Appendix: Illustration of the SeMo SW development flow for the experiment of section 7.2

In this experiment of cooperation scenario, heterogeneous multiple robots in Table 1 cooperate to find all colored papers in a given grid area. There are two teams: MasterTeam and SlaveTeam. The most powerful robot (”iRobot Create”) is designated as a master robot, and the other robots are grouped into a slave team. All robots first move from the start point (0,0) to the target point (5,5). After reaching the target point, the slave team performs the same action: searching the colored papers. If it detects a colored paper that has not been found yet, it notifies the color to the master robot. Otherwise, it ignores the colored paper and keeps going. Until the slave team finds all colored papers, the master team waits. When the slave team finds all of the colored papers, the master and slave team return to the starting point.

Table 1: Target platforms

	TI Evalbot	**NXT Lego**	**iRobot Create**
OS	uC-OS3	NXT-OSEK	Linux (Ubuntu)
Processor	LM3S9B92 (80MHz)	Atmel 32-bit ARM (48MHz)	ARM Cortex-A7 (quad, 900MHz)
Memory	96KB	64KB	1GB
Extension	Arduino UNO (for sensors)	-	Raspberry Pi 2 (for controller)

Mission Specification

<table>
<tr><th>Plan / Mode</th><th>Listen Plan</th><th>Report Plan</th><th>Action Plan</th></tr>
<tr><td>Remote Control</td><td rowspan="3">Receive commands from operator</td><td>Broadcast commands</td><td>Move by command</td></tr>
<tr><td>Autonomous Move</td><td rowspan="2">Send its location, sensor values</td><td>Move to specific point avoiding obstacles</td></tr>
<tr><td>Autonomous Return</td><td>Move to starting point avoiding obstacles</td></tr>
<tr><td>Wait</td><td>Receive status of SlaveTeam</td><td>Forward status of SlaveTeam to operator</td><td>Stand by</td></tr>
</table>

Figure 1: Specification of behaviors by robot team ”MasterTeam”

