5. (1.4H3.5) Justification of Design Decisions . . . 34
2.7.4. (1.4H4) Formalisation of the Solution . . . 34
2.8. (1.5H) Object navigation_system . . . 38
2.8.1. (1.5H1) Problem Definition . . . 38
2.8.1.1. (1.5H1.1) Statement of the Problem . . . 38
2.8.1.2. (1.5H1.2) Analysis Of Requirements . . . 38
2.8.2. (1.5H2) Informal Solution Strategy . . . 38
2.8.3. (1.5H3) Formalisation of the Strategy . . . 38
2.8.3.1. (1.5H3.1) Identification of Objects . . . 38
2.8.3.2. (1.5H3.2) Identification of Operations . . . 38
2.8.3.3. (1.5H3.3) Grouping Operations and Objects . . . 38
2.8.3.4. (1.5H3.4) Graphical Description . . . 39
2.8.3.5. (1.5H3.5) Justification of Design Decisions . . . 40
2.8.4. (1.5H4) Formalisation of the Solution . . . 40
2.9. (1.6H) Object pilot . . . 47
2.9.1. (1.6H1) Problem Definition . . . 47
2.9.1.1. (1.6H1.1) Statement of the Problem . . . 47
2.9.1.2. (1.6H1.2) Analysis Of Requirements . . . 47
2.9.2. (1.6H2) Informal Solution Strategy . . . 47
2.9.3. (1.6H3) Formalisation of the Strategy . . . 47
2.9.3.1. (1.6H3.1) Identification of Objects . . . 47
2.9.3.2. (1.6H3.2) Identification of Operations . . . 47
2.9.3.3. (1.6H3.3) Grouping Operations and Objects . . . 47
2.9.3.4. (1.6H3.4) Graphical Description . . . 48
2.9.3.5. (1.6H3.5) Justification of Design Decisions . . . 48
2.9.4. (1.6H4) Formalisation of the Solution . . . 48
2.10. (2H) Object standard_io . . . 53
2.10.1. (2H1) Problem Definition . . . 53
2.10.1.1. (2H1.1) Statement of the Problem . . . 53
2.10.1.2. (2H1.2) Analysis Of Requirements . . . 53
2.10.2. (2H2) Informal Solution Strategy . . . 53
2.10.3. (2H3) Formalisation of the Strategy . . . 53
2.10.3.1. (2H3.1) Identification of Objects . . . 53
2.10.3.2. (2H3.2) Identification of Operations . . . 53
2.10.3.3. (2H3.3) Grouping Operations and Objects . . . 53
2.10.3.4. (2H3.4) Graphical Description . . . 54
2.10.3.5. (2H3.5) Justification of Design Decisions . . . 54
2.10.4. (2H4) Formalisation of the Solution . . . 54
Appendix A. OPERATION/OBJECT CROSS-REFERENCE . . . 58
Appendix B. REQUIREMENT/OBJECT CROSS-REFERENCE . . . 59
Appendix C. OBJECT/REQUIREMENT CROSS-REFERENCE . . . 60