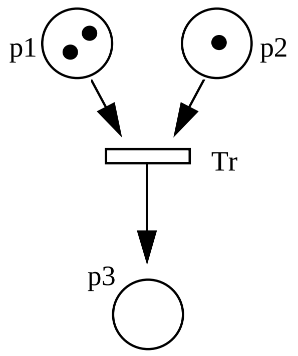
Appendix A

Cabernet and TER Nets

Cabernet is a customisable environment developed at the Politecnico de Milano specifically to research the benefits of formal techniques in the development of real-time systems [112]. The Cabernet formal kernel (Cab Nets) is based on a HLTPN called TER Nets (Time Environment/Relationship nets) [48, 100]. In Cab Nets, C++ is used as annotation language to specify actions and predicates, which are associated to transitions, and data types which are associated to places, and indirectly to tokens. Actions perform data transformations, removing and producing tokens when a transition fires. Predicates act on top of the basic Petri net firing rules, defining the token selection policy to be used on firings.

Tokens carry data values and a special time variable, called *chronos*, which represents the time when the token was produced [47]. Tokens are seen as *environments* with associated values (that are carried by them) to variables (present in action declarations). Tokens created on a transition firing have the same value on their chronos variables. The firing policy guarantees a *firing sequence monotonicity*, i.e., the chronos values assigned in any firing sequence are monotonically non-decreasing with respect to the order in which they were produced. Figure A.1 shows a Cab net example where a transition *Tr* is allowed to fire if there are tokens in its input places and the value (*p1.x*) of a token from *p*1 is smaller than the value (*p2.x*) of a token from *p2*. When *Tr* fires, it produces a token to place *p3* which corresponds to the addition of *p1.x* and *p2.x*.



predicate(tr)::p1.x < p2.y
action(tr):: p3.y = p1.x + p2.y

Figure A.1. A simple Cab net.

Cab nets have inherited the formal semantics of the time model present in TER nets. So, besides the common analysis techniques that can be carried out over the underlying pure Petri net, other formal techniques developed for proving temporal properties can also be used.

A.1 Tracing Tool

Since Cabernet does not offer any facilities for recording data generated during simulation, the execution analysis of big complex systems becomes very difficult and tedious without the possibility of using automated analysis techniques which could be carried out later.

Nevertheless, Cabernet is a customisable environment and so is able to be adapted to specific user needs. In fact, Cabernet has a *tool generator* which permits the designer to create tools using safe untimed Cab nets as an agent to glue together primitive operations available in the system or even other tools already defined by the user.

Thus, a simulation recording tool was created using these primitives together with some new primitives that were developed specially to this case in order to enable file creation and handling. These new primitives were incorporated by the author of this dissertation thanks to the Politecnico de Milano, which made the Cabernet source code available. Nowadays, these new primitives are an integral part of the Cabernet environment as it is distributed by the Politecnico de Milano. Figure A.2 shows the simulation recording tool developed. The new added primitives are: *getEnabTime*(), *getSubnetName*(), and *printfile*(). Also, the primitive *fire*() had some modifications. Details of each primitive can be found in [112].