Fig. 1 illustrates what tasks to be performed in each combination of mode and plan for the $MasterTeam$. It has four different modes of operation and three plans are running concurrently. In the SeMo framework, we first write a mission script for behavior specification as follows.

```
{
  MasterTeam: iRobotCreate irobot
  SlaveTeam: TIEvalbot ti, NXTLego nxt
}

MasterTeam.Listen.ReceiveCmd{
  receive(USER, USER.RC_CMD)
  if(USER.RC_CMD == "RC_MODE")
    throw CHANGE_RC_MODE
```

```
  else if(USER.RC_CMD == "AUTO_MODE")
    throw CHANGE_MOVE_MODE
  else if(USER.RC_MD == "RETURN_MODE")
    throw CHANGE_RETURN_MODE
} repeat(2 SEC)

MasterTeam.Listen.ReportStatus{
  receive(USER, USER.RC_CMD)
  receive(SlaveTeam, SlaveTeam.COLOR)
  receive(SlaveTeam, SlaveTeam.LOCATION)
  if(USER.RC_CMD == "RC_MODE")
    throw CHANGE_RC_MODE
  else if(USER.RC_MD == "RETURN_MODE")
    throw CHANGE_RETURN_MODE
} repeat(2 SEC)

MasterTeam.Report.SendDefault{
  send(USER, MasterTeam.DISTANCE)
  send(USER, MasterTeam.CAMERA)
  send(USER, MasterTeam.LOCATION)
} repeat(2 SEC)

MasterTeam.Report.Forward{
  send(SlaveTeam, USER.RC_CMD)
}

MasterTeam.Report.ForwardStatus{
  send(USER, SlaveTeam.COLOR)
  send(USER, SlaveTeam.LOCATION)
} repeat(2 SEC)

MasterTeam.Action.AutonomousReturn{
  move("0,0")
  if(LOCATION == "0,0"){
    throw CHANGE_FINISH
  }
} repeat(LOCATION != "0,0")

MasterTeam.Action.AutonomousMove{
  move("5,5")
  if(LOCATION == "5,5")
    throw CHANGE_WAIT_MODE
} repeat(LOCATION != "5,5")

MasterTeam.Action.RemoteControl{
  process(USER.RC_CMD)
} repeat()

MasterTeam.Action.Wait{
  standby()
} repeat()

#MODE Definition
MasterTeam.RC_MODE{
  set(Listen, ReceiveCmd)
  set(Report, Forward)
```

```
    set(Action, RemoteControl)
}
MasterTeam.AUTO_MODE{
    set(Listen, ReceiveCmd)
    set(Report, SendDefault)
    set(Action, AutonomousMove)
}
MasterTeam.WAIT_MODE{
    set(Listen, ReportStatus)
    set(Report, ForwardStatus)
    set(Action, Wait)
}
MasterTeam.RETURN_MODE{
    set(Listen, ReceiveCmd)
    set(Report, SendDefault)
    set(Action, AutonomousReturn)
}
MasterTeam.FINISH{
    set(Listen, OFF)
    set(Report, OFF)
    set(Action, OFF)
}

#main
MasterTeam.main{
    case(AUTO_MODE):
        catch(CHANGE_RC_MODE): mode = RC_MODE
        catch(CHANGE_WAIT_MODE): mode = WAIT_MODE
        catch(CHANGE_RETURN_MODE): mode = RETURN_MODE
    case(RC_MODE):
        catch(CHANGE_MOVE_MODE): mode = AUTO_MODE
        catch(CHANGE_WAIT_MODE): mode = WAIT_MODE
        catch(CHANGE_RETURN_MODE): mode = RETURN_MODE
    case(WAIT_MODE):
        catch(CHANGE_RC_MODE): mode = RC_MODE
        catch(CHANGE_RETURN_MODE): mode = RETURN_MODE
    case(RETURN_MODE):
        catch(CHANGE_MOVE_MODE): mode = AUTO_MODE
        catch(CHANGE_RC_MODE): mode = RC_MODE
        catch(CHANGE_WAIT_MODE): mode = WAIT_MODE
        catch(CHANGE_FINISH): mode = FINISH
    default: mode = AUTO_MODE
}

#------------------------------- SlaveTeam -------------------------------
SlaveTeam.Listen.ReceiveCmd{
    receive(MasterTeam, USER.RC_CMD)
    if(USER.RC_CMD == "RC_MODE")
        throw CHANGE_RC_MODE
    else if(USER.RC_CMD == "AUTO_MODE")
        throw CHANGE_MOVE_MODE
} repeat()

SlaveTeam.Listen.CheckStatus{
    receive(SlaveTeam, SlaveTeam.COLOR)
} repeat()
```

```
SlaveTeam.Report.ShareStatus{
  send(MasterTeam, SlaveTeam.COLOR)
  send(MasterTeam, SlaveTeam.LOCATION)
  if(COLOR == "RGB")
    throw CHANGE_RETURN_MODE
} repeat(COLOR != "RGB")

SlaveTeam.Action.AutonomousMove{
  move("5,5")
  if(LOCATION == "5,5")
    throw CHANGE_SEARCH_MODE
} repeat(LOCATION != "5,5")

SlaveTeam.Action.Search{
  search(colored_paper)
} repeat(COLOR != "RGB")

SlaveTeam.Action.RemoteControl{
  process(USER.RC_CMD)
} repeat()

SlaveTeam.Action.AutonomousReturn{
  move("0,0")
  if(LOCATION == "0,0"){
    throw CHANGE_FINISH
  }
} repeat(LOCATION != "0,0")

SlaveTeam.AUTO_MODE{
  set(Listen, ReceiveCmd)
  set(Action, AutonomousMove)
}
SlaveTeam.SEARCH_MODE{
  set(Listen, CheckStatus)
  set(Report, ShareStatus)
  set(Action, Search)
}
SlaveTeam.RC_MODE{
  set(Listen, ReceiveCmd)
  set(Report, OFF)
  set(Action, RemoteControl)
}
SlaveTeam.RETURN_MODE{
  set(Listen, ReceiveCmd)
  set(Report, OFF)
  set(Action, AutonomousReturn)
}
SlaveTeam.FINISH{
  set(Listen, OFF)
  set(Report, OFF)
  set(Action, OFF)
}

SlaveTeam.main{
  case(AUTO_MODE):
```

```
178       catch(CHANGE_SEARCH_MODE): mode = SEARCH_MODE
179       catch(CHANGE_RC_MODE): mode = RC_MODE
180     case(RC_MODE):
181       catch(CHANGE_MOVE_MODE): mode = AUTO_MODE
182     case(SEARCH_MODE):
183       catch(CHANGE_RETURN_MODE): mode = RETURN_MODE
184     case(RETURN_MODE):
185       catch(CHANGE_RC_MODE): mode = RC_MODE
186       catch(CHANGE_FINISH): mode = FINISH
187     default: mode = AUTO_MODE
188   }
```

In lines 1-5, we specify how two teams are composed; $MasterTeam$ has a single "iRobotCreate" robot and $SlaveTeam$ consists of a "TIEvalbot" and a "NXTLego". Lines 8-109 describe the behavior of $MasterTeam$ and lines 110-188 show the behavior of $SlaveTeam$. We'll focus on the $MasterTeam$'s behavior.

