# Verification of Real Time Operating System with Model Checking

Toshiaki Aoki
Japan Advanced Institute of Science and Technology

## Background

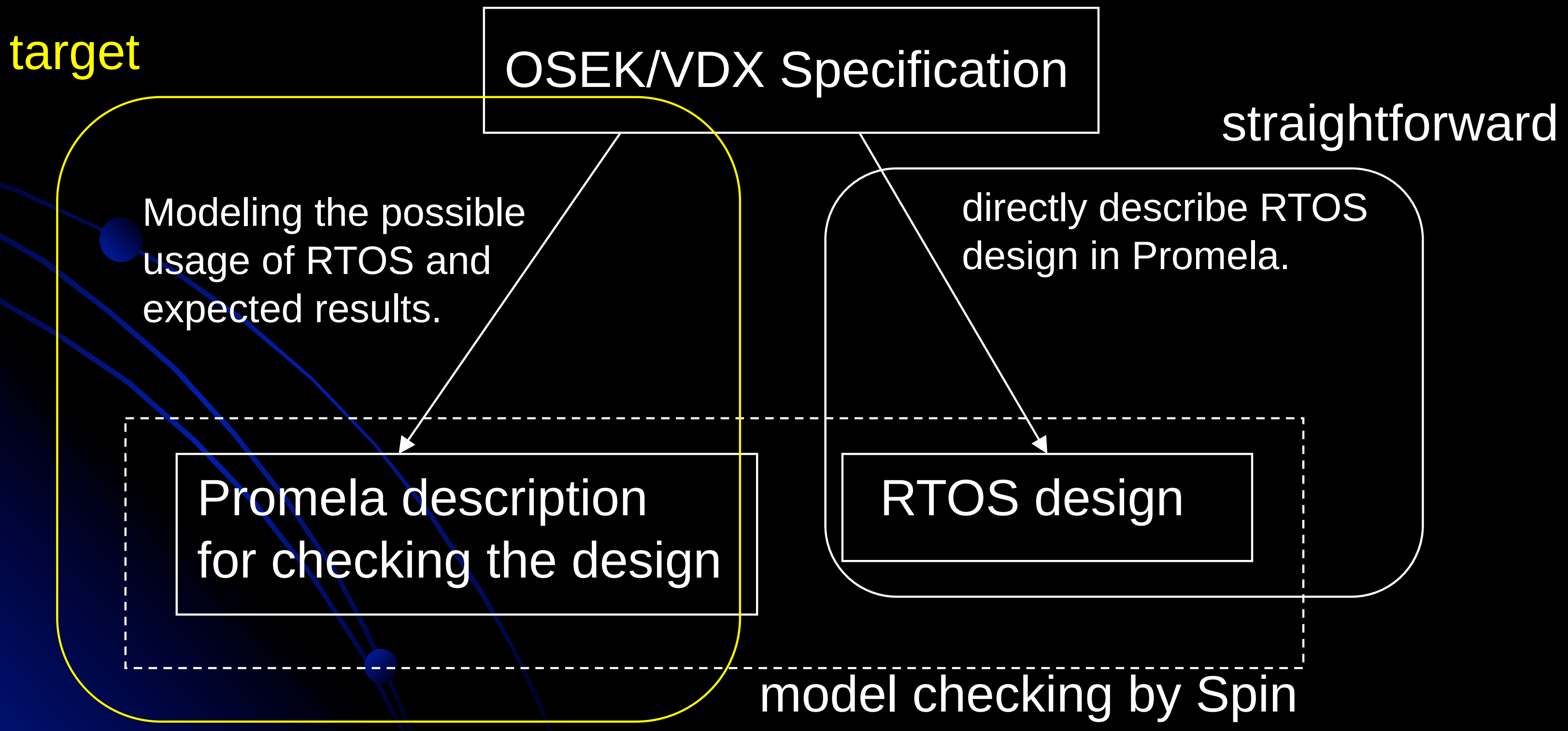
- RTOS (Real-Time Operating Systems) are used for embedded systems.
  - Those RTOS are provided for various platforms by various companies.
- Standards of RTOS are proposed.
  - OSEK/VDX(AUTOSAR OS), μITRON, etc.
- Standards are defined as documents described in natural language.
  - The documents are likely misunderstood.
  - The documents themselves tend to be ambiguous.
- Those may lead to implement RTOS which does not conform to the standards.
  - Such RTOS makes embedded systems unreliable.

## Aim

- We are proposing a method to ensure that RTOS conforms to the standards.
  - Our target is OSEK/VDX RTOS.
- It is very hard to ensure the conformance after the implementation by testing.
  - Test cases to ensure the conformance depend on the inside of the implementation.
  - Describing exhaustive test cases is very hard.
- We ensure the conformance in the design phase of RTOS development.
  - We apply model checking to the design of RTOS to exhaustively check it.
  - We are using Spin model checker.

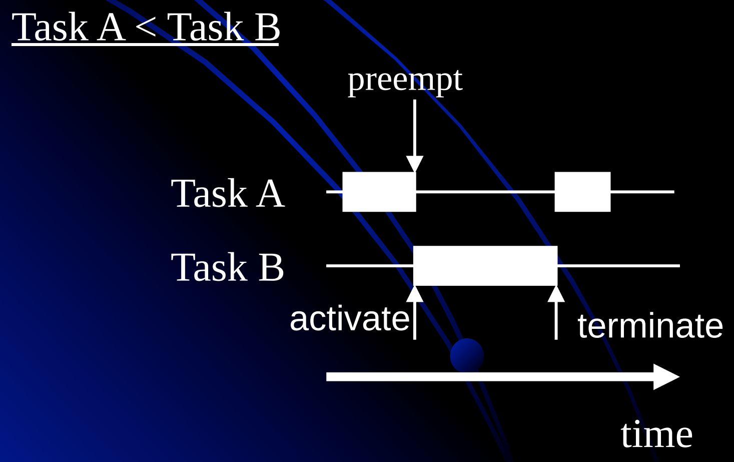
# Approach

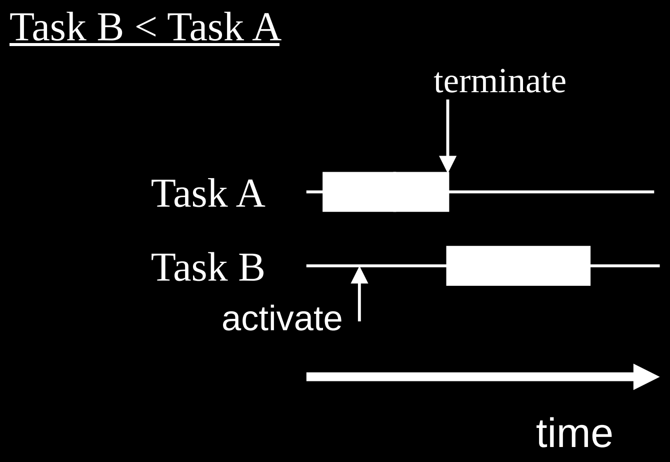
- We directly describe RTOS design in Promela.
  - Promela is the specification language of Spin.
- We need the other Promela descriptions for checking the design.
  - How to use RTOS and expected results are described in them.
  - We obtain such Promela descriptions by modeling the possible usage of OSEK/VDX RTOS and expected results based on the specification.


