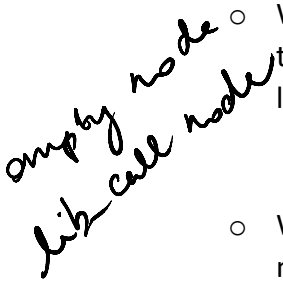
PLEXIL Timing Analysis – Design Notes

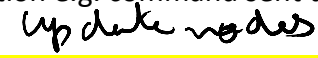
Author: Pranav Srinivas Kumar

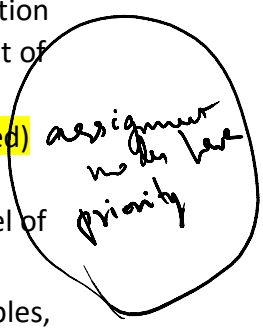
This document presents a preliminary design for timing analysis of Plexil Plans

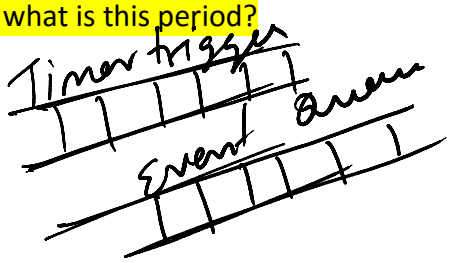
Plexil Summary:

- Plexil plans hierarchically represent execution nodes.
- Execution nodes are arranged as a tree
  - Leaf nodes are action nodes and internal nodes are control nodes.
- The execution of each node is governed by a set of conditions.
  - When action nodes execute, the node execution takes a finite worst-case execution time (WCET) to perform an action e.g. command sent to an actuator, assignment of local variable etc.
    - **QUESTION**: How is conflict resolved when multiple nodes (when executed) affect the same variables? E.g. parallel assignments
  - When internal control nodes are executed, they are expanded into the next level of nodes in the tree.
- Conditions dictate the execution of nodes and may depend on both environment variables, and local variables.
- Interface to access the environment variable state – Lookups can appear in both conditions and assignments
  - LookupNow – A single 'request-based' lookup
    - **QUESTION**: What happens if the Server providing this state has blocking semantics? <- Can a WCET be guaranteed?
  - LookupOnChange – Event-based repeated lookup
  - LookupWithFrequency – Repeated lookup with certain frequency
- In general, Plexil node execution follows the following steps:
  - Is it time to execute nodes? ← Triggered by a periodic timer or by external events
    - If yes, check all node conditions
    - Find candidate nodes
    - Snapshot the state of all relevant environment variables & local variables
    - Execute all candidate nodes synchronously
- **Informal Semantics:**
  - Execution of plan proceeds in discrete time steps ← *macro steps*
  - All external events are processed in the order in which they were received
  - An external event and all its cascading effects are processed before the next event is processed ← run-to-completion semantics
    - **QUESTION**: What happens if no external events are received? Does the Plexil execution go to sleep? Or is there a periodic check on the node conditions? If so, what is this period?









Lookup Now
Lookup On change

- A macro step of execution consists of a number of *micro steps*
  - Each micro step corresponds to the parallel synchronous execution of atomic steps of the individual plan nodes
  - **QUESTION**: Is there any possible prioritization on node execution? Some nodes may be more important/critical than other nodes. When some critical error event occurs, can some Plexil node of high priority be executed first? Is there any determinism to the execution of Plexil nodes during this *"parallel synchronous execution"* stage?
  - **QUESTION**: Is the Plexil executive single-threaded? If so, then how does the parallel execution work? Is it perceived as parallel execution since (1) we use the same values/state for all variables for all node executions and (2) all events waiting in the queue are ignored till the current execution step completes?
- Plexil execution is also characterized by an event queue
  - The event queue is a FIFO queue that receives notifications on events
  - When an event "fires", the post-conditions of the event modify environment variables
  - Plexil execution acknowledges events in the event queue by checking node conditions and executing candidate nodes.

