NASA Ames Research Center
Autonomous Systems and Robotics
Planning and Scheduling Group

PLEXIL Workshop

An Introduction to PLEXIL and the Universal Executive

Part 2: Standard Plexil Language

Standard Plexil – Outline

Standard Plexil - Introduction

- Standard programming syntax for PLEXIL
- Example

```
SimpleAssignment:
{
  Integer foo = 0;
  PostCondition: foo == 3;
  Assignment: foo = 3;
}
```

- Complied (translated) into PLEXIL XML
  - XML format described by XML schemas found in `plexil/schema`

Standard Plexil – Nodes

- General format:

```
<node name>:
{
  <node attributes>
  <node body>
}
```

- Node name, attributes, and body are all optional. E.g. an empty node:

```
{
}
```

Standard Plexil – Node Attributes

- Node Attributes
  - Variables
  - Conditions
  - Interface
  - Library Nodes

Standard Plexil – Variables

- A node may declare local variables.
  - Visible to the node and its descendants (lexical scope)
  - Of type Boolean, integer, real, string, or array
- Examples

```
Boolean isReset = true;
Integer n = 123;
Real pi = 3.14159;
String message = "hello there";
Integer scores[100];
Real defaults[10] = #(1.3 2.0 3.5);
```

Standard Plexil – Conditions

- A node's conditions are Boolean expressions.
  - If omitted, defaults apply
  - Up to one clause for each condition type:
    - Start, end, pre, post, invariant, and repeat condition
- Examples

```
StartCondition: Node1.outcome == SUCCESS;
EndCondition: SignalEndOfPlan.state == FINISHED ||
                              SendAbortUpdate.state == FINISHED;
PostCondition: AtGoal;
InvariantCondition: LookupOnChange(ZZZZCWEC5520J) == 1;
RepeatCondition: Count < 10;
```

Standard Plexil – Interface

- Example

```
{
  Integer x = 2;
  String message = “Enter number:” ;
  NodeList:
    {
      InOut Integer x;
      In String message;
      Integer y = 5;
      NodeList:
        { Assignment: x = y + 2; }
        { Command print(message); }
    }
}
```

  - Note that all nodes are anonymous!

Standard Plexil – Interface (continued)

- A node's *interface* is the set of variables it can access.
  - An interface includes *readable* (in) and *writable* (in-out) variables.
  - Defaults to the union of the parent's interface and local variables.
- Interface clauses *restrict* the node's interface.
  - By default, variables can be read and written in descendant nodes.
- When an Interface is clause is used, it defines the *entire* interface of the node.

Standard Plexil – Library Nodes

- Library nodes are nodes you wish to call in other nodes.
  - They are invoked by *Library Call Nodes*.
- Any node can be a library node.
  - Library nodes often have Interface clauses.
    - These interface clauses *must* declare the type.
- Library nodes are top level nodes (one per file).
- Upcoming slide on library nodes has examples.

Standard Plexil – Node Types

- Node Types
  - Empty Node
  - Assignment Node
  - Command Node
  - Function Call Node
  - Update Node
  - Library Call Node
  - List Node
- The type of the node is determined by its *body*.

Standard Plexil – Empty Node

- Empty nodes have no body. They may contain only attributes.
- Example:

```
VerifyTemp:
{
 PostCondition: LookupNow("engine_temperature") > 100.0;
}
```

- Common uses for empty nodes:
  - Verification of a state (as in above example)
  - Stubs (for testing or incremental development)

Standard Plexil – Assignment Node

- Identified by an Assignment clause, e.g.

```
// A simple assignment node

IncrementCounter:
{
  Assignment: ExecutionCount = 1 + ExecutionCount;
}
```

- The assigned variable must be writable.
- The source (RHS) of the assignment is an expression whose type must match that of the variable.
- Expressions are described later.