In Fig.1, each cell represents a verbal description of a composite service that is described in the script language in lines 6-60. A service such as move or standby used in the composite service is a basic service and is assumed to be given in the database. The values such as distance, camera, and location are also predefined values that the robot can use. The script editor helps the user to know which values and services are available in each robot. Lines 63-87 present which actions will be taken for each mode of operation, which corresponds to each row of Fig.1. In each mode, three plans are running concurrently with the specified composite service. By default, the $FINISH$ mode is defined to finish all plans, which is written in lines 83-87.

The main loop of lines 90-108 depicts the mode transition based on the event caught by $catch$ phrase during the execution of the current mode. The behavior of $SlaveTeam$ is written similarly to that of $MasterTeam$.

Strategy Description

To fill the large abstraction gap between mission specification and model-based task graph specification, the strategy description is needed. The following shows a part of the strategy description file for $MasterTeam$.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Strategy name="MasterTeam" xmlns="...">
    <ServiceDetailInfo>
        <Service plan_name="Action" name="move">
            <TargetRobot name="iRobotCreate" nickName="irobot" ea="1"/>
            <ActionInfo>
                <Action name="CheckGoal" initial="Yes" actionId="0">
                    <ExecuteAlgorithm name="Localization">
                        <Parameter name="currentLocation" type="metric" dataType=
                            "consume"/>
                    </ExecuteAlgorithm>
                    <Parameter name="goalLocation" type="metric" dataType="
                        internal"/>
                    <TransitionInfo>
                        <Transition srcId="0" dstId="1">
                            <Condition cond="currentLocation==goalLocation"/>
                        </Transition>
                        <Transition srcId="0" dstId="5">
                            <Condition cond="currentLocation!=goalLocation"/>
                        </Transition>
                    </TransitionInfo>
                </Action>
                <Action name="CheckObstacle" actionId="1">
                    <CheckValueOutput name="Ultrasonic">
                        <Parameter type="integer" name="distance" dataType="
                            consume"/>
```

```
                    </CheckValueOutput>
                    <TransitionInfo>
                        <Transition srcId=”1” dstId=”2”>
                            <Condition cond=”distance&gt;20”/>
                        </Transition>
                        <Transition srcId=”1” dstId=”3”>
                            <Condition cond=”distance&lt;20”/>
                        </Transition>
                    </TransitionInfo>
                </Action>
                <Action name=”MoveRandom” actionId=”2”>
                    <ExecuteActuator name=”Move”>
                        <Parameter type=”integer” name=”cmd” value=”Random”
                            dataType=”produce”/>
                        <Parameter type=”MoveCmd” name=”status” dataType=”consume
                            ” />
                    </ExecuteActuator>
                    <TransitionInfo>
                        <Transition srcId=”2” dstId=”4” />
                    </TransitionInfo>
                </Action>
                <Action name=”AvoidObstacle” actionId=”3”>
                    <ExecuteActuator name=”Move”>
                        <Parameter type=”integer” name=”cmd” value=”Turn”
                            dataType=”produce”/>
                        <Parameter type=”MoveCmd” name=”status” dataType=”consume
                            ” />
                    </ExecuteActuator>
                    <TransitionInfo>
                        <Transition srcId=”3” dstId=”4” />
                    </TransitionInfo>
                </Action>
                <Action name=”UpdatePosition” actionId=”4”>
                    <ExecuteAlgorithm name=”Localization”>
                        <Parameter type=”MoveCmd” name=”status” dataType=”produce
                            ” />
                    </ExecuteAlgorithm>
                    <TransitionInfo>
                        <Transition srcId=”4” dstId=”0” />
                    </TransitionInfo>
                </Action>
                <Action name=”Finish” final=”Yes” actionId=”5”>
                    <Parameter value=”1” type=”integer” name=”result” dataType=”
                        internal”/>
                    <TransitionInfo>
                        <Transition srcId=”5” dstId=”0”/>
                    </TransitionInfo>
                </Action>
            </ActionInfo>
        </Service>    <!-- ellipsis -->
    </ServiceDetailInfo>
    <NonFunctionalInfo>
        <BatteryRequirement>
            <Condition cond=”Battery&lt;30”>
                <Sensor name=”Camera” periodLevel=”2”/>
                <Sensor name=”Ultrasonic” periodLevel=”2”/>
```