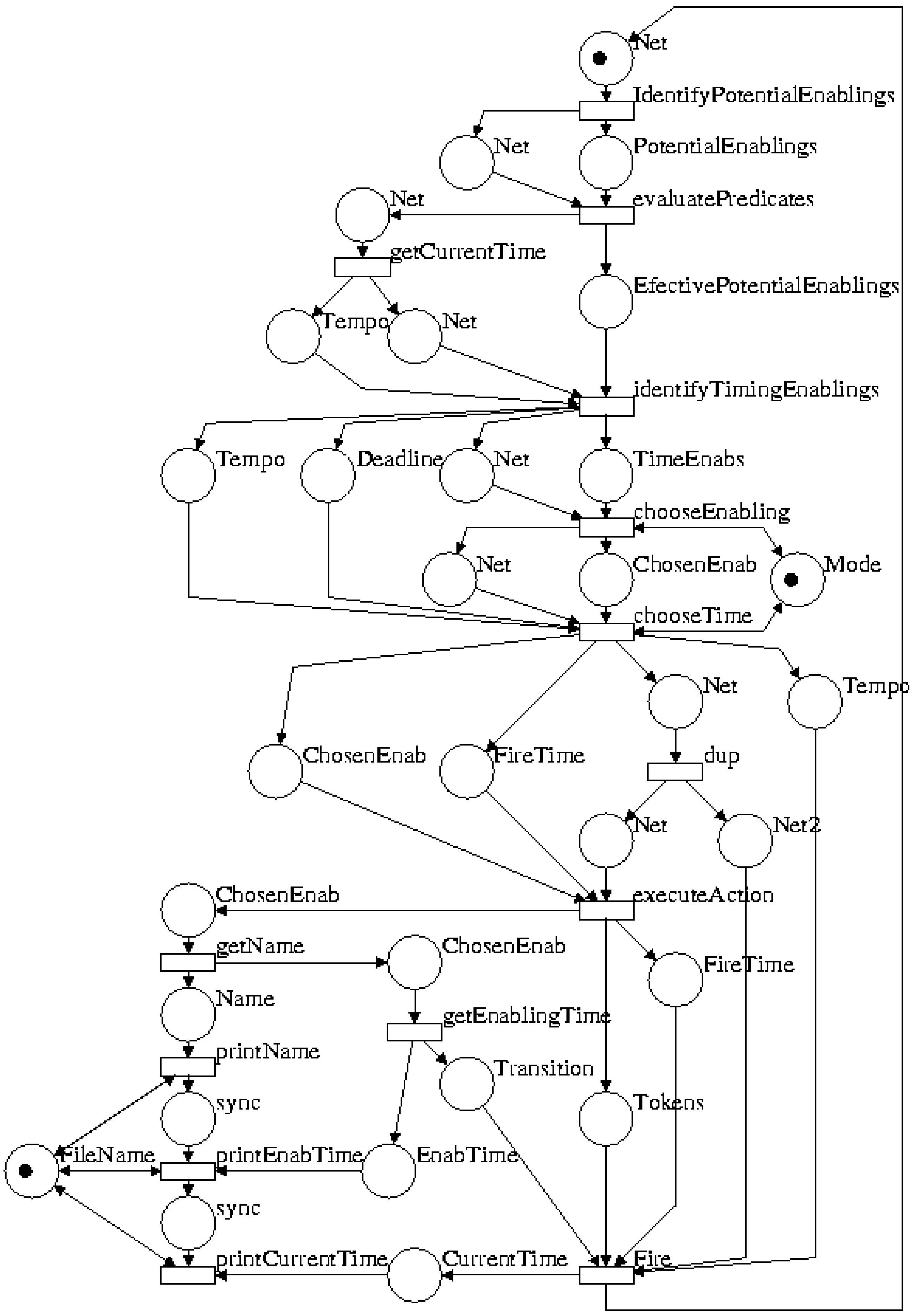


Figure A.2. Simulation Recording Tool

Appendix B

Petri Net Extension Semantics

The figures below show the TER net equivalents to the extensions proposed in this dissertation. The symbols *T* and *A* are used to represent execution time and action, respectively. All extensions proposed appear on the left-hand side of the symbol "≡", and the equivalent TER net representation appear on the right-hand side.