target
OSEK/VDX Specification
straightforward
Modeling the possible usage of RTOS and expected results.
directly describe RTOS design in Promela.
Promela description for checking the design
RTOS design
model checking by Spin

# OSEK/VDX RTOS

- OSEK/VDX RTOS is used for automotive systems.
  - Currently, its standard is being developed as AUTOSAR OS.
  - The most of the OSEK/VDX standard specification is inherited to AUTOSAR OS.
- OSEK/VDX RTOS deal with preemptive fixed-priority multi-tasks, resources, interrupts, and so on.
  - The resources are similar to lock/unlock primitives with a priority ceiling protocol.
  - Fixed priorities are assigned to the interrupts.

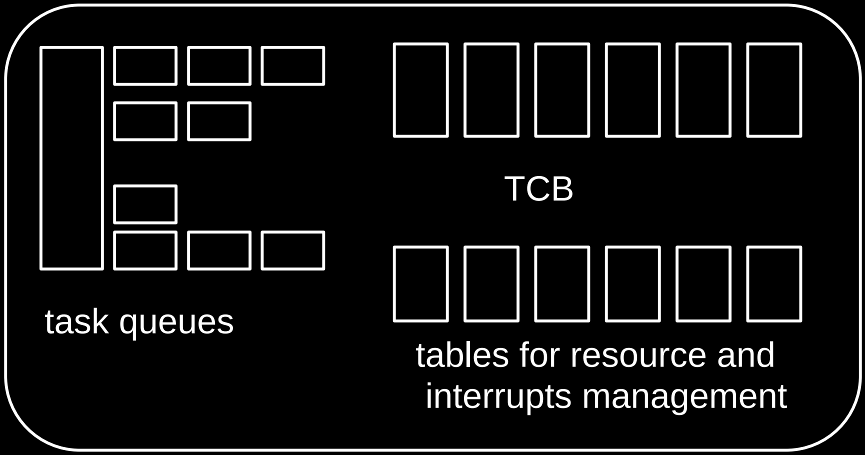




# Design of OSEK/VDX RTOS

- We focus on the scheduler of RTOS.
  - The most of the specification defines how to schedule the tasks.
- It is easy to describe the scheduler in Promela.
  - The scheduler is realized by task queues and tables to keep their information.
  - Those are straightforwardly described in Promela.
    - You can download the Promela description of μITRON RTOS which is similar to OSEK/VDX RTOS in the following URL.
    - http://aoki-www.jaist.ac.jp/~toshiaki/modules/tinyd0/index.php?id=10

The scheduler of OSEK/VDX RTOS

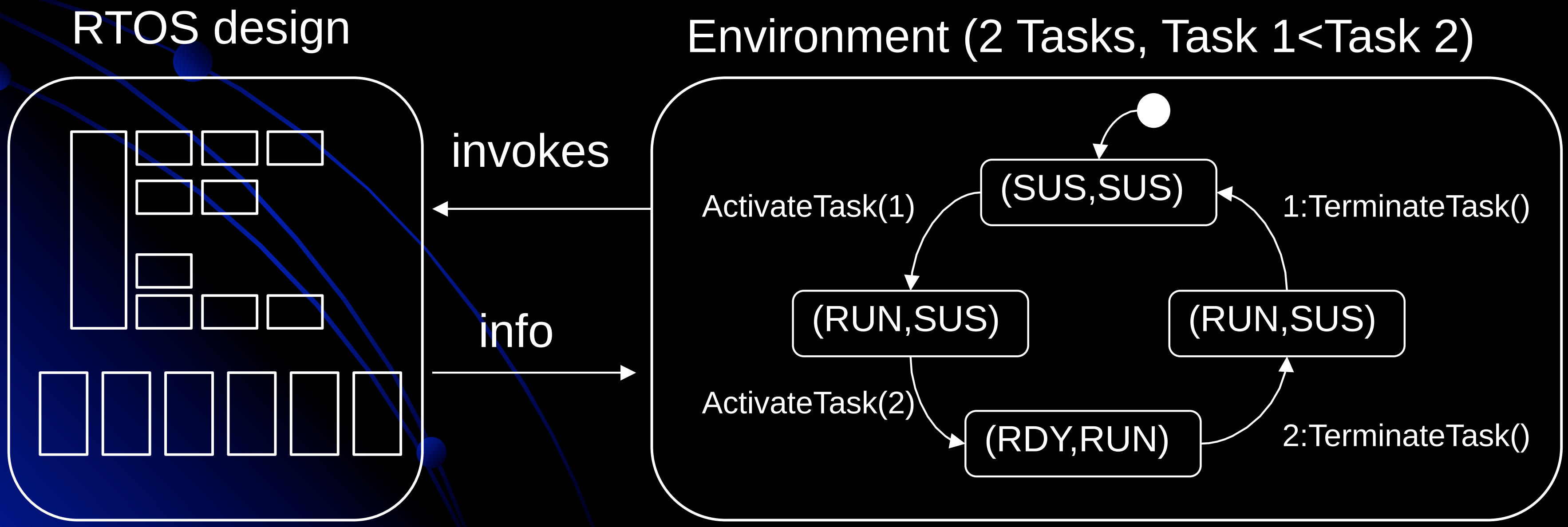


## Design Verification

- RTOS is an open system.
  - RTOS does scheduling of the tasks if it gets stimulus such as system call invocations.
  - RTOS does not do anything if it does not get any stimulus.
- We need the description of the outside of RTOS to verify the RTOS design.
  - The outside consists of an application which invokes the system calls and hardware which causes the interrupts.
- The outside of the verification target is called an *environment*.

# Environment

- The description of an environment consists of
  - invocations of the system calls of RTOS.
    - represented as state transition models with the system calls.
  - expected results of those invocations.
    - represented as assertions assigned to those states.
- Model checking the RTOS design in combination with the description of the environment by Spin.

