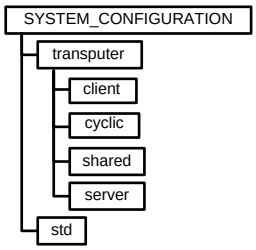
SYSTEM_CONFIGURATION IS

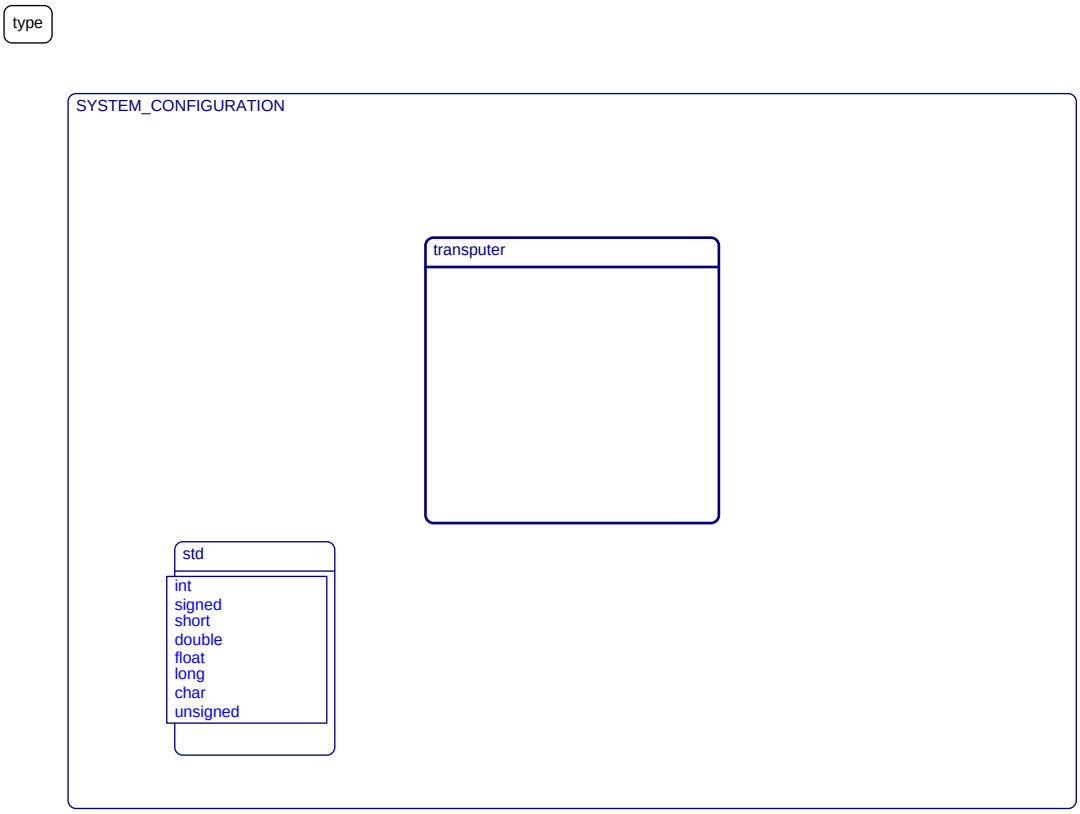
ROOT_OBJECTS
--|\\Groslulu\home4\stood\stood4.3\libs\std|--,
--|\\Groslulu\home4\stood\stood4.3\examples\transputer|--

