APPENDIX C: MAPPING SCENARIO INTO PETRI-NET

A Petri-Net ***PN*** is derived from a structured scenario ***S*** by identifying: scenario triggering – initial event (Title, Goal, Context, Resource and Actor), event ocurrences (episode sentence and alternative solution step) and their guard conditions and constraints, non-sequential constructs (#<Episodes Series>#), and scenario completion – final event (Context: Post-condition). For each *event*, a *transition* is created for denoting the location of event occurrence. *Input places* are created to denote the locations of its c*onditions* (pre-condition, condition, cause) and *constraints*. *Output places* are created to denote the location of its *post-conditions*. *Event*, *condition* and *constraint* labels are assigned to these *transitions* and *places*, accordingly. The initial marking ***M₀*** of the ***PN*** is then created to denote the *initial state*, in which tokens are added into input places that represent pre-conditions (context) or constraints. Execution of the scenario begins at this initial marking ***M₀*** which semantically means the system initial state, including the availability of all resources, pre-conditions or constraints. It ends at the same marking that semantically means the release of these resources, pre-conditions or constraints.

The *first step* of the transformation method applies mapping rules to translate scenario triggering, event occurrences and scenario completion into Petri-Net elements (transition, place and arc). Fig. 1 depicts the visual transformations from Scenario into Petri-Net using a structure composed of left-hand and right-hand sides (LHS → RHS). LHS is the conditional part of the rule (scenario section or component), and RHS is basically the expected result of the rule (sub Petri-Net).

In order to preserve the event sequences, we add appropriate *input dummy place, output dummy place* or *dummy transition* to the sub Petri-Nets. These dummy places link sub Petri-Nets derived from sequential events (e.g. episode 1 and episode 2), enabling the information flow among events. A *dummy transition* is added to the sub Petri-Nets derived from scenario triggering, scenario completion, conditional event occurrences (conditional or Loop episode) and non-sequential constructs (#<Episodes Series>#).

The *second step* composes the sub Petri-Nets generated from scenario sections into a whole Petri-Net by *Link* (Definition 4.1) and *Fusion* (Definition 4.2) operations.

Definition 4.1 (Link Sub Petri-Nets). Two sub Petri-Nets are linked among them by fusing the *output dummy place* of the first one with the *input dummy place* of the second one.

Definition 4.2 (Fusion Place). *Places* with the same label are fused among them by merging their *input* and *output arcs*. Fig. 2 describes the steps of the algorithm for transforming a structured scenario into a Petri-Net.