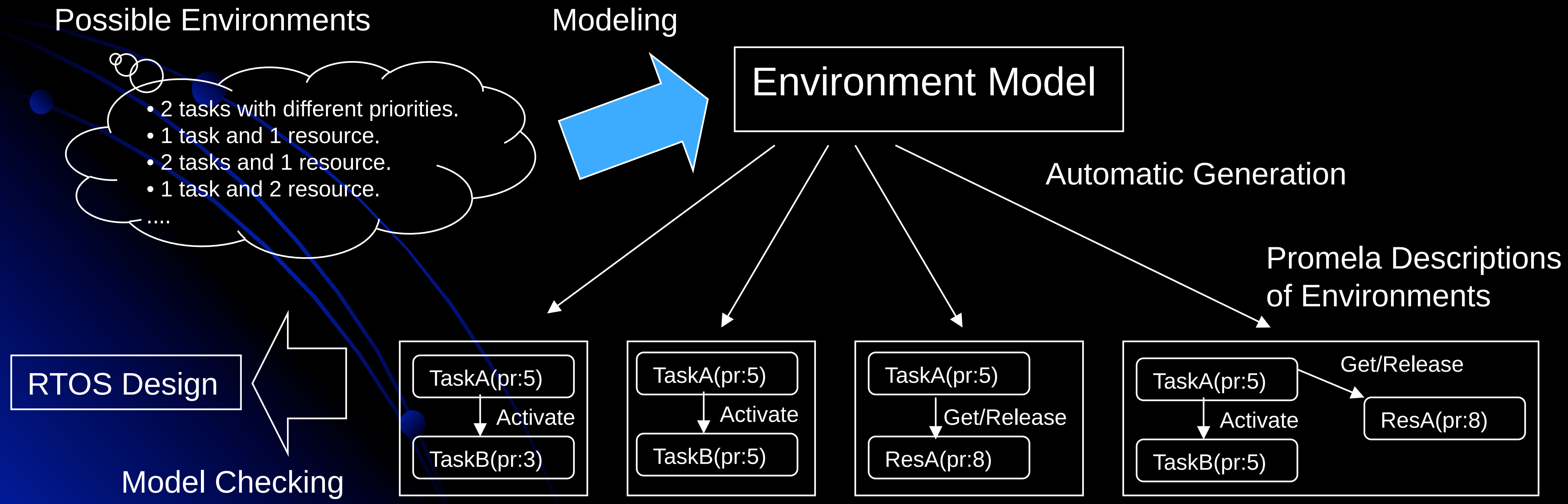


## Approaches to Describe Environments

- Automatic generation by static analysis on programs.
  - BEG (Bandera Environment Generator) [O.Tkachuk ’03]
  - Effective if the programs corresponding to the environment available.
  - Usually not the case in the design stage.
- Invoke all the system calls non-deterministically
  - Universal environment [J.Penix ’00]
  - Able to check all the execution sequences exhaustively.
  - Description of properties becomes complex. State explosion.
- Our approach: Modeling execution sequences in a specific range depending on the properties to check.
  - Ex.) Normal execution sequences, abnormal execution sequences, and interrupt handling ...
  - Description of properties becomes simple. Possible to avoid state explosion.

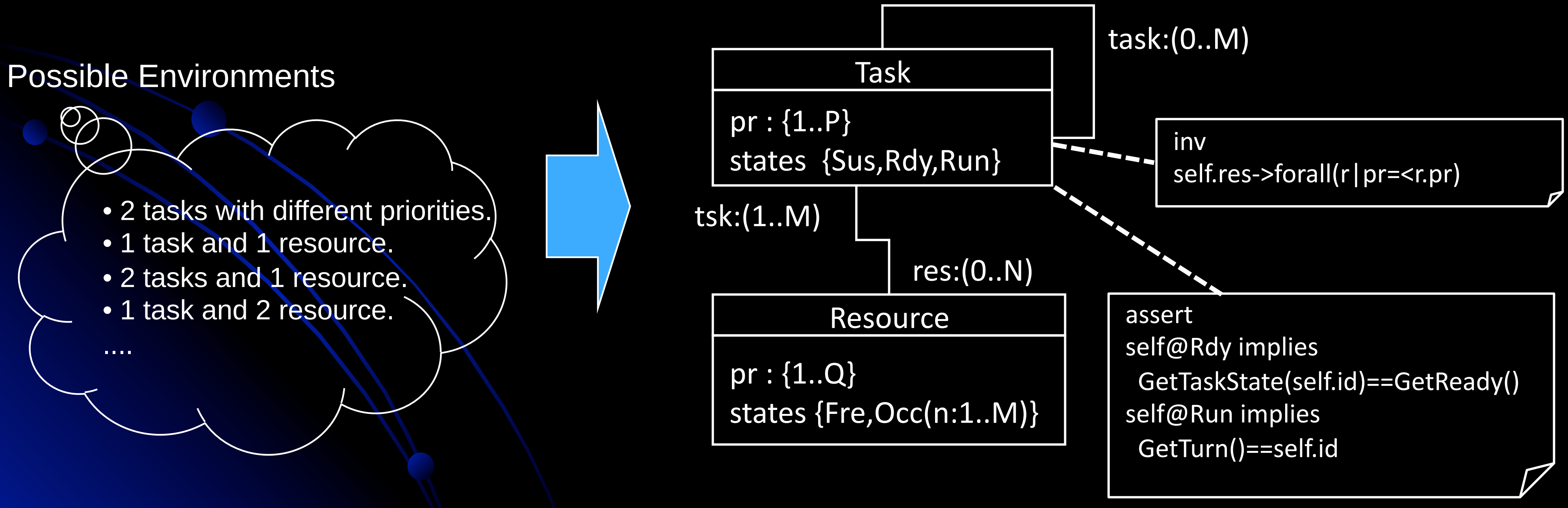
# Environment Modeling

- Various environments can be considered.
  - The number of tasks, the variation of priorities, number of the resources, invocation relations of the system calls, and so on.
  - It is impossible to make all the descriptions of the environments by hand.
  - If all the environments are realized by one Promela description, that may cause state explosion problem.
- We model the variations of the environments, then automatically generate the Promela description of each environment.



# Environment Modeling

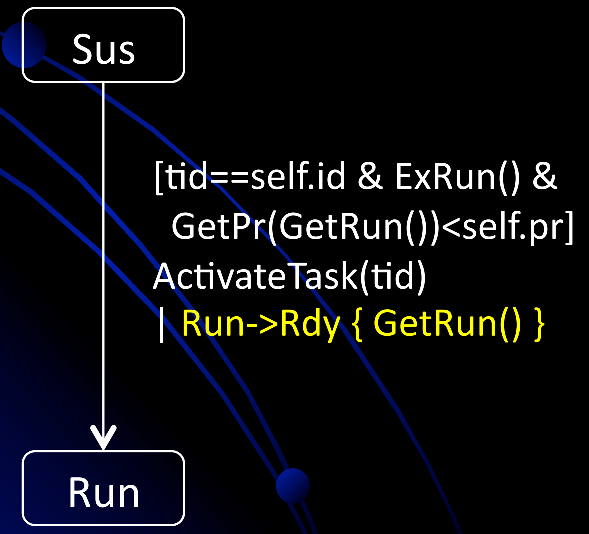
- We model possible structures of the environments using class diagram.
  - The variations of the environments are represented as multiplicities of associations among classes.
  - Constraints on attribute values and multiplicities are described in OCL (Object Constraint Language).