```
74                  <Actuator name="Move" speedLevel="2"/>
75              </Condition>
76          </BatteryRequirement>
77      </NonFunctionalInfo>
78  </Strategy>
```

Lines 4-66 describe the $move$ service defined in the $Action$ plan. First, we specify which robot the service description is applied as shown in line 5. The $move$ service is described by a textual specification of a finite state machine (FSM) that consists of six fine-grain services, called *actions*. This FSM will be included in the control task when a task graph is automatically synthesized. Each action is given an identifier (*id*) along with its name. Each action can execute an algorithm, execute an actuator, or receive an output value from a sensor or an algorithm. For instance, on lines 7-10, the $currentLocation$ value is taken from an algorithm called *Localization*. And a transition is triggered by comparing it with the internal variable called $goalLocation$.

Also, non-functional requirements can be specified in the strategy description file. In this example, we add an adaptive resource management policy to save the battery energy. In lines 70-76, if the remaining battery energy is lower than 30, the period of *Camera* and *Ultrasonic* sensor is reduced to level 2, and the speed of *Move* actuator is reduced to level 2.

Task-graph Specification

Based on the mission script and strategy description file, the task graphs are automatically generated. The behavior of each robot is specified by a task graph as shown in Fig.2. From the team formation information specified by the mission script, a task graph for each robot is created. For the communication between robots, a library task is added for the management of shared information. The sensor tasks used by a robot are instantiated by referring to the data to be transmitted and the parameter defined in the strategy stage. In the case of algorithm and actuator tasks, they are added according to the service specified in the mission script and strategy description file. For example, the mission script says that $MasterTeam$ transmits the values of distance, camera, and location to the user. Thus, ultrasonic sensor task, camera task, and localization algorithm task are synthesized in the task graph. Note that a task may be expanded as a task sub-graph in case the sub-graph is registered in the database as a composite service that the task performs.

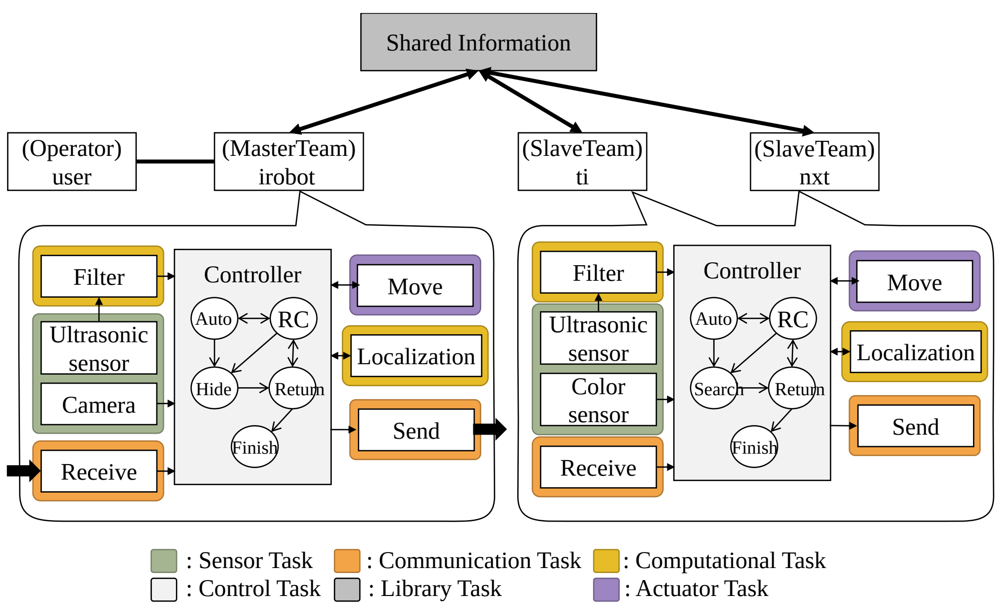

Figure 2: Task graph specification of the example