END

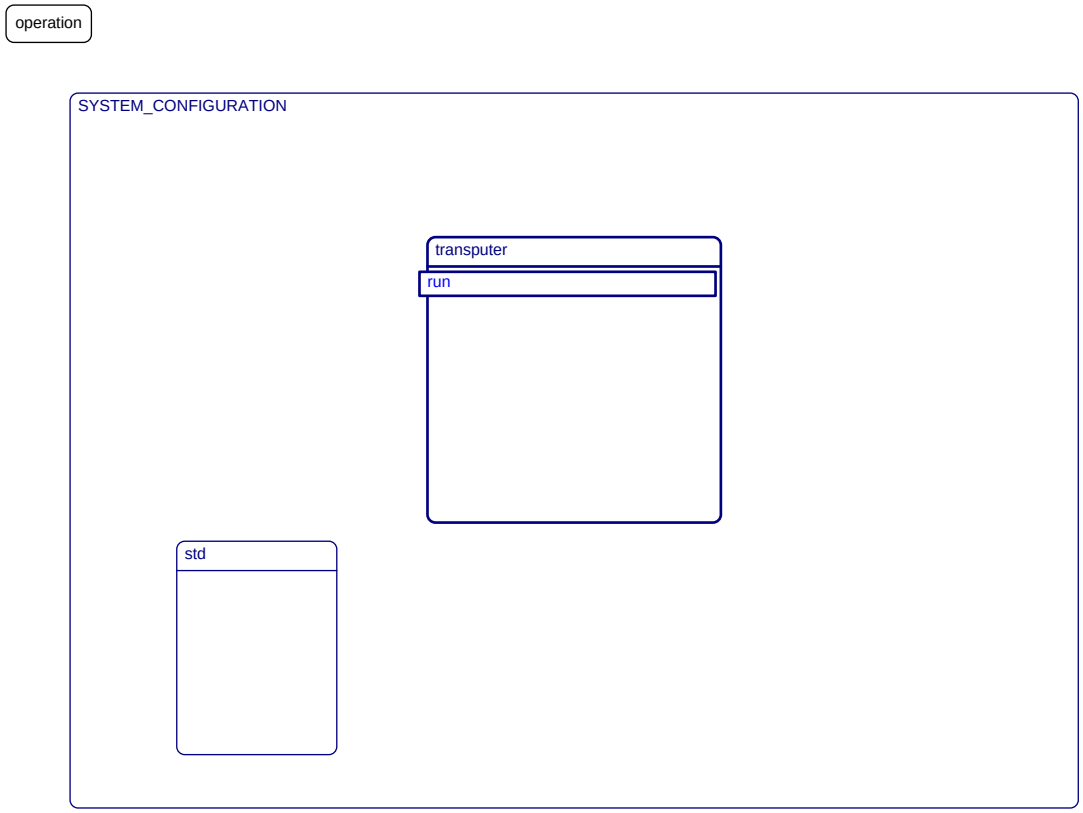
Design Tree



Structural (types) Diagram



Functional (oper.) Diagram



OBJECT transputer IS

PASSIVE

pragmas

```
PRAGMA main
  (operation_name => run)
PRAGMA cc
  (compiler => gcc)
PRAGMA target
  (OS => transputer,
   include_dir => transputer)
```

DESCRIPTION

PROBLEM

Statement of the Problem (text)