# Environment Modeling

- The possible behavior of the tasks and resources are described using statechart diagram with some extension.
  - Introducing a *derived transition*.
  - The derived transition causes the state transition of the other instances.
    - | S1 -> S2 {ins} causes the state transition of the instance 'ins' from 'S1' to 'S2'.
  - The derived transition makes the statechart model simple.

A transition of task behavior

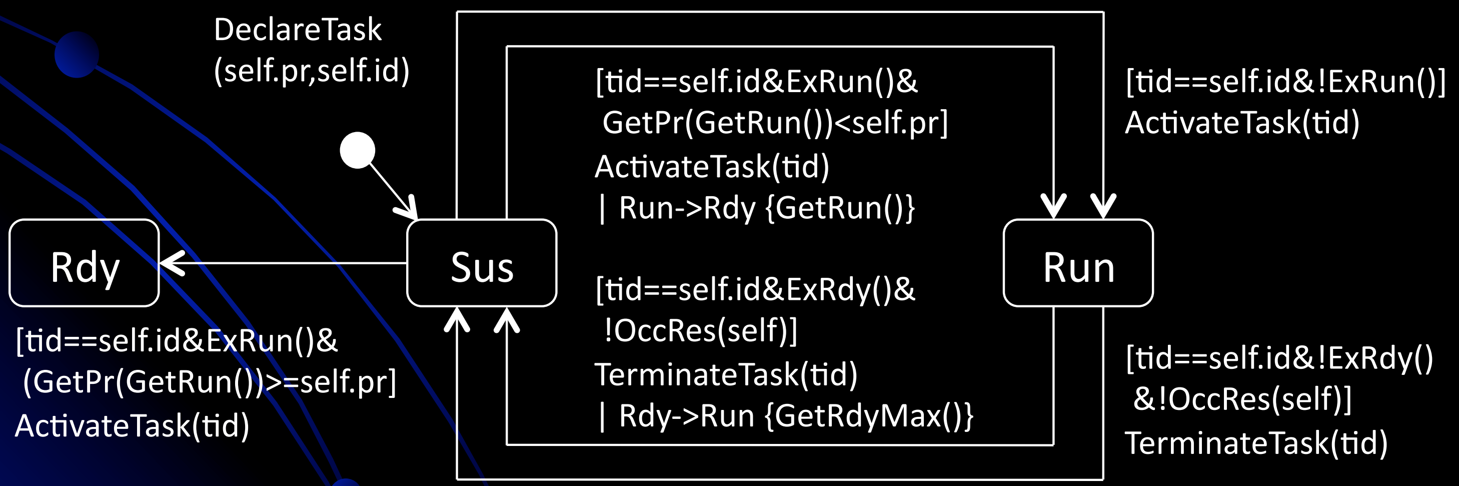


1. Currently, a task is in the suspended state, and there is another task which is in the run state.
   EX) Task 1: Sus, Task 2: Run, ...
2. If the task having a priority which is higher than the running task moves to the run state, the running task moves to the ready state simultaneously.
   EX) Task 1: Run, Task 2: Rdy

# Environment Modeling

- The possible behavior of the tasks and resources are described using statechart diagram with some extension.
  - We assign an assertion to each state.
  - The assertion checks whether an observed state of RTOS is the expected one or not.

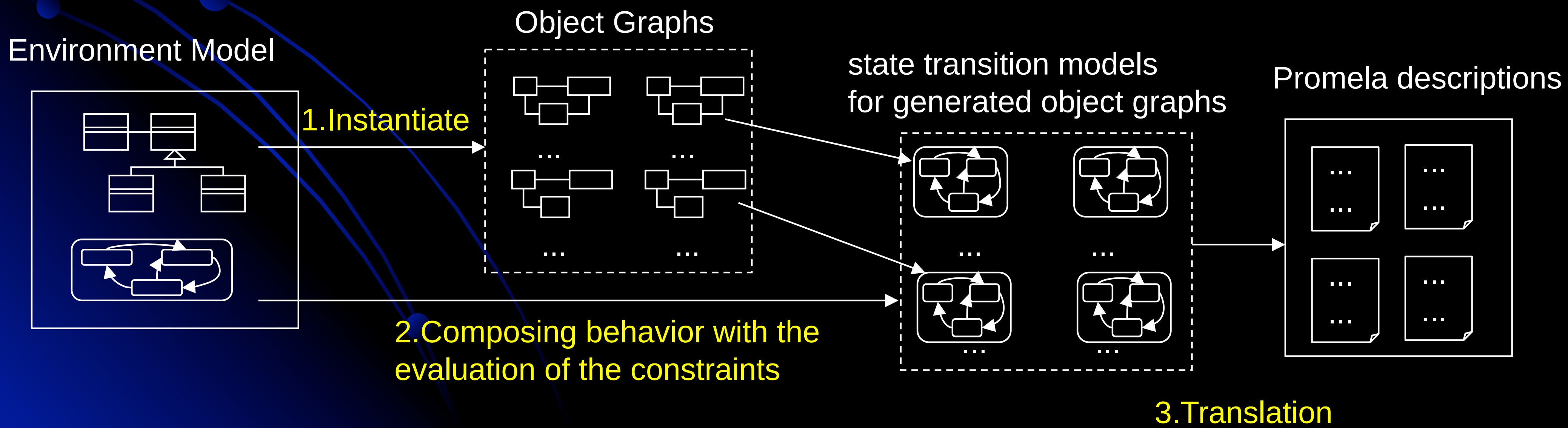
Statechart Model for Task Behavior



DeclareTask (self.pr,self.id)
Rdy
Sus
Run
[tid==self.id&ExRun()& GetPr(GetRun())<self.pr] ActivateTask(tid) | Run->Rdy {GetRun()}
[tid==self.id&!ExRun()] ActivateTask(tid)
[tid==self.id&ExRdy()& !OccRes(self)] TerminateTask(tid) | Rdy->Run {GetRdyMax()}
[tid==self.id&ExRun()& (GetPr(GetRun())>=self.pr] ActivateTask(tid)
[tid==self.id&!ExRdy() &!OccRes(self)] TerminateTask(tid)