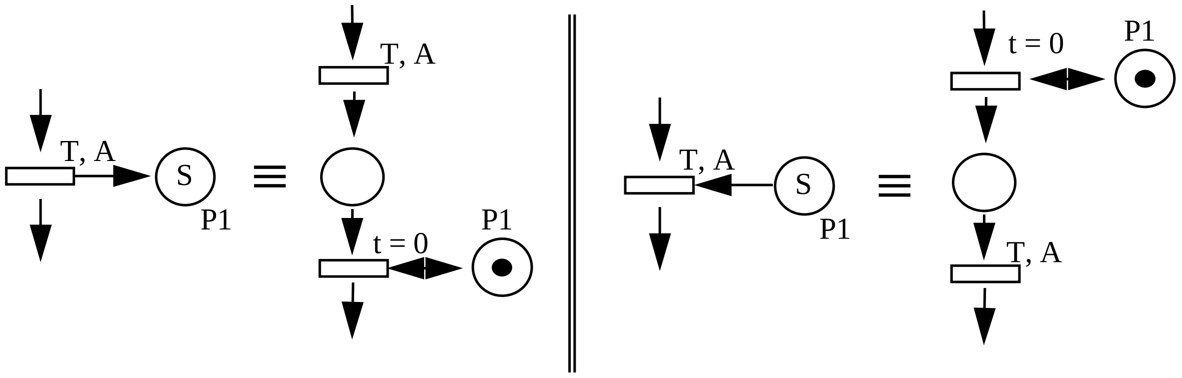
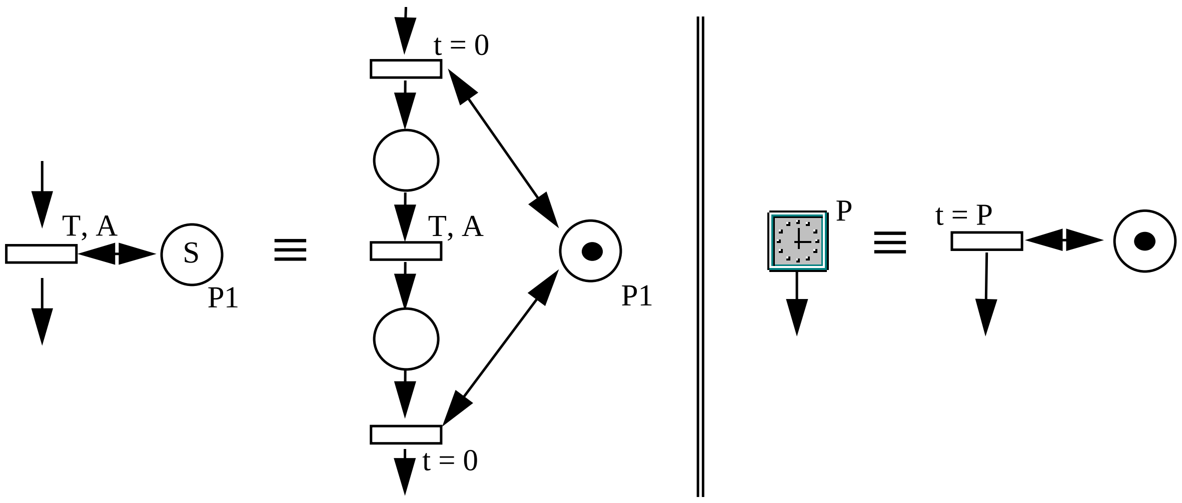


Figure B.1. Store equivalents when used for input or output.



a) Stores used for input and output. **b) Periodic transitions**

Figure B.2. Store and periodic transitions

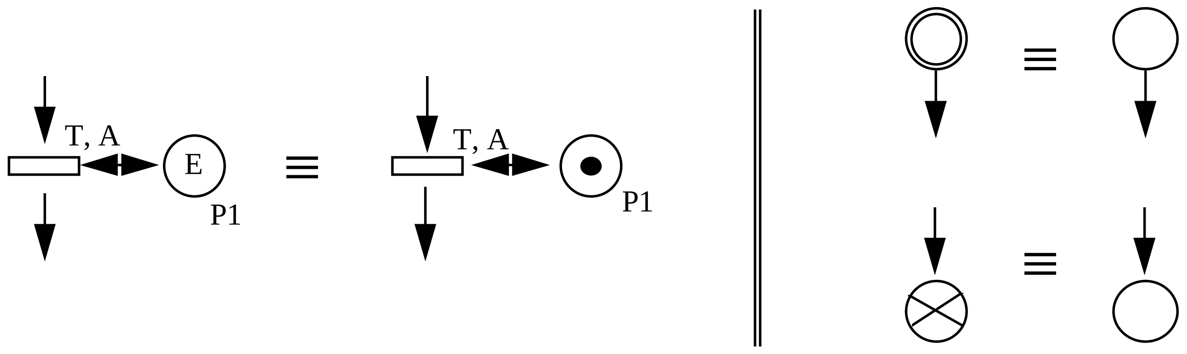


Figure B.3. Mutual exclusion, input and output places.

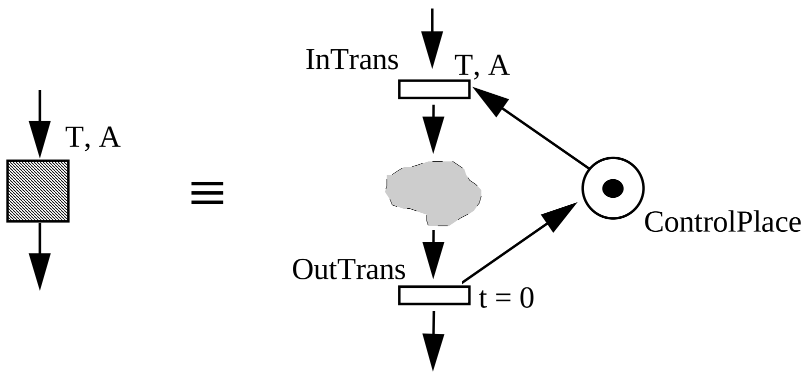


Figure B.4. Subnet equivalent.