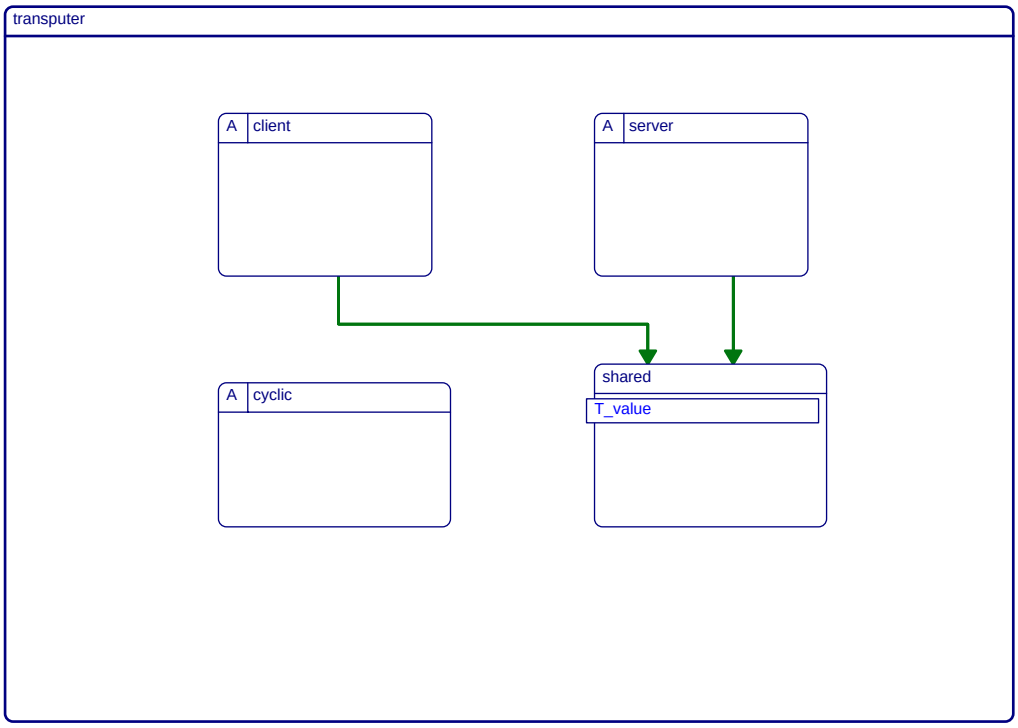
This is a demonstration example to present HOOD real-time features and their implementation into C code with a Real Real-Time Operating System library.
Code generation rules may be customized to comply with other targets.