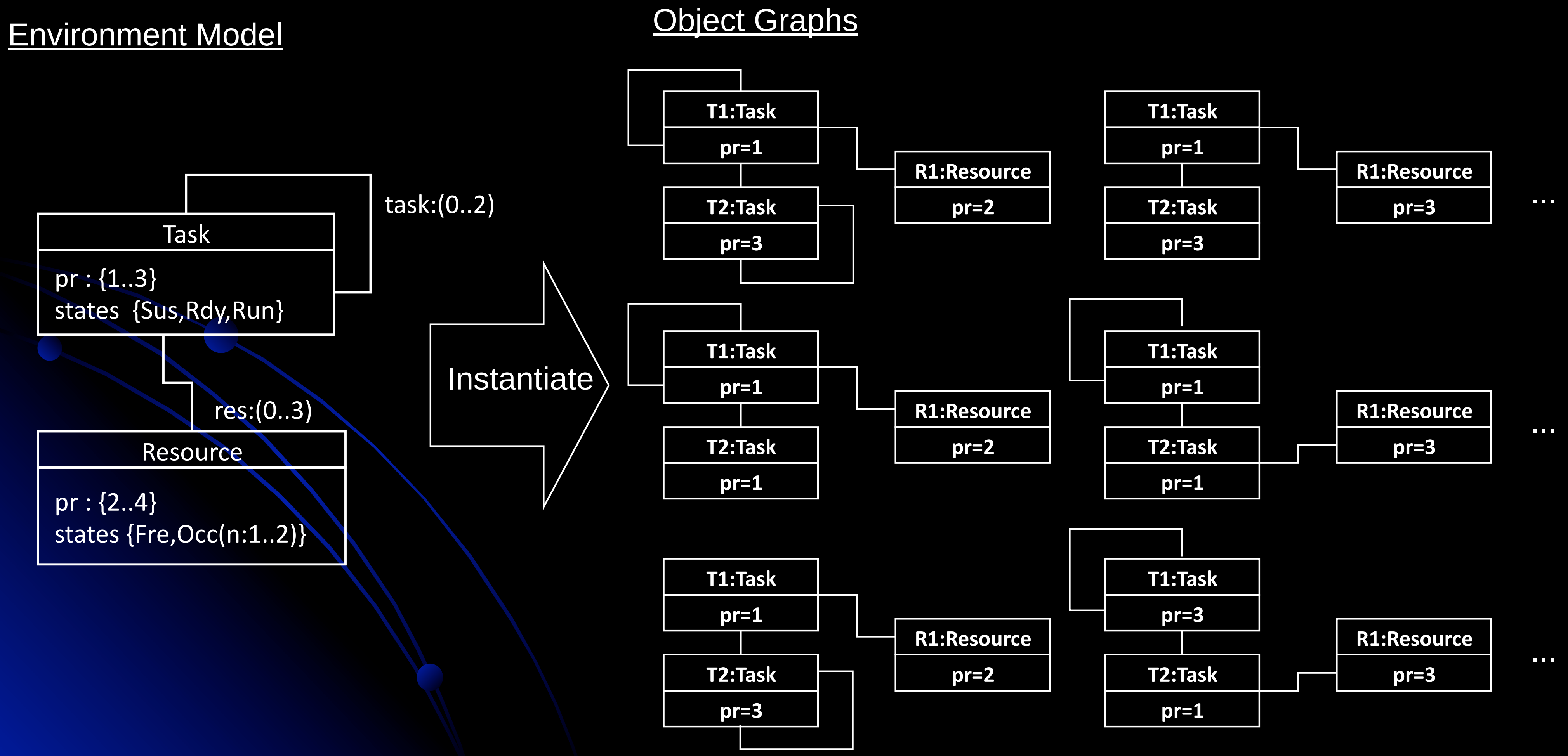
# Generation of Environments

- We automatically generate environments from the environment model.
  - A Promela description of an environment is generated for each of the variations described in the environment model.
- Procedure of the generation.
  1. Generate all the object graphs in a given bound from the class model of the environment model.
  2. Make the state transition models of each of the object graphs based on the statechart model of the environment model.
  3. Translate the state transition models into Promela descriptions.



# Generation of Environments

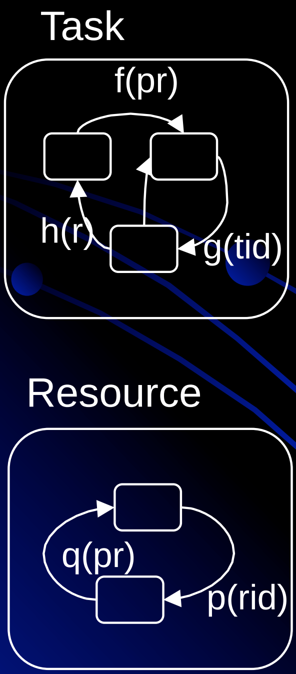
1. Generate all the object graphs in a given bound from the class model of the environment model.



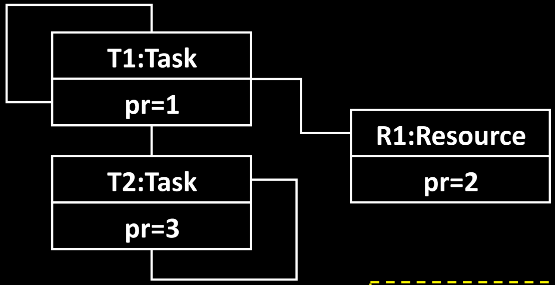
# Generation of Environments

2. Make the state transition models of each of the object graphs based on the statechart model of the environment model.

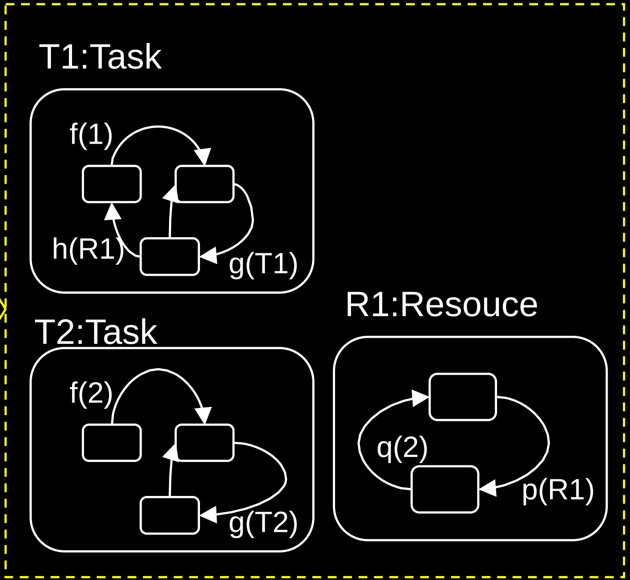
Environment Model



An object graph

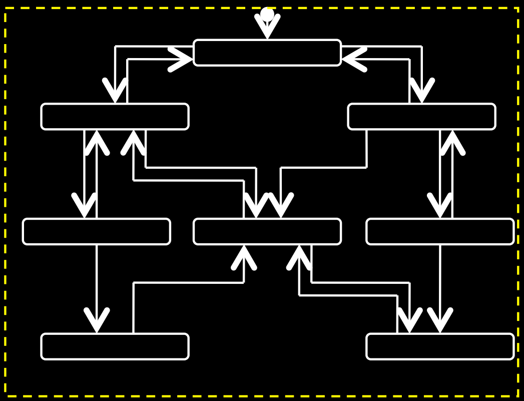


Instantiate state transition models based on the environment model.



