PrTNet Gromet Specification

PrTNet Gromet is a JSON based representation of stochastic Petri nets. Abstractly, a stochastic Petri net is a collection of *state* variables, and a collection of *transitions*, which specify how the state variables may change. Each state variable has an initial condition, and the format also supports *parameterized* Petri nets, used to describe a family of related Petri nets.

The specification uses the following concepts:

- *junctions* are used to represent the states of the Petri Net
- *boxes* are used to group conceptually related entities; in particular the entire model is a box containg the junctions for the state variables, and boxes for the transitions
- *ports* are used to mark inputs or outputs of boxes; data only flows in and out of boxes through ports
- *wires* are used to specify how data flows around the system; wires may connect two ports, or a port and a junction:
  - a wire that flows into a junction indicates setting the value of a state variable
  - a wire flowing out of a junction indicates getting the value of a state variable

Top Level Object

A PrTNet Gromet is JSON object with the following form:

```
PRTNET :=
  { "syntax"    = "Gromet"
  , "type"      = "PrTNet"
  , "name"      = STRING
  , "metadata"  = null          // XXX: Format to be determined
  , "uid"       = STRING
  , "root"      = BOX_ID        // The top-level box for the model
  , "ports"     = [ PORT ]
  , "wires"     = [ WIRE ]
  , "junctions" = [ JUNCTION ]
  , "boxes"     = [ BOX ]
  }
```

Unique Identifiers

The specifications uses various forms of unique identifiers, which are all strings, but we give them explicit names to indicate what entities they identify.

```
BOX_ID      := STRING
```

```
PORT_ID     := STRING
WIRE_ID     := STRING
JUNCTION_ID := STRING
```

Note that these identifiers are unique for a given PrTNet object, but are *not* globally unique.

Value Types

Most entities also contain a field indicating its type, which may have be one of the following:

```
VALUE_TYPE :=
    "T:Real"                // The type of real numbers
  | "T:Bool"                // The type of booleans
```

Junctions

Junctions are used to represent the states of the Petri Net. They will only appear in the top-level box for the model. Each junction will have the following connections:

- A wire from each event box that may modify the state to the junction
- A wire from an *expression box* to the junction, specifying the initial state of the variable
- A wire from the junction to each *event* box that needs to use the value of the variable.

```
JUNCTION :=
  { "syntax"     = "Junction"
  , "uid"        = JUNCTION_ID
  , "type"       = "T:State"
  , "value_type" = VALUE_TYPE
  }
```

Wires

Wires are used to specify how data flows around the model.

```
WIRE :=
  { "syntax"      = "Wire"
  , "uid"         = WIRE_ID
  , "type"        = "T:Directed"
  , "value_type"  = VALUE_TYPE
  , "metadata"    = null                     // XXX
  , "src"         = PORT_ID | JUNCTION_ID
  , "tgt"         = PORT_ID | JUNCTION_ID
  }
```

Ports

Ports are used to specify inputs and outputs of a box. Input ports are of type `"T:Input"` while output ports are of type `"T:Output"`. An exception to this is that the ports for the outermost model box are of type `"T:Parameter"`.

```
PORT :=
  { "syntax"     = "Port"
  , "type"       = "T:Parameter" | "T:Input" | "T:Output"
  , "name"       = STRING
  , "metadata"   = null        // Format to be determined
  , "value_type" = VALUE_TYPE
  , "uid"        = PORT_ID
  , "box"        = BOX_ID      // Owner of the port
  }
```

Boxes

Boxes are a generic components used to group related entities. There are two flavors of boxes:

- *relation* boxes are used for grouping, and may contain other boxes
- *expression* boxes contain a tree describing a mathemtical expression

```
BOX := REL_BOX | EXPR_BOX
```

Relation Boxes

Relation boxes have the following format:

```
REL_BOX :=
  { "syntax"    = "Relation"
  , "type"      = "PrTNet" | "T:Event" | "T:Enable" | "T:Rate" | "T:Effect"
  , "uid"       = BOX_ID
  , "name"      = STRING
  , "wires"     = [ WIRE_ID ]      // Wires contained in this box
  , "boxes"     = [ BOX_ID ]       // Nested boxes
  , "ports"     = [ PORT_ID ]      // Ports for the box
  , "junctions" = [ JUNCTION_ID ]  // Nested junctions
  }
```