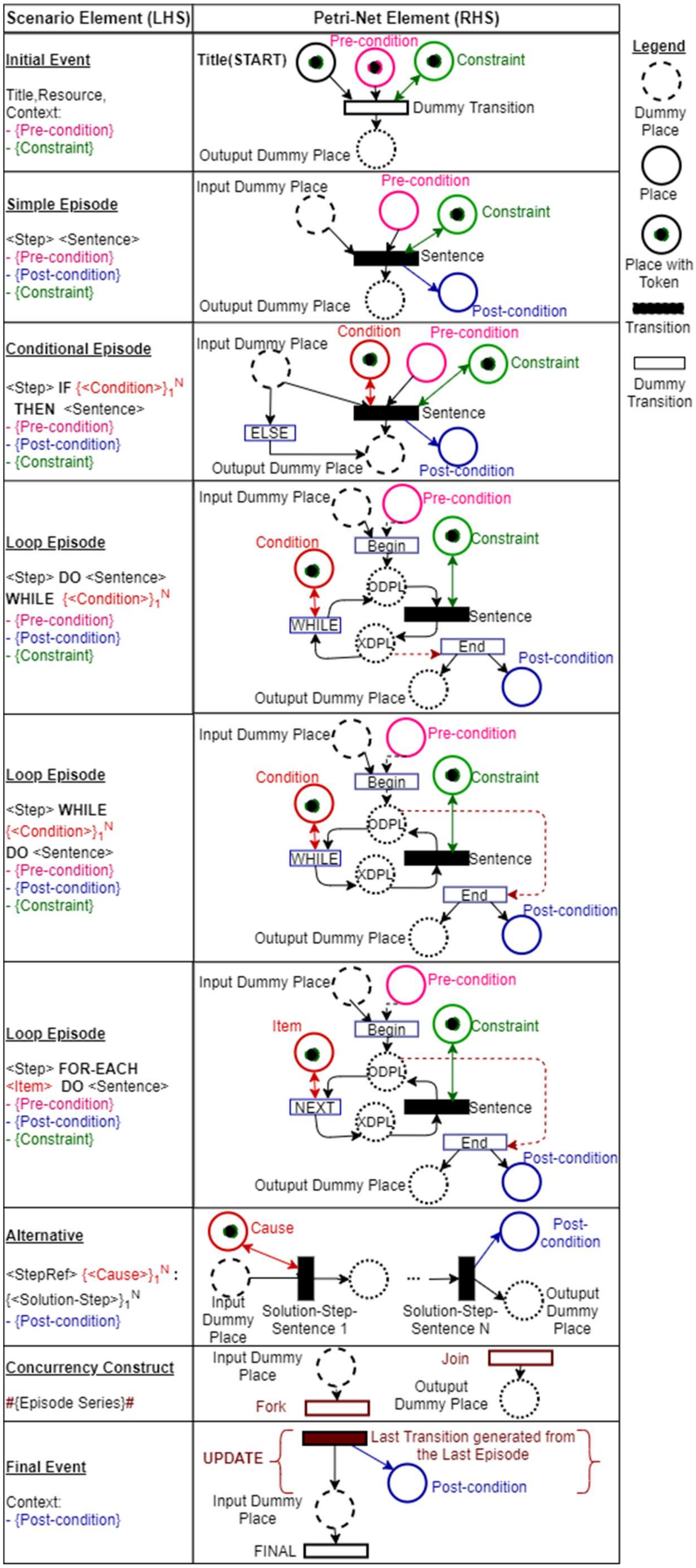