Compose the state transition models with the evaluation of the constraints.

Environment

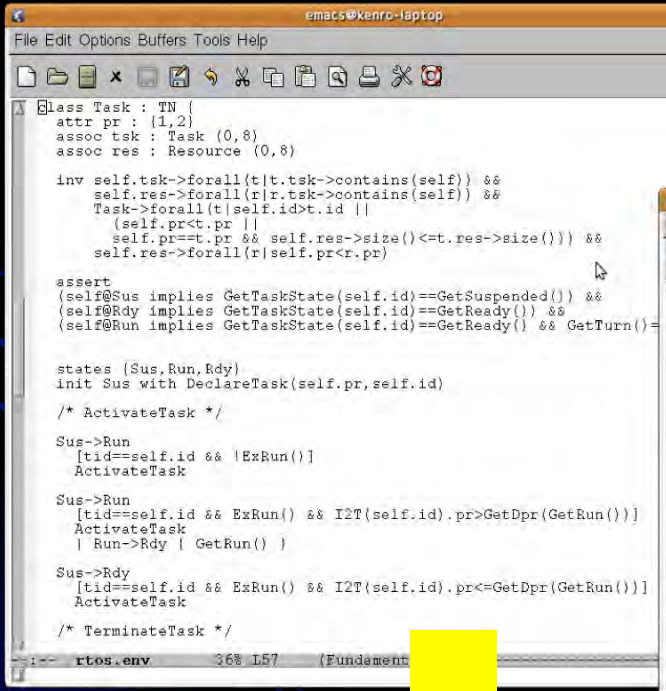


3.We can directly translate the environment to the Promela description.

# Environment Generator

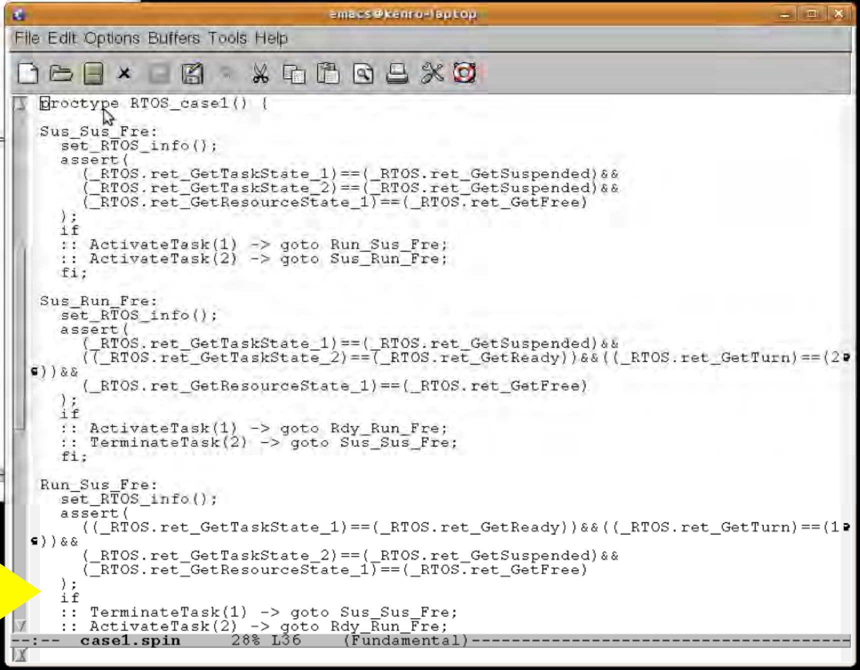
- We have implemented an environment generator.
  - Implemented in SML/NJ.
  - The environment model is described in text.
  - Automatically generates Promela descriptions of all the environments.

Environment model（rtos.env）



```
class Task : TN {
  attr pr : {1,2}
  assoc tsk : Task (0,8)
  assoc res : Resource (0,8)

  inv self.tsk->forall(t|t.tsk->contains(self)) &&
      self.res->forall(r|r.tsk->contains(self)) &&
      Task->forall(t|self.id>t.id ||
        (self.pr<t.pr ||
        self.pr==t.pr && self.res->size()<=t.res->size())) &&
      self.res->forall(r|self.pr<r.pr)

  assert
  (self@Sus implies GetTaskState(self.id)==GetSuspended()) &&
  (self@Rdy implies GetTaskState(self.id)==GetReady()) &&
  (self@Run implies GetTaskState(self.id)==GetReady() && GetTurn()=


  states {Sus,Run,Rdy}
  init Sus with DeclareTask(self.pr,self.id)

  /* ActivateTask */

  Sus->Run
    [tid==self.id && !ExRun()]
    ActivateTask

  Sus->Run
    [tid==self.id && ExRun() && I2T(self.id).pr>GetDpr(GetRun())]
    ActivateTask
    | Run->Rdy { GetRun() }

  Sus->Rdy
    [tid==self.id && ExRun() && I2T(self.id).pr<=GetDpr(GetRun())]
    ActivateTask

  /* TerminateTask */
--:--  rtos.env       36% L57     (Fundament
```