Standard Plexil – Command Node

- Identified by a Command clause, e.g.

```
// A simple command node
ConfirmProceed:
{
  Boolean result;
  EndCondition: isKnown(result);
  PostCondition: result;
  Command: result = QueryYesNo("Proceed with instructions?");
}
```

- The assigned variable is optional and must be writable.

- Call to command immediately returns a *handle,* finishing the node. (Plan's execution is not blocked).
  - This is independent of the returned *value*.

Standard Plexil – Function Call Node

- Identified by the FunctionCall statement.

```
// A simple function call node
EstimatedTimeOfArrival:
{
  In Real x, y, z;
  InOut Real ETA;
  FunctionCall: ETA = ComputeETA (x,y,z);
}
```

- The assigned variable is optional and must be writable.

- Very similar to Command node. Differences:
  - Semantically, should not have effect on external world
  - Node ends after value is returned

Standard Plexil – Update Node

- Identified by an Update clause

```
// A simple update node
SendAbortUpdate:
{
  StartCondition: MonitorAbortSignal.state == FINISHED;
  Update: taskId = taskTypeAndId[1], result = -2;
}
```

- Any number of name/value bindings are allowed.

Standard Plexil – Library Call Node

- Identified by a LibraryCall clause
  - Example library node:

```
F:
{
  In Integer i;
  InOut Integer j;
  Assignment: j = j * j + i;
}
```

  - Example call to above library node (note declaration):

```
LibraryNode F(In Integer i, InOut Integer j);

LibraryCallTest:
{
  Integer k = 2;
  LibraryCall: F(i=12, j=k);
}
```

Standard Plexil – List Node

- Identified by a NodeList clause. Example:

```
Root:
{
  NodeList:
    { Assignment: count = count + 1; }
    Detect:
     { StartCondition: LookupOnChange(“button-pressed”); }
    React:
    {
      StartCondition: Detect.State == FINISHED;
      Command: activate_device()
    }
 }
```

  - The first node is anonymous and unconstrained.
  - The second node, `Detect`, is empty.
  - The third node, `React`, runs after `Detect`.

Standard Plexil – Wrapup

- Data types and expressions
  - The UNKNOWN value
  - Numeric Expressions
  - Boolean Expressions
  - String Expressions
  - Arrays
- World State (lookups)
- Node State
- Compiling into XML

Standard Plexil – The UNKNOWN value

- Extends every type
- Default initial value for variables and array elements
- Results when a lookup fails
- Results when a requested node timepoint is invalid
- Part of PLEXIL's *three-value* Boolean logic
- Not a literal – cannot be used in a plan
  - Instead, queried through isKnown operator

Standard Plexil - Numeric Expressions

- Evaluate to numbers (integer or real)
- Literals
  - Integers
  - Reals
- Variables of type integer or real
- Lookups
- Node timepoint values
- Arithmetic operations
  - Add, subtract, multiply, divide
  - Square root, absolute value
- Arrays: size, element index, elements (for numeric arrays)

Standard Plexil – Numeric Expressions (continued)

- Examples

```
234
12.9
X (where X was declared Integer)
Bar (where Bar was declared Real)
LookupNow ("ExternalTemperature")
TakePicture.EXECUTING.START   (a node timepoint)
Bar + 4.5
X - (30 + LookupNow("x") )
3 * X
(3 * X)/(X - 20)
sqrt(X)
abs(X)
Entries[X] (where Entries is an array of integers)
```

- Integers and reals can be mixed in many operations (semantics intuitive, but needs documentation!)