The most challenging is the synthesis of the control task that is generated as a hierarchical FSM based on the mission specification and strategy description. Fig.3 illustrates how the mode and plan information in the mission script is translated into a state transition diagram in the top-level FSM. Fine-grained services defined in the strategy description file are translated into a bottom-level FSM. In this example, the *move* service is refined into six actions, which are represented by six states in the FSM. In each state, values are received from the sensor and algorithm task, and an algorithm or actuator task is executed.

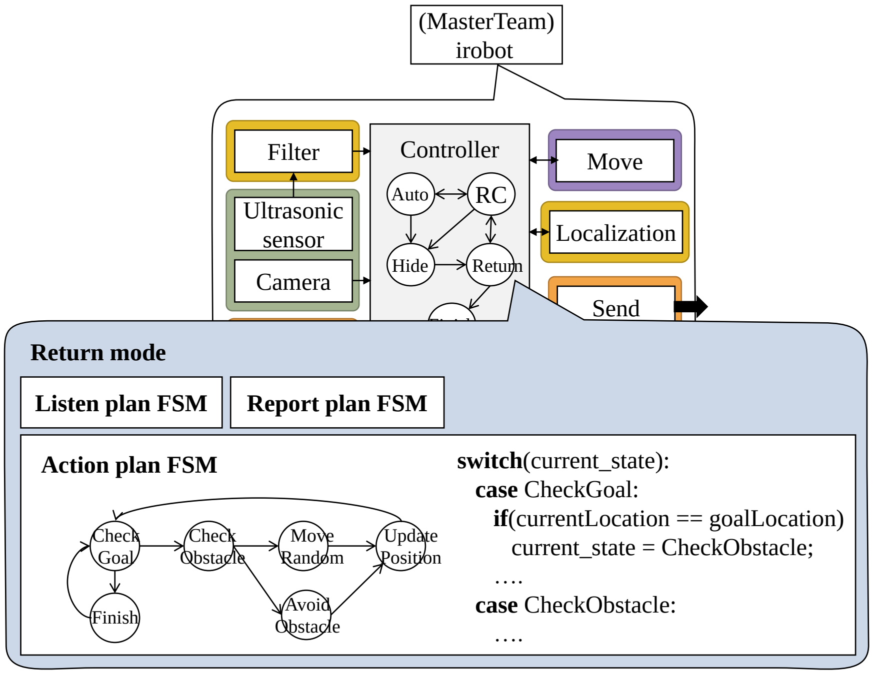

Figure 3: Control task specification for the example

Robot Code Generation

The code generation engine generates the target code from the task graph of each robot. As mentioned in the paper, code generation flow consists of two main steps: platform independent code generation and platform dependent code generation.

In the first step, we construct data structures for tasks, channels, and libraries and the skeleton of the main schedule code that creates the task threads. Fig.4 illustrates an example data structure for a task. Since the data types for the thread, mutex, and conditional variable depend on the actual SW platform or operating system, we define macros such as MUTEX_TYPE, COND_TYPE, and THREAD_TYPE. These macros will be defined with actual data types depending on the SW platform of the target architecture.

The skeleton of the scheduler code is explained in section 6 of the paper. And inter-task communication method depends on the mapping result. There are two types of inter-task communication: one is the library call, and the other is communication between tasks through the channel.

For a library task, the library wrapper and stub code are generated by the code generation engine. Fig.5 (a) shows the thread routine for a library wrapper thread. It checks whether a request packet arrives at its library channel. If a packet arrives, it parses the packet and checks which service function to call. After it calls the function, it sends a result packet to the caller task if its return type is not void. A separate wrapper thread is generated for each library task.

Fig.5 (b) shows library stub code for a library caller task. When the task calls a library function and the corresponding library task is mapped onto a different platform, a stub function is declared. In the stub function, it packetizes the service request information including the index of the service function and its arguments, and sends the packet to the library wrapper thread of the remote platform.

The communication code for local connection includes implementation of generic APIs such as MQ_SEND/RECEIVE and LIB_SEND/RECEIVE. If a target architecture has global shared memory, communication is implemented via