Promela description（case1.spin）



```
proctype RTOS_case1() {

Sus_Sus_Fre:
  set_RTOS_info();
  assert(
    (_RTOS.ret_GetTaskState_1)==(_RTOS.ret_GetSuspended)&&
    (_RTOS.ret_GetTaskState_2)==(_RTOS.ret_GetSuspended)&&
    (_RTOS.ret_GetResourceState_1)==(_RTOS.ret_GetFree)
  );
  if
  :: ActivateTask(1) -> goto Run_Sus_Fre;
  :: ActivateTask(2) -> goto Sus_Run_Fre;
  fi;

Sus_Run_Fre:
  set_RTOS_info();
  assert(
    (_RTOS.ret_GetTaskState_1)==(_RTOS.ret_GetSuspended)&&
    ((_RTOS.ret_GetTaskState_2)==(_RTOS.ret_GetReady))&&((_RTOS.ret_GetTurn)==(2
))&&
    (_RTOS.ret_GetResourceState_1)==(_RTOS.ret_GetFree)
  );
  if
  :: ActivateTask(1) -> goto Rdy_Run_Fre;
  :: TerminateTask(2) -> goto Sus_Sus_Fre;
  fi;

Run_Sus_Fre:
  set_RTOS_info();
  assert(
    ((_RTOS.ret_GetTaskState_1)==(_RTOS.ret_GetReady))&&((_RTOS.ret_GetTurn)==(1
))&&
    (_RTOS.ret_GetTaskState_2)==(_RTOS.ret_GetSuspended)&&
    (_RTOS.ret_GetResourceState_1)==(_RTOS.ret_GetFree)
  );
  if
  :: TerminateTask(1) -> goto Sus_Sus_Fre;
  :: ActivateTask(2) -> goto Rdy_Run_Fre;
--:--  case1.spin      28% L36     (Fundamental)----------------------------------
```

# Generation Results

- We generate various environments for the verification of the RTOS design by our environment generator.
- It takes very long time to check the RTOS design against all the descriptions.
  - Ex) It take 12 hours to check about 1000 descriptions by the following machine.
    - CPU: Core2DUO 2.13 GHz, Memory 2Gbyte.
  - It may take 12*50 = 600h= 25 days (about 1 month).

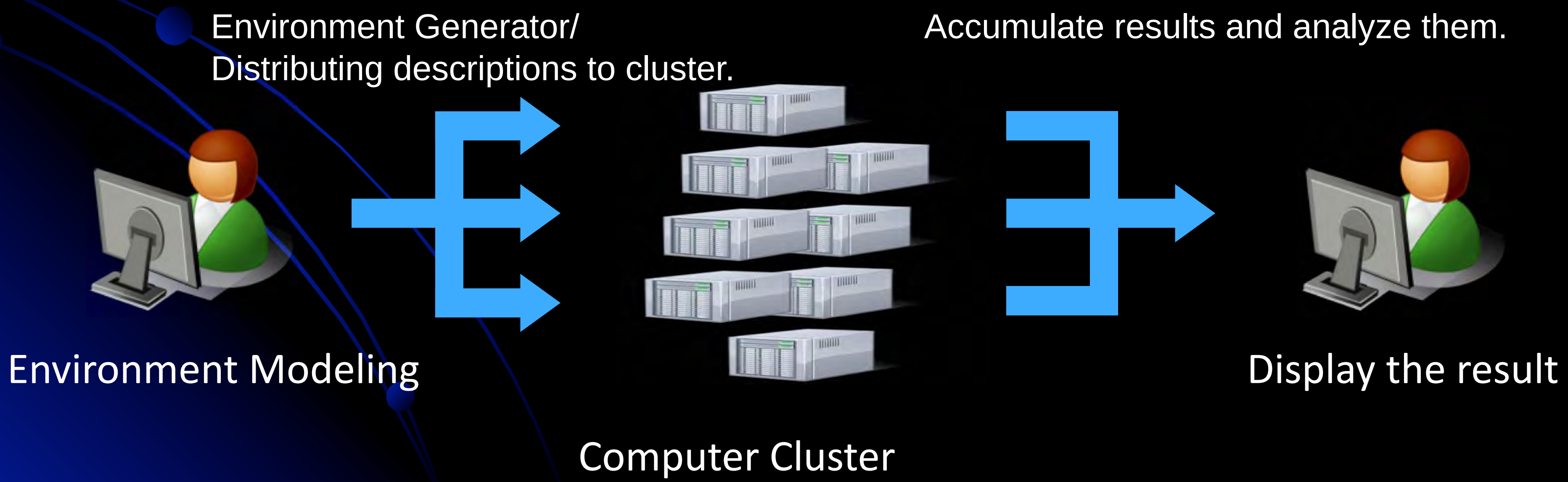
The number of generated Promela descriptions

| R/T | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 0 | 4 (0.0s) | 20 (0.0s) | 140 (0.5s) | 1540 (81.3s) |
| 1 | 8 (0.0s) | 104 (0.2s) | 1496 (31.6s) | 30664 (9.4h) |
| 2 | 12 (0.1s) | 468 (2.6s) | 15132 (56.1m) | N/A |
| 3 | 16 (0.1s) | 1840 (61.4s) | N/A | N/A |

N/A:
It takes too long..

## Verification by Computer Cluster

- We are doing model checking by computer cluster.
    - Check a large number of the Promela descriptions by many workstations.
- Currently, we are using collaborative facilities for verification, named 'SATSUKI' in AIST.
    - We did some experiments.
    - We checked all the description by the following cluster.
        - The number of nodes: 70
        - Sun Fire X4150, Xeon X5260 3.3GHz Dual Core, 8Gbyte Memory.
    - About 6 hours.



# Verification Results

- We found several bugs of the RTOS design by this approach.
  - We first checked the RTOS design by some environments and remove found bugs.
  - Then, we verified it by this approach.
- A found bug.
  - Structure of tasks and resources.
    - Tasks: T1 with the priority 3, T2 with the priority 1.
    - Resources: R1 with the priority 4, R2 with the priority 2.
      - R1 is used by T1 and T2. R2 is used by only T2.
    - This structure is somewhat special.
  - In this case, the priority ceiling protocol was not correctly realized in the RTOS design.
    - TCB was not updated when a task returns a resource for a specific execution sequence.
  - We made a mistake in condition branches .

## Discussion

- Our approach can be regarded as bounded model checking.
  - Variations are described in the class diagram.
  - We instantiate a part of them by bounding the multiplicities and attribute values.

→ It is useful to find bugs in the RTOS design.