Timing Properties Considered:

In order to perform any useful timing analysis, Plexil plans need to be annotated with timing properties; properties that directly affect system properties like worst-case response times.

The following are the timing properties considered for **modeling** Plexil plan execution with Colored Petri nets:

- Minimum and Maximum inter-arrival times on events
  - Events are external to the Plexil plan
  - Events affect and often modify environment variables
  - Example Event model:
    - {Event="WheelStuck", InterArrivalTime=10, NextArrival=0, Effect = [{VariableName="WheelStuck", NewValue="true"}
- WCET of commands/assignments in leaf nodes
  - **ASSUMPTION**: All commands and assignments are asynchronous steps taking finite WCET i.e. execution is not dependent on any external factors (no blocking behavior)
- WCET on checking all conditions in each node
  - Condition checks may depend on environment variables in which case we use lookups
  - WCET of all lookups ← All lookups have an inherent cost – This cost may be negligible.
  - **ASSUMPTION**: When an environment variable changes (because of an event) during (or around the time of lookup), we use the updated value of the variable ← Is this right?
    - **QUESTION**: Or should I assume a snapshot of the variable made when the macro step started?

Command handler

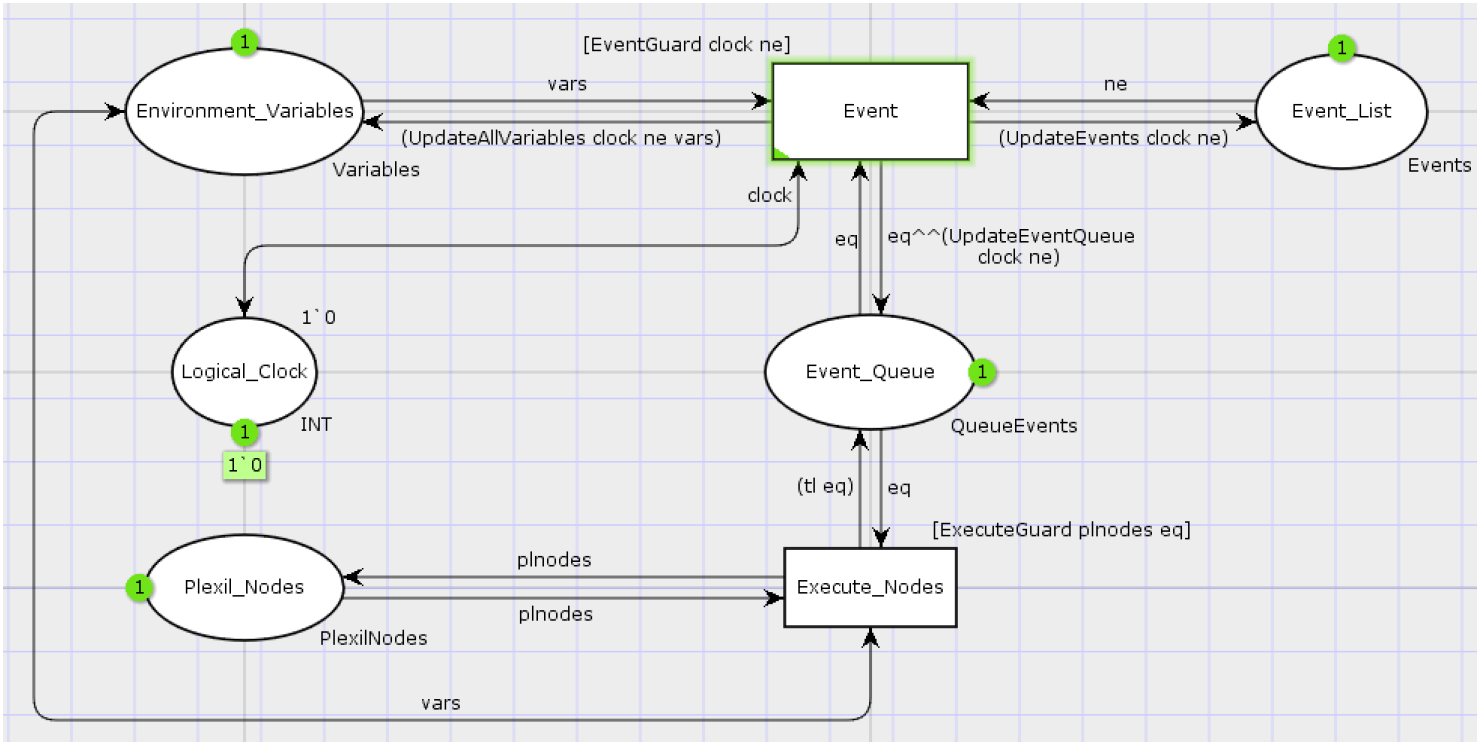
ACK

Lookup Now Can do a blocking query at

CPN Model:

The following is the CPN model for analyzing Plexil execution.

- The place *Event_List* holds a list of events