```
typedef struct{
  unsigned int task_id;
  TASK_TYPE task_type;
  unsigned int period;
  …
  MUTEX_TYPE mutex;
  COND_TYPE cond;
  THREAD_TYPE thread;
} TASK;

TASK tasks[] = {
  {0, TimeDriven, …, MUTEX_INIT_INLINE, … }, … }
```

Figure 4: Data structure for a task

```
void library_wrapper_routine(){
  while(true){
    LIB_RECEIVE(received packet);
    function_index = received packet.function_index;
    switch(function_index){
      case 0:
        LIB_CALL(…);
        if return type is not void
          LIB_SEND(result packet);
        break; …
}
```

(a) Library stub code

```
LIBFUNC (int, function, int arg0, …){
  MUTEX_LOCK;
  LIB_SEND(…);
  if return type is not void
    LIB_RECEIVE(…);
  MUTEX_UNLOCK;
}
```

(b) Library wrapper code

Figure 5: Library (a) wrapper and (b) stub code

shared memory. Otherwise, explicit message passing should be performed for communication, which is the case for distributed platforms.

For remote communication, a communication wrapper task is generated. A part of the communication task is displayed in Fig.6. Since multiple logical channels between tasks share the same physical channel between platforms, the wrapper task has to arbitrate incoming/outgoing data. In our heterogeneous robot platforms, there are two types of remote connection: one is a Bluetooth channel for sharing colored-paper what they find out and remote-control

```
while(true){
  if available data in connection
    receive data;
    if data is for normal channel
      MQ_SEND(…);
    else if data is for library channel
      LIB_SEND(…);
  for all outgoing normal/library channels:
    if data in a normal channel
      MQ_RECEIVE(…)
    else if data in a library channel
      LIB_RECEIVE(…)
    send data;
}
```

Figure 6: The skeleton code of a communication wrapper task

commands from the operator, the other is between TI Evalbot and Arduino UNO board via an I2C interface for sensor connection.

The communication wrapper checks whether data is available on a physical channel. If data is available, it receives data and sends the data to a proper logical channel. And it checks all outgoing channels, and if there are available data in a logical channel, it reads and sends the data to the physical channel. Even though the basic behavior of the wrapper task is simple, there are several implementation issues for stable and reliable communication such as error detection and recovery. Also, the wrapper task should consider the unique characteristics of the physical channel.

The second step is platform dependent code generation that depends on the operating system and the robot hardware platform. Some generic APIs are included in the generated code for portability. So the code generation engine has to generate the target specific code for those APIs. Fig.7 shows an example of target-specific code for thread creation for each target platform. Also, it generates a $makefile$ or appropriate project files to use the software development tool of each target platform.

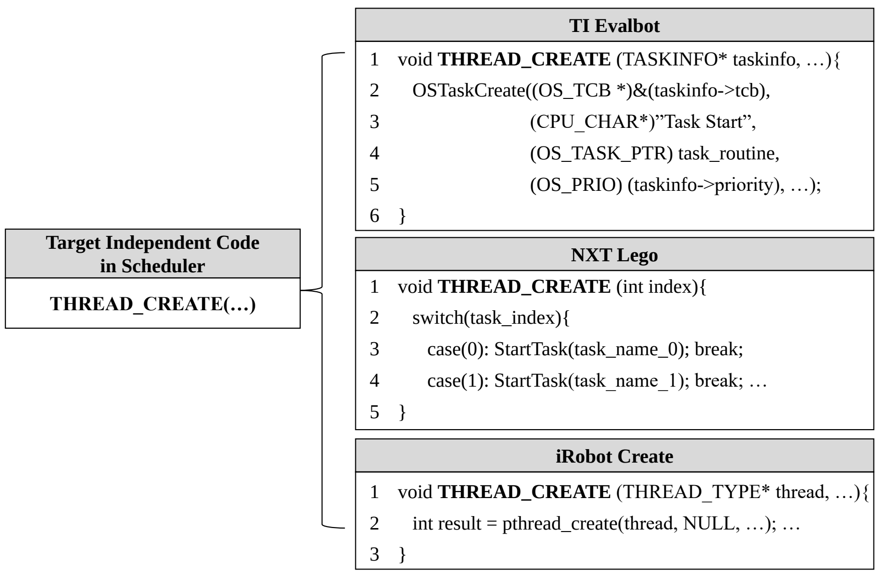

Figure 7: Target specific code for thread creation