SOLUTION

Structural Description

Structural (types) Diagram

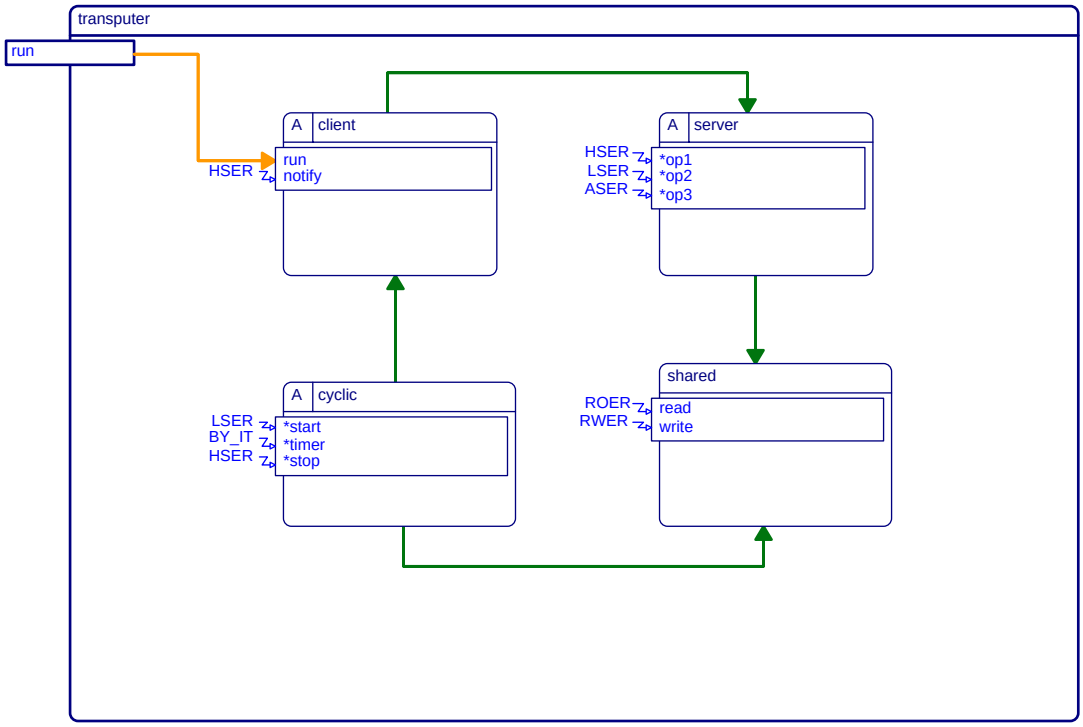
type



Functional Description

Functional (oper.) Diagram

operation



PROVIDED_INTERFACE

OPERATIONS

run

operation declaration (hood)

```
run;
```

REQUIRED_INTERFACE

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

INTERNALS

OBJECTS

```
client;
cyclic;
shared;
server;
```

OPERATIONS

run

implemented_by

```
client.run
```

END transputer

OBJECT client IS

ACTIVE

pragmas

```
PRAGMA target_param
  (param => priority,
   value => low)
PRAGMA target_param
  (param => wsize,
   value => 1000)
```

PROVIDED_INTERFACE

OPERATIONS

run

operation declaration (hood)

```
run;
```

notify

operation declaration (hood)

```
notify;
```

OBJECT_CONTROL_STRUCTURE

constrained operations

```
notify CONSTRAINED_BY HSER;
```

REQUIRED_INTERFACE

```
OBJECT server;
  TYPES
    NONE
  CONSTANTS
    NONE
  OPERATION_SETS
    NONE
  OPERATIONS
    op1; op2;
  EXCEPTIONS
    NONE
OBJECT shared;
  TYPES
    T_value;
  CONSTANTS
    NONE
  OPERATION_SETS
    NONE
  OPERATIONS
    NONE
  EXCEPTIONS
    NONE
OBJECT std;
  TYPES
    int;
  CONSTANTS
    NONE
  OPERATION_SETS
    NONE
  OPERATIONS
    NONE
  EXCEPTIONS
    NONE
```

INTERNALS

DATA

mode

data declaration (c)

```
int mode = 1;
```

data access from pseudo_code

```
(da) client.mode IS USED BY
  (op) client.notify [RW]
  (op) client.run [RW]
```

OPERATION_CONTROL_STRUCTURES

OPERATION run IS

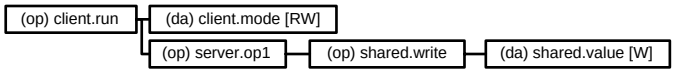
used operations

```
server.op1
```

operation code (pseudo)

```
mode := 1
server.op1(x => mode)
```

call tree from pseudo_code



operation code (c)

```
{
  mode = 1;
  server__op1((shared__T_value)mode,0);
  return 0;
}
```

END run

OPERATION notify IS

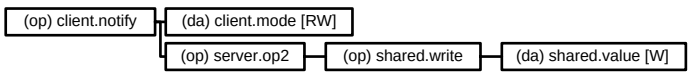
used operations

```
server.op2
```

operation code (pseudo)

```
mode := 2
server.op2(x => mode)
```

call tree from pseudo_code



operation code (c)

```
{
  mode = 2;
  server__op2((shared__T_value)mode);
}
```

END notify

END client

OBJECT cyclic IS

ACTIVE

PROVIDED_INTERFACE

OPERATIONS

start

operation declaration (hood)
```
start;
```

timer

operation declaration (hood)
```
timer;
```

stop

operation declaration (hood)
```
stop;
```

OBJECT_CONTROL_STRUCTURE

constrained operations
```
start CONSTRAINED_BY STATE LSER;
timer CONSTRAINED_BY BY_IT --|10|-- ASER STATE;
stop CONSTRAINED_BY STATE HSER;
```

REQUIRED_INTERFACE

```
OBJECT client;
  TYPES
    NONE
  CONSTANTS
    NONE
  OPERATION_SETS
    NONE
  OPERATIONS
    notify;
  EXCEPTIONS
    NONE
OBJECT shared;
  TYPES
    NONE
  CONSTANTS
    NONE
  OPERATION_SETS
    NONE
  OPERATIONS
    write;
  EXCEPTIONS
    NONE
OBJECT std;
  TYPES
    int;
  CONSTANTS
    NONE
  OPERATION_SETS
    NONE
  OPERATIONS
    NONE
  EXCEPTIONS
    NONE
```