Appendix C
Theorems

Theorem 1: $\mathbb{Z}(l)$ is a set of segments whose completion time precede and include l's completion time and which execute within a period of time where it does not exist any instant t when all processors are idle.

Proof: ***i***) The first part comes from the definition of $\mathbb{Z}(l)$ (Section 4.5).

ii.a) In order to prove the second part it is necessary to prove that $\forall i$ than there is no instant t, $r'_i \leq t < e_i$, when all processors are idle. The proof is done using contradiction with the following assumption:

Assumption: There is an instant t inside the given interval when all processors are idle.

Using the eligibility conditions (Section 4.4.3), and since $r'_i \leq t < e_i$, than it is clear that Condition 1.a is true. So, either Condition 1.b, or 1.c, or 2 must be false, otherwise i would be eligible for execution, and the initial assumption would become false.

If Condition 2 is false, then there exist a segment j_1 which satisfies Condition 1 and makes Condition 2 be false for i. But from the initial assumption, j_1 is not executing either and the same reasoning can be applied recursively. So, it would be necessary an infinite number of segments with a false Condition 2 to keep one of these segments from executing, which is impossible since the segment set is finite.

If Condition 1.b is false, i.e., $\exists\, j \mid j \otimes i \wedge s_j < t \wedge \neg(e_j \leq t)$, than j has started its execution before t and has not finished yet. Since at instant s_j all conditions were true for j, then Condition 1 is true at time $t \in (s_j, e_j)$ as well, since Condition 1.a is true and Conditions 1.b and 1.c must be true, otherwise j would not have started in the first place. Hence, the only reason for j not execute at time t is if Condition 2 is false for j. Then again the argument used for Condition 2 above can be used and so some segment is executing.

If Condition 1.c is false, i.e., $\exists\, j_1 \mid j_1 \mapsto i \wedge \neg(e_{j1} \leq t)$, and using the release adjustment rule (Section 4.4.1) $r'_{j1} \leq r'_i$, than $r'_{j1} \leq t < e_{j1}$. This corresponds to the initial argument used for segment *i*. Then again recursion, but this time applied over the whole proof, can be used to show that an infinite number of segments is necessary.

Since all possibilities proved impossible, the initial assumption cannot be true.

ii.b) From $\mathbb{Z}(l)$ definition, $\forall i \in \mathbb{Z}(l), i \neq l, \exists k \in \mathbb{Z}(l) \mid r_k < e_i$, and from *ii.a* it is clear that there is no time $t \in (s_i, e_k)$ where all processors are idle. Applying recursively from *l*, it follows that there is no time $t \in (s_i , e_l)$ where all processors are idle, and the proof is finished.

Theorem 2: If the schedule is not feasible then a feasible schedule can only be found if it is possible to re-schedule some pair of segments in $\mathbb{Z}(l)$ such that *l* is scheduled earlier.

Proof: It is obvious that since *l* is the latest segment the only way to improve the result is to have *l* scheduled earlier. From Theorem , $\mathbb{Z}(l)$ comprises all segments that execute between $\min\{ r'_i \mid i \in \mathbb{Z}(l) \}$and e_l.

To prove that re-scheduling segments that are not in $\mathbb{Z}(l)$ do not improve *l*'s schedule, it is necessary to notice that if $i \notin \mathbb{Z}(l) \Rightarrow \forall j \in \mathbb{Z}(l), e_i \leq r'_j \lor e_i \geq e_l$ (from $\mathbb{Z}(l)$'s definition). No segment can start executing before its release time. So, since $e_i \leq r'_j$, no $j \in \mathbb{Z}(l)$ can be scheduled before e_i. So, if *i* changes its execution order with *j*, then *i* would have to start executing in a time $t \geq r'_j$. The execution of *i* among segments in $\mathbb{Z}(l)$ would bring two possibilities:

1. *i* executes in a processor that was previously idle during that time. If *i* does not change the scheduling of any other segment (e.g. through precedence or exclusion relations), then nothing changes and no improvements were made. Otherwise, those affected segments would be pushed to execute later, since they cannot use processor time units previously used by *i*, resulting in a worse schedule.

2. *i* uses processor time units which were used by segments in $\mathbb{Z}(l)$, and again since these segments cannot use processor time units previously used by *i*, they would be pushed to execute later and a worse schedule would result.