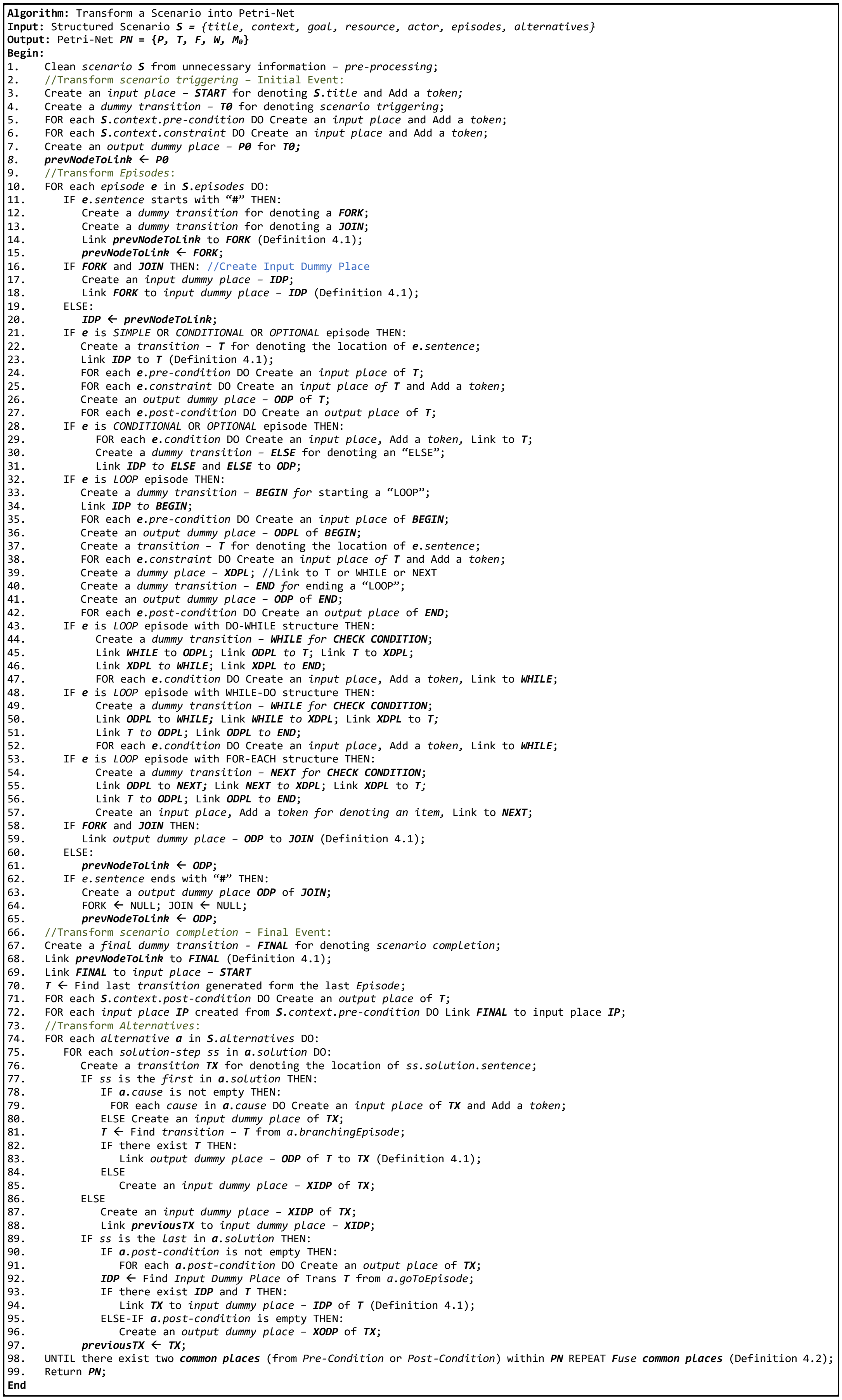