Standard Plexil – Boolean Expressions

- Boolean literals
  - `true, false`
  - PLEXIL has a three-valued Boolean logic, which adds `UNKNOWN`, but this is not a valid literal
    - Use `isKnown` operator to detect `UNKNOWN`.
- Boolean-typed variables
  - `Boolean flag = false;`
  - `StartCondition: flag;`
- Lookups that return a Boolean-valued state
- Array elements (of Boolean arrays)

Standard Plexil – Boolean Expressions (continued)

- Comparison
  - Equal, not equal

```
Postcondition: attempts == successes;
Precondition: arm_status != engaged;
```

  - Less than, greater than (or equal)

```
StartCondition: temperature < 70;
InvariantCondition: altitude > 4000;
PreCondition: LookupOnChange(“score”) >= 10;
Precondition: LookupNow(“tachometer”) < 6500;
```

Standard Plexil – Boolean Expressions (continued)

- Operations (NOTE: these are not “short circuiting”)

Negation (not):	`!`
Disjunction (or):	`! !`
Conjunction (and):	`&&`
Exclusive Or:	`XOR`

- Examples

```
StartCondition: ! LookupOnChange(“engine_on”);
StartCondition: temp > 100 || rpm > 6000
StartCondition: score < 10 && my_turn
Assignment: result = (x > 10) XOR (y > 10)
```

Standard Plexil – Boolean Expressions (continued)

- Examples

```
True
False
CommandReceived (where CommandReceived was declared Boolean)
LookupOnChange("Rover:initialized")
count <= 30 (where count was declared Integer)
LookupNow("Rover:batteryCharge") > 120.0
! CommandReceived
LookupOnChange("Rover:initialized") || CommandReceived
Flags[3] (where Flags is an array of Booleans)
isKnown(val) (where val is any variable)
node3.state == FINISHED && node3.outcome == SUCCESS
```

Standard Plexil – String Expressions

- Evaluate to strings
  - Literal strings (double quoted, as in “hello”)
  - Variables of type string
  - Lookups
  - String concatenation (+)

- Examples

```
"foo"
"Would you like to continue?"
Username (where Username was declared string)
LookupNow("username")
"Hello, " + "Fred"     => "Hello, Fred"
"Hello, " + Username
```

Standard Plexil – Arrays

- Example

```
{
  // array of 10 Booleans
  Boolean flags[10];

  // array of 6 integers, with X[0]=1, X[1]=3, X[2] = 5.
  // X[3] through X[5] are UNKNOWN.
  Integer X[6] = #(1 3 5);

  Assignment: X[3] = X[2] + 1;
}
```

Standard Plexil – World State

- Obtained through *lookups*
  - `LookupNow (<state_name>)`
    - Immediate (poll)
    - Valid only in check conditions (pre, post, invariant) and action node bodies
  - `LookupOnChange (<state_name>, <tolerance>)`
    - Tolerance optional, defaults to 0
    - Value returned when state changes (subscribe)
    - Valid only in gate conditions (start, end, repeat, skip)

Standard Plexil – World State (continued)

- Example

```
HeatRoom:
{
  StartCondition: LookupOnChange("Temperature") < 70;
  Postcondition: LookupNow("Temperature") > 70 &&
                 LookupNow("Temperature") < 74
  Command: RunHeaterCycle();
}
```

Standard Plexil – Node State

- Consists of:
  - Current execution state
  - Start and end times of each execution state
  - Outcome of finished nodes
  - Failure type of failed nodes
  - Last command handle, for command nodes
- Accessible only for current node and its parent, children, and siblings.

Standard Plexil – Node State (continued)

```
Root:
{
  EndCondition: Bar.state == FINISHED;
  PostCondition: Bar.outcome == SUCCESS ||
                 Foo.failure != INVARIANT_CONDITION_FAILED;
  NodeList:
    Foo: { ... }
    Bar:
    {
      StartCondition:
        Foo.command_handle == “COMMAND_ACCEPTED” &&
        Foo.EXECUTING.START > 300.0;
    }
}
```

This says: Root ends when Bar is finished; Root is successful if Bar is successful, or Foo failed while maintaining its invariant; Bar starts when Foo's command has been accepted, and Foo started executing sometime after time 300.

Standard Plexil – Compiling into XML

- By convention, Plexil files have extension `.ple`
- Files must contain a single plan.
- Plexil files are translated into XML with the `plexil` command

```
plexil foo.ple
```

  - The resulting file is `foo.plx`
  - Errors and warning will get printed if there are problems. Fix them and try again!