- The transition *Event* triggers when the *Logical_Clock* reaches the inter-arrival time of the event
- When an event fires, all relevant environment variables are updated with new values
- When an event fires, the event notification enqueues on the *Event_Queue*
- When events are available on the *Event_Queue*, the transition *Execute_Nodes* can fire.
- Execution of nodes will perform the (1) necessary checks, (2) find a subset of execution nodes to progress, and (3) execute the nodes (progressing state and updating local variables). The execution also calculates the WCET of each node/step.
    - **QUESTION**: Since the node execution is parallel, by how much will the Logical_Clock progress when one execution cycle completes i.e. when one round of execution (of all nodes) completes.
        - Is it MAX_WCET calculated as *max(WCET_node1, WCET_node2, …)*
        - Or is it *(WCET_node1 + WCET_node2 + …)*
- From this model, we can calculate worst-case trigger to response time
    - Trigger = some event
    - Response = completion of some plexil node execution e.g. leaf actuator node



Condition Checking:

- Conditions rely on data – both environment variables and local variables
- In CPN Tools, conditions need to be standard ML expressions that can be evaluated – eval(some_string) does not work
- So, conditions cannot be expressed as strings, part of a CPN token
- Conditions need to be generated as part of the "Execute" function that bind to arcs in the CPN model.
- This makes the model a bit less generic as the Execute function that drives the transition is now dependent on (and tightly bound to) the plan.

Sample Plan:

