

---