Now consider that $e_i \geq e_l$. If *i* is brought to execute before e_l, the contention for the processor utilisation would increase which would bring no benefits to the schedulability.

Appendix D

Partitioning Report

An example of an automatically generated report produced by the partitioner tool is shown here. The report below describes the partitioning process to which the example shown in section 5.7 was submitted.

```
******************************************************************
       PARTITIONING RESULTS
******************************************************************

******************************************************************
       TASKS INITIAL ATTRIBUTES
******************************************************************

TASK ID: 1
   Name: T0
   Processor: sw
   Implementation: SW and FIXED
   Period  : 200
   Rel.Time: 0
   WCET  SW: 50
   WCET  HW: 50
   Deadline: 151

TASK ID: 2
   Name: T11
   Processor: sw
   Implementation: SW and NOT FIXED
   Period  : 200
   Rel.Time: 1
   WCET  SW: 40
   WCET  HW: 20
   Deadline: 51

TASK ID: 3
   Name: T12
   Processor: sw
   Implementation: SW and NOT FIXED
   Period  : 200
   Rel.Time: 40
   WCET  SW: 10
   WCET  HW: 10
   Deadline: 91

TASK ID: 4
   Name: T21
   Processor: sw
   Implementation: SW and NOT FIXED
   Period  : 200
   Rel.Time: 60
   WCET  SW: 50
   WCET  HW: 10
   Deadline: 140
```

```
TASK ID: 5
   Name: T22
   Processor: sw
   Implementation: SW and NOT FIXED
   Period  : 200
   Rel.Time: 0
   WCET  SW: 20
   WCET  HW: 10
   Deadline: 140

TASK ID: 6
   Name: T23
   Processor: sw
   Implementation: SW and NOT FIXED
   Period  : 200
   Rel.Time: 90
   WCET  SW: 50
   WCET  HW: 30
   Deadline: 140

*******************
EXCLUSION RELATIONS
*******************

   (1, 2) (1, 3)
   (2, 1) (2, 4)
   (3, 1) (3, 4) (3, 6)
   (4, 2) (4, 3)
   (6, 3)


********************
PRECEDENCE RELATIONS
********************

   (2, 3)
   (4, 5)
   (5, 6)


*********************************************
COMMUNICATION DATA: (task1, task2, num.Bytes)
*********************************************


******************************************************************
       COST FUNCTION WEIGHTS
******************************************************************

   weight 1 = 1
   weight 2 = 1
   weight 3 = 1
   weight 4 = 1
   weight 5 = 1



TASK CONSISTENCY CHECK RESULTS:
*******************************

   All tasks are consistent.

SCHEDULING PERIOD: 200

NUMBER OF SEGMENTS: 6
```

```
*******************************************************************
       PRE-PARTITIONING:
*******************************************************************

INDIVIDUAL TASK PARTITIONING:

   PRECEDENCE ADJUSTMENT AND PARTITIONING BASED ON EXCLUSION:
      New relation (seg/task): (2/3) -> (3/4)
      New relation (seg/task): (2/3) -> (5/6)
      New relation (seg/task): (1/2) -> (0/1)

   CONSTRAINT ADJUSTMENT AND PARTITIONING BASED ON PRECEDENCE:

   Partitioning List (seg/task) = (5/6) (4/5) (3/4) (2/3) (1/2)
   Tasks Moved to Hw:
      Task 4

   PRECEDENCE ADJUSTMENT AND PARTITIONING BASED ON EXCLUSION:
      New relation (seg/task): (2/3) -> (5/6)
      New relation (seg/task): (1/2) -> (0/1)

   CONSTRAINT ADJUSTMENT AND PARTITIONING BASED ON PRECEDENCE:
      Constraint adjustments performed

   PRECEDENCE ADJUSTMENT AND PARTITIONING BASED ON EXCLUSION:
      New relation (seg/task): (2/3) -> (0/1)

   CONSTRAINT ADJUSTMENT AND PARTITIONING BASED ON PRECEDENCE:
      Constraint adjustments performed

   PRECEDENCE ADJUSTMENT AND PARTITIONING BASED ON EXCLUSION:
      No changes made

*******************************************************************
       SYSTEM PARTITIONING