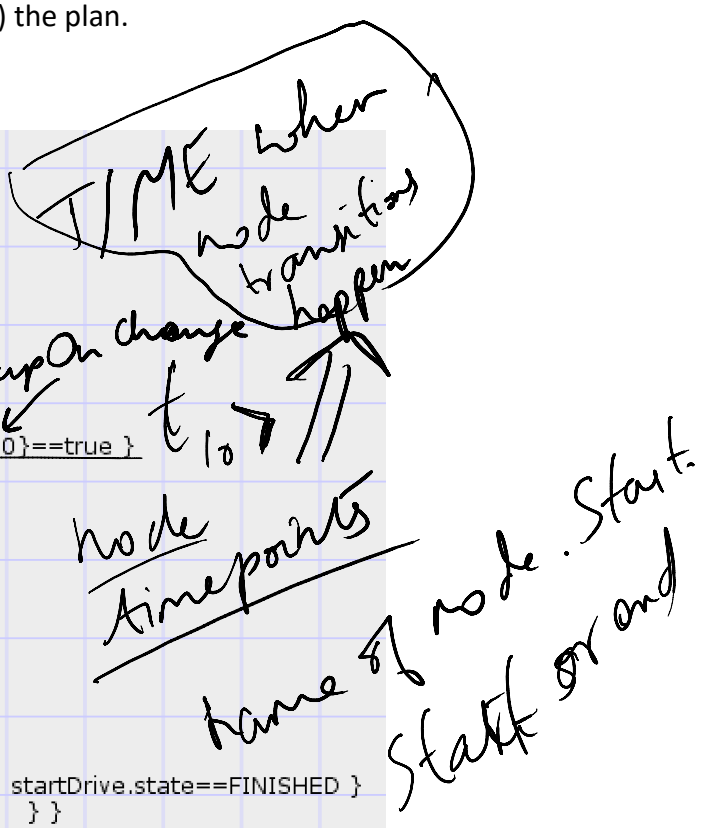
```
(* TEST PLEXIL PLAN *)
Node:{
   NodeId:DriveToRedRock1;
   Boolean haveRR=false, stop=false;

    NodeList:{

        Node:{
           NodeId: SenseRR;
           Interface:{ inout: haveRR;}
           StartCondition: {
                 LookupWithFrequency:{found_RR, 10}==true }
           Assignment: haveRR=true;    }

        Node:{
            NodeId:ContDrive
            Interface:{ in: stop; }
            NodeList:{
                  Node:{
                     NodeId: StartDrive;
                     Command: "Rover:drive";     }
                  Node:{
                     NodeId: StopDrive;
                     Interface:{ in stop;}
                     StartCondition: { stop==true &&  startDrive.state==FINISHED }
                     Command: "Rover:stop";    }       } }

         Node:{
              NodeId: SetRRFlag
              Interface:{ inout stop; in haveRR; }
              StartCondition: haveRR == true;
              Assignment: stop = true;   } } }
```



Sample Plexil Plan in Standard ML

```
(* PLEXIL NODES *)
1`[{NodeType="List", NodeID="DriveToRedRock1", State="Inactive",
                    Parent="NULL",
                    VariableList = [{VariableName="haveRR", VariableType="BOOL", VariableValue="false"},
                                    {VariableName="stop", VariableType="BOOL", VariableValue="false"}],
                    ConditionList=[],
                    AssignmentList=[],
                    CommandList=[],
                    Outcome=""},

   {NodeType="Assignment", NodeID="SenseRR", State="Inactive",
                Parent="DriveToRedRock1",
                    VariableList=[],
                    ConditionList=[{ConditionType="Start", ConditionName="StartCondition", Value="true", WCET=1, ExecTime=0}],
                    AssignmentList=[{VariableName="haveRR", VariableType="BOOL", NewValue="true"}],
                    CommandList=[],
                    Outcome=""},

   {NodeType="List", NodeID="ContDrive", State="Inactive",
                Parent="DriveToRedRock1",
                    VariableList=[],
                    ConditionList=[],
                    AssignmentList=[],
                    CommandList=[],
                    Outcome=""},

   {NodeType="Command", NodeID="StartDrive", State="Inactive",
                Parent="ContDrive",
                    VariableList=[],
                    ConditionList=[],
                    AssignmentList=[],
                    CommandList=[{CommandName="Rover::drive", WCET=1, ExecTime=0}],
                    Outcome=""},

   {NodeType="Command", NodeID="StopDrive", State="Inactive",
                Parent="ContDrive",
                    VariableList=[],
                    ConditionList=[{ConditionType="Start", ConditionName="StartCondition", Value="true", WCET=1, ExecTime=0}],
                    AssignmentList=[],
                    CommandList=[{CommandName="Rover::stop", WCET=1, ExecTime=0}],
                    Outcome=""},

   {NodeType="Assignment", NodeID="SetRRFlag", State="Inactive",
                Parent="DriveToRedRock1",
                    VariableList=[],
                    ConditionList=[{ConditionType="Start", ConditionName="StartCondition", Value="true", WCET=1, ExecTime=0}],
                    AssignmentList=[{VariableName="stop", VariableType="BOOL", NewValue="true"}],
                    CommandList=[],
                    Outcome=""}]
```