Fig. 1. Mapping scenario elements into Petri-Net elements.



```
Algorithm: Transform a Scenario into Petri-Net
Input: Structured Scenario S = {title, context, goal, resource, actor, episodes, alternatives}
Output: Petri-Net $PN = \{P, T, F, W, M_0\}$
Begin:
    Clean scenario S from unnecessary information – pre-processing;
    //Transform scenario triggering – Initial Event:
    Create an input place – START for denoting S.title and Add a token;
    Create a dummy transition – T0 for denoting scenario triggering;
    FOR each S.context.pre-condition DO Create an input place and Add a token;
    FOR each S.context.constraint DO Create an input place and Add a token;
    Create an output dummy place – P0 for T0;
    prevNodeToLink ← P0
    //Transform Episodes:
    FOR each episode e in S.episodes DO:
       IF e.sentence starts with “#” THEN:
          Create a dummy transition for denoting a FORK;
          Create a dummy transition for denoting a JOIN;
          Link prevNodeToLink to FORK (Definition 4.1);
          prevNodeToLink ← FORK;
       IF FORK and JOIN THEN: //Create Input Dummy Place
          Create an input dummy place – IDP;
          Link FORK to input dummy place – IDP (Definition 4.1);
       ELSE:
          IDP ← prevNodeToLink;
       IF e is SIMPLE OR CONDITIONAL OR OPTIONAL episode THEN:
          Create a transition – T for denoting the location of e.sentence;
          Link IDP to T (Definition 4.1);
          FOR each e.pre-condition DO Create an input place of T;
          FOR each e.constraint DO Create an input place of T and Add a token;
          Create an output dummy place – ODP of T;
          FOR each e.post-condition DO Create an output place of T;
       IF e is CONDITIONAL OR OPTIONAL episode THEN:
            FOR each e.condition DO Create an input place, Add a token, Link to T;
            Create a dummy transition – ELSE for denoting an “ELSE”;
            Link IDP to ELSE and ELSE to ODP;
       IF e is LOOP episode THEN:
          Create a dummy transition – BEGIN for starting a “LOOP”;
          Link IDP to BEGIN;
          FOR each e.pre-condition DO Create an input place of BEGIN;
          Create an output dummy place – ODPL of BEGIN;
          Create a transition – T for denoting the location of e.sentence;
          FOR each e.constraint DO Create an input place of T and Add a token;
          Create a dummy place – XDPL; //Link to T or WHILE or NEXT
          Create a dummy transition – END for ending a “LOOP”;
          Create an output dummy place – ODP of END;
          FOR each e.post-condition DO Create an output place of END;
       IF e is LOOP episode with DO-WHILE structure THEN:
            Create a dummy transition – WHILE for CHECK CONDITION;
            Link WHILE to ODPL; Link ODPL to T; Link T to XDPL;
            Link XDPL to WHILE; Link XDPL to END;
            FOR each e.condition DO Create an input place, Add a token, Link to WHILE;
       IF e is LOOP episode with WHILE-DO structure THEN:
            Create a dummy transition – WHILE for CHECK CONDITION;
            Link ODPL to WHILE; Link WHILE to XDPL; Link XDPL to T;
            Link T to ODPL; Link ODPL to END;
            FOR each e.condition DO Create an input place, Add a token, Link to WHILE;
       IF e is LOOP episode with FOR-EACH structure THEN:
            Create a dummy transition – NEXT for CHECK CONDITION;
            Link ODPL to NEXT; Link NEXT to XDPL; Link XDPL to T;
            Link T to ODPL; Link ODPL to END;
            Create an input place, Add a token for denoting an item, Link to NEXT;
       IF FORK and JOIN THEN:
          Link output dummy place – ODP to JOIN (Definition 4.1);
       ELSE:
          prevNodeToLink ← ODP;
       IF e.sentence ends with “#” THEN:
          Create a output dummy place ODP of JOIN;
          FORK ← NULL; JOIN ← NULL;
          prevNodeToLink ← ODP;
    //Transform scenario completion – Final Event:
    Create a final dummy transition - FINAL for denoting scenario completion;
    Link prevNodeToLink to FINAL (Definition 4.1);
    Link FINAL to input place – START
    T ← Find last transition generated form the last Episode;
    FOR each S.context.post-condition DO Create an output place of T;
    FOR each input place IP created from S.context.pre-condition DO Link FINAL to input place IP;
    //Transform Alternatives:
    FOR each alternative a in S.alternatives DO:
       FOR each solution-step ss in a.solution DO:
          Create a transition TX for denoting the location of ss.solution.sentence;
          IF ss is the first in a.solution THEN:
             IF a.cause is not empty THEN:
               FOR each cause in a.cause DO Create an input place of TX and Add a token;
             ELSE Create an input dummy place of TX;
             T ← Find transition – T from a.branchingEpisode;
             IF there exist T THEN:
                Link output dummy place – ODP of T to TX (Definition 4.1);
             ELSE
                Create an input dummy place – XIDP of TX;
          ELSE
             Create an input dummy place – XIDP of TX;
             Link previousTX to input dummy place – XIDP;
          IF ss is the last in a.solution THEN:
             IF a.post-condition is not empty THEN:
                FOR each a.post-condition DO Create an output place of TX;
             IDP ← Find Input Dummy Place of Trans T from a.goToEpisode;
             IF there exist IDP and T THEN:
                Link TX to input dummy place – IDP of T (Definition 4.1);
             ELSE-IF a.post-condition is empty THEN:
                Create an output dummy place – XODP of TX;
          previousTX ← TX;
    UNTIL there exist two common places (from Pre-Condition or Post-Condition) within PN REPEAT Fuse common places (Definition 4.2);
    Return PN;
End
```

Fig. 2. Transforming a Scenario into Petri-Net.