Relation boxes are nested as follows:

- The only box of type `"PrTNet"` is the outermost box containing the whole model
- The `"PrTNet"` box may contain only `"T:Event"` boxes and expression boxes; The expression boxes at this level are used for the inital values of the state variables, as well as to name expressions that may be used in multiple places in the model

- An "T:Event" box is going to have exactly 3 nested boxes:
  - One of type "T:Enable"
  - One of type "T:Rate"
  - One of type "T:Effect"
- A "T:Enable" box contain a single expresison box, describing the enabling condition for the given event
- A "T:Rate" box contains a single expresison box, descriing the rate at which this event may occut
- A "T:Effect" box contains a collection of expression boxes, one for each variable that may be affected by the event.

Only "T:Event" and "T:Effect" boxes have output ports, which are used to show how state variables are affected by events.

Expression Boxes

Expression boxes contain a mathematical expression. Some have an output port, which is used to indicate where the expression is used. Others (e.g. in "T:Enable" and in "T:Rate") do not have an output port as the expession is only part of the specification.

```
EXPR_BOX :=
  { "syntax"    = "Expression"
  , "uid"       = BOX_ID
  , "name"      = STRING
  , "tree"      = EXPR              // Exp
  , "ports"     = [ PORT_ID ]       // Ports for the box
  , "wires"     = [ ]
  , "boxes"     = [ ]
  , "junctions" = [ ]
  }
```

Mathemtical expressions are encoded as follows:

```
EXPR := LIT_EXPR      // A literal
      | PORT_ID       // Link to a port
      | CALL_EXPR     // Use of an operator

LIT_EXPR :=
  { "type"      = VALUE_TYPE
  , "value"     = STRING           // Value for the literal
  , "metadata"  = null             // Format to be determined
  }

CALL_EXPR :=
  { "syntax"  = "Expr"
  , "call"     = OPERATOR
  , "args"     = [ EXPR ]
  }

OPERATOR :=
  { "syntax" = "RefOp"
  , "name"   = "not" | "exp" | "log"
             | "+" | "-" | "*" | "/"
             | "lt" | "leq" | "=="
             | "and" | "or"
             | "if"
  }
```

Example

The following picture is a visualization of a PrTNet document. The ports at the top part of box are input ports, while the ports at the bottom are output ports.

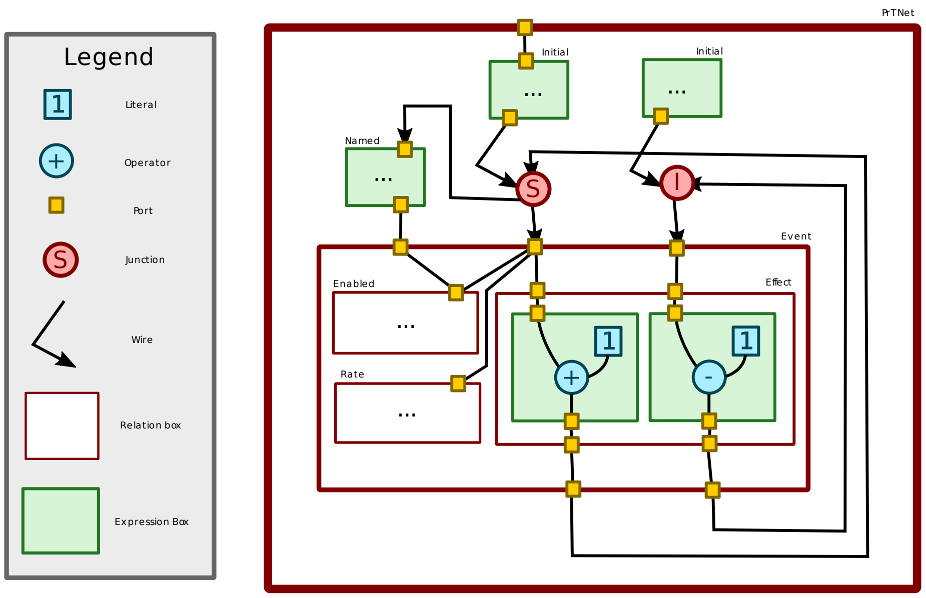

Figure 1: Example