INTERNALS

DATA

reg

data declaration (c)
```
int reg = 0;
```

data access from pseudo_code
```
(da) cyclic.reg IS USED BY
  (op) cyclic.timer [R]
```

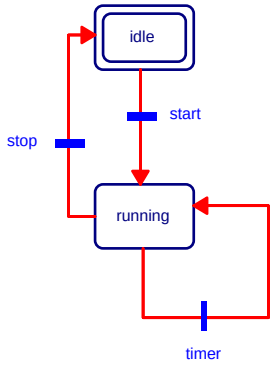
status

data declaration (c)
```
static int status = 0;
```

data access from pseudo_code
```
(da) cyclic.status IS USED BY NONE
```

OBJECT_CONTROL_STRUCTURE

state transition diagram



idle

state assignment (c)
```
status = 0;
```

state test (c)
```
status == 0
```

running

state assignment (c)
```
status = 1;
```

state test (c)
```
status == 1
```

start

transition event
```
start
```

timer

transition event
```
timer
```

stop

transition event
```
stop
```

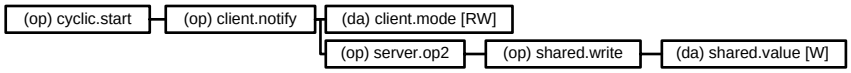
OPERATION_CONTROL_STRUCTURES

OPERATION start IS

used operations
```
client.notify
```

operation code (pseudo)
```
client.notify
```

call tree from pseudo_code



operation code (c)
```
{
  client__notify();
}
```

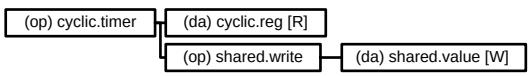
END start

OPERATION timer IS

used operations
```
shared.write
```

operation code (pseudo)
```
shared.write(v => reg);
```

call tree from pseudo_code



operation code (c)
```
{
  shared__write(reg);
}
```

END timer

OPERATION stop IS

operation code (c)

```
{
  cyclic__OBCS_stop();
}
```

END stop

END cyclic

OBJECT shared IS

PASSIVE

PROVIDED_INTERFACE

TYPES

T_value

type attributes (hood)

```
ATTRIBUTES NONE
```

type definition (c)

```
typedef int shared__T_value;
```

OPERATIONS

read

operation declaration (hood)

```
read return T_value*;
```

write

operation declaration (hood)

```
write(v : in T_value);
```

OBJECT_CONTROL_STRUCTURE

constrained operations

```
read CONSTRAINED_BY ROER;
write CONSTRAINED_BY RWER;
```

REQUIRED_INTERFACE

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

INTERNALS

DATA

value

data declaration (c)

```
shared__T_value value = 0;
```

data access from pseudo_code

```
(da) shared.value IS USED BY
  (op) shared.read [R]
  (op) shared.write [W]
```

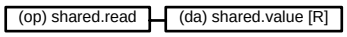
OPERATION_CONTROL_STRUCTURES

OPERATION read IS

operation code (pseudo)

```
return value
```

call tree from pseudo_code



operation code (c)

```
{
  return &value;
}
```

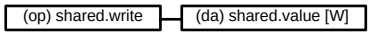
END read

OPERATION write IS

operation code (pseudo)

```
value := v
```

call tree from pseudo_code



operation code (c)

```
{
  value = v;
}
```

END write

END shared

OBJECT server IS

ACTIVE

PROVIDED_INTERFACE

OPERATIONS

op1

operation spec. description (text)
Puts server into mode1.

operation declaration (hood)
```
op1(x : in out shared.T_value; y : in shared.T_value);
```

op2

operation spec. description (text)
Puts server into mode2.