*******************************************************************

BRANCH AND BOUND SCHEDULING
***************************

EDF SCHEDULING OF NODE 0

   CONSTRAINT ADJUSTMENT BASED ON PRECEDENCE:
      No changes made

   PRECEDENCE ADJUSTMENT BASED ON EXCLUSION:
      No changes made

   Lateness   = 20
   Lowerbound = 20

   Last Late Segment/Task = 0/1

   Partitioning Set (seg/task):
      (0/1)  (4/5)  (5/6)

   Lateness P.S. = 20

The scheduling is NOT FEASIBLE

The scheduling is OPTIMAL
```

```
SYSTEM PARTITIONING ITERATION No. 1
***************************************

   Partitioning Set from Node No. 0
   Minimum time to be moved: 20
   Tasks Moved to Hw:
      Task 6

   Total moved time: 50

BRANCH AND BOUND SCHEDULING
***************************

EDF SCHEDULING OF NODE 0

   CONSTRAINT ADJUSTMENT BASED ON PRECEDENCE:
      Constraint adjustments performed

   PRECEDENCE ADJUSTMENT BASED ON EXCLUSION:
      New relation (seg/task): (2/3) -> (5/6)
      New relation (seg/task): (1/2) -> (0/1)

   CONSTRAINT ADJUSTMENT BASED ON PRECEDENCE:
      Constraint adjustments performed

   PRECEDENCE ADJUSTMENT BASED ON EXCLUSION:
      New relation (seg/task): (2/3) -> (0/1)

   CONSTRAINT ADJUSTMENT BASED ON PRECEDENCE:
      Constraint adjustments performed

   PRECEDENCE ADJUSTMENT BASED ON EXCLUSION:
      No changes made

   Lateness   = -10
   Lowerbound = -10

The scheduling is FEASIBLE
The scheduling is OPTIMAL

******************************************************************
       SCHEDULING RESULT OF FEASIBLE NODE 0
******************************************************************

Time: 0
   Segment  : The processor is IDLE
   TASK   ID: NONE
   Processor: sw

Time: 0
   Segment  : The processor is IDLE
   TASK   ID: NONE
   Processor: _hw1

Time: 0
   Segment  : The processor is IDLE
   TASK   ID: NONE
   Processor: _hw2

Time: 1
   Segment  : 1
   TASK   ID: 2
   Processor: sw

Time: 41
   Segment  : 2
```

```
    TASK   ID: 3
    Processor: sw

Time: 51
    Segment  : 0
    TASK   ID: 1
    Processor: sw

Time: 60
    Segment  : 3
    TASK   ID: 4
    Processor: _hw1

Time: 70
    Segment  : The processor is IDLE
    TASK   ID: NONE
    Processor: _hw1

Time: 70
    Segment  : 4
    TASK   ID: 5
    Processor: sw

Time: 90
    Segment  : 0
    TASK   ID: 1
    Processor: sw

Time: 90
    Segment  : 5
    TASK   ID: 6
    Processor: _hw2

Time: 120
    Segment  : The processor is IDLE
    TASK   ID: NONE
    Processor: _hw2

Time: 121
    Segment  : The processor is IDLE
    TASK   ID: NONE
    Processor: sw


******************************************************************
       TASKS FINAL ATTRIBUTES
******************************************************************
TASK ID: 1
    Name: T0
    Processor: sw
    Implementation: SW and FIXED
    Period  : 200
    Rel.Time: 0
    WCET  SW: 50
    WCET  HW: 50
    Deadline: 151

TASK ID: 2
    Name: T11
    Processor: sw
    Implementation: SW and NOT FIXED
    Period  : 200
    Rel.Time: 1
    WCET  SW: 40
    WCET  HW: 20
    Deadline: 51
```

```
TASK ID: 3
    Name: T12
    Processor: sw
    Implementation: SW and NOT FIXED
    Period  : 200
    Rel.Time: 40
    WCET  SW: 10
    WCET  HW: 10
    Deadline: 91

TASK ID: 4
    Name: T21
    Processor: _hw1
    Implementation: HW and FIXED
    Period  : 200
    Rel.Time: 60
    WCET  SW: 50
    WCET  HW: 10
    Deadline: 140

TASK ID: 5
    Name: T22
    Processor: sw
    Implementation: SW and NOT FIXED
    Period  : 200
    Rel.Time: 0
    WCET  SW: 20
    WCET  HW: 10
    Deadline: 140

TASK ID: 6
    Name: T23
    Processor: _hw2
    Implementation: HW and FIXED
    Period  : 200
    Rel.Time: 90
    WCET  SW: 50
    WCET  HW: 30
    Deadline: 140
```