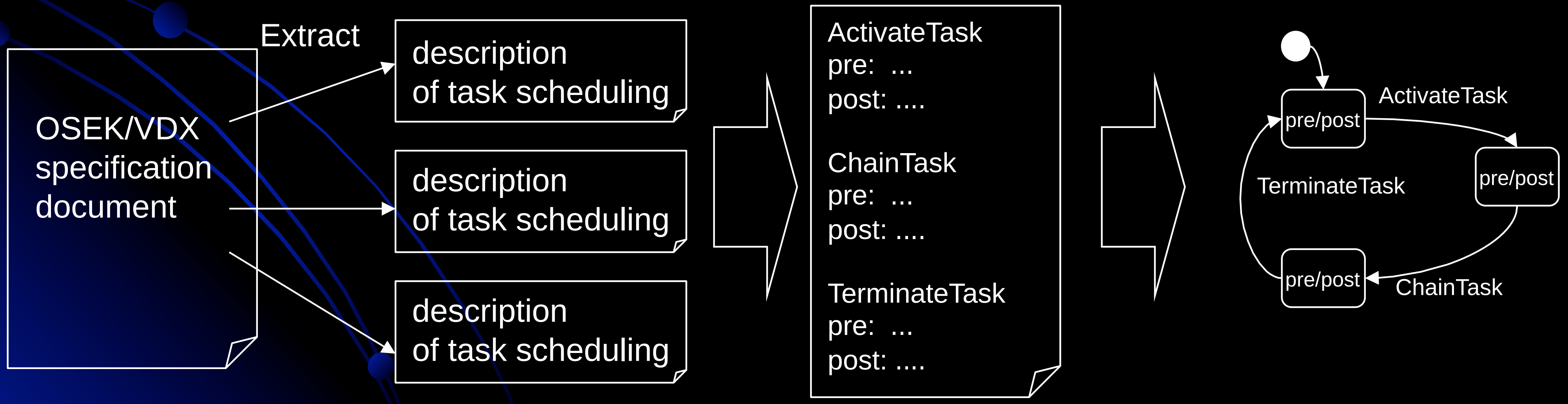
  - Comparing the model checking results with each other help us to find the bugs.
- We need theorem proving to ensure that the RTOS design is correct for all the variations.
  - If we focus on a specific RTOS, the variations may finite.
  - However, state explosion problem may occur because RTOS usually deals with hundreds of tasks.

## Discussion

- Our approach makes it possible to do model checking in parallel.
  - We statically divide state space to be checked according to the variations of the environments.
- We have advantage to searching larger state space of the RTOS design.
  - Parts of the state spaces checked by the generated environments are overlapped.
  - We still have an advantage in the state space.
- Many results are obtained.
  - It is very hard to check all of them precisely.
    - 100,000 checks may return 100,000 counter examples.
  - Conversely, we obtain much information about the checks.
    - Applying statistic methods such as cluster analysis and  machine learning for analyzing the results is meaningful.
    - Visualizing the results allows us to intuitively understand what happens in the RTOS design.

# Discussion

- We carefully made the environment model based on the OSEK/VDX specification document.
  - Extracting descriptions of the tasks scheduling from the document
    - Those descriptions are scattered in the document.
  - We make pre/post conditions of each system calls from those descriptions.
    - The conditions are described in natural language mixed with logical formulas.
  - We construct the statechart model based on the pre/post conditions.
- Formal treatments are needed.
  - Some parts of the document are ambiguous.
  - Removing mistakes made in the transformations of the document.

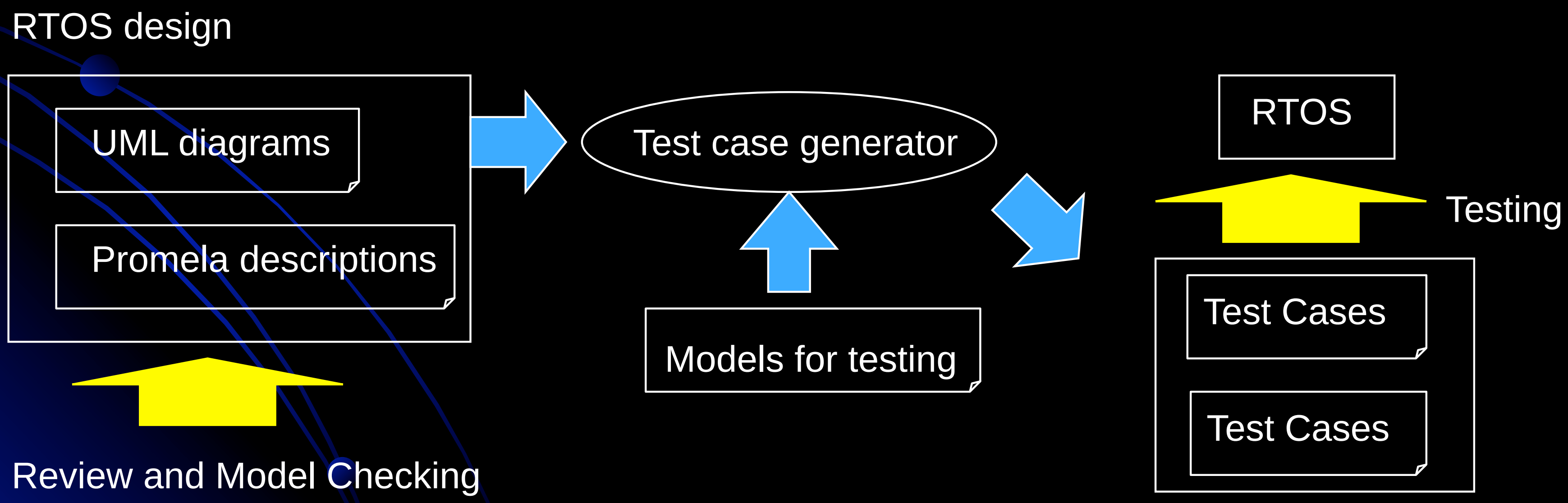


# Ongoing Works

- Model checking a huge number of descriptions using computer cluster.
  - How to assign the descriptions to machines of the computer cluster.
    - Achieve the load balance of the machines.
  - How to analyze results obtained by model checking.
    - Visualization of the results.
    - statistical analysis of the results.
- Formalization of the OSEK/VDX specification document.
  - We are describing the document in VDM.
    - Identifying what problems are to formalize standard documents.
  - We will start to study about desirable styles of standard documents in a formal specification language.
    - Working with Prof.Kokichi Futatsugi's group using Cafe/OBJ.

# Ongoing Works

- We are verifying a RTOS which will be embedded in a new series of cars.
  - Collaborating with DENSO and NEC(NEC Electronics and NEC Microsystems)
  - Conforming to ISO61508/26262 standards.
  - Improving the quality of RTOS.
- Mainly dealing with the design and testing of RTOS.



## Conclusion

- We introduced our approach to verifying a RTOS design with model checking.
  - Modeling environments of RTOS.
  - Automatic genenration of Promela descriptions for model checking the RTOS design.
  - Model checking is performed in computer cluster.
- Our environment modeling is applicable not only to the OSEK/VDX RTOS design but also to the other ones.
  - That will be effective to systems which have various environments such as OS, middleware and libraries.
  - We are developing tools which perform model checking in computer cluster, then analyze and visualize those results.
- We are applying our approach to practical OSEK/VDX RTOS with some automotive companies.

- Testing specification called MODISTARC is provided to ensure the conformance of RTOS to the OSEK/VDX standard.
  - MODISTARC does not provide test cases but functions to be tested.
    - We need define test cases based on MODISTARC.
  - It is very hard to define exhaustive test cases based on MODISTARC.
    - The test cases depend on RTOS products.