operation declaration (hood)
```
op2(x : in shared.T_value);
```

op3

operation spec. description (text)
Exits mode1 and mode2.

operation declaration (hood)
```
op3;
```

OBJECT_CONTROL_STRUCTURE

constrained operations

```
op1 CONSTRAINED_BY HSER STATE;
op2 CONSTRAINED_BY LSER STATE;
op3 CONSTRAINED_BY ASER STATE;
```

REQUIRED_INTERFACE

```
OBJECT shared;
  TYPES
    T_value;
  CONSTANTS
    NONE
  OPERATION_SETS
    NONE
  OPERATIONS
    write; read;
  EXCEPTIONS
    NONE
OBJECT std;
  TYPES
    int;
  CONSTANTS
    NONE
  OPERATION_SETS
    NONE
  OPERATIONS
    NONE
  EXCEPTIONS
    NONE
```

INTERNALS

CONSTANTS

TRUE

constant definition (c)
```
#define TRUE 1
```

FALSE

constant definition (c)
```
#define FALSE 0
```

DATA

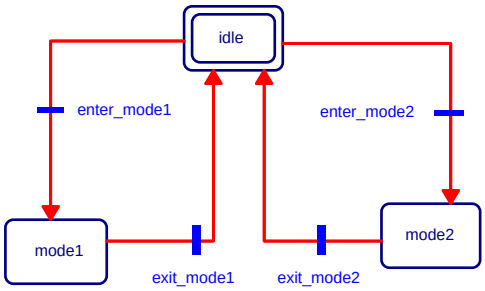
status

data declaration (c)
```
static int status = 0;
```

data access from pseudo_code
```
(da) server.status IS USED BY NONE
```

OBJECT_CONTROL_STRUCTURE

state transition diagram



idle

state description (text)
idle is the initial state, where the server waits for a call to op1 to enter mode 1 or op2 to enter mode 2.

state assignment (c)
```
status = 0;
```

state test (c)
```
status == 0
```

mode1

state description (text)
mode 1 is the destination state after a call to op1

state assignment (c)
```
status = 1;
```

state test (c)
```
status == 1
```

mode2

state description (text)
mode 2 is the destination state after a call to op2

state assignment (c)
```
status = 2;
```

state test (c)
```
status == 2
```

enter_mode1

transition event
```
op1
```

trans description (text)
This transition occurs when receiving a call to op1 while being in idle state.

trans condition (c)
```
TRUE
```

enter_mode2

transition event
```
op2
```

trans description (text)
This transition occurs when receiving a call to op2 while being in idle state.

trans condition (c)
```
TRUE
```

exit_mode1

transition event
```
op3
```

trans description (text)
This transition occurs when receiving a call to op3 while being in mode 1 state.

trans condition (c)
```
TRUE
```

exit_mode2

transition event
```
op3
```

trans description (text)
This transition occurs when receiving a call to op3 while being in mode 2 state.

trans condition (c)
```
TRUE
```

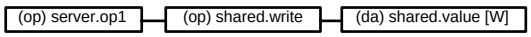
OPERATION_CONTROL_STRUCTURES

OPERATION op1 IS

used operations
```
shared.write
```

operation code (pseudo)
```
shared.write(v => x)
```

call tree from pseudo_code



operation code (c)
```
{
  shared__write(x);
}
```

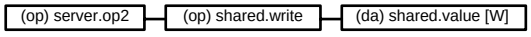
END op1

OPERATION op2 IS

used operations
```
shared.write
```

operation code (pseudo)
```
shared.write(v => x)
```

call tree from pseudo_code



operation code (c)
```
{
  shared__write(x);
}
```

END op2

OPERATION op3 IS

used operations
```
shared.read
```

operation code (pseudo)
```
tmp := shared.read
```

call tree from pseudo_code

operation code (c)

```
{
  shared__T_value* tmp;
  tmp = (shared__T_value*)shared__read;
}
```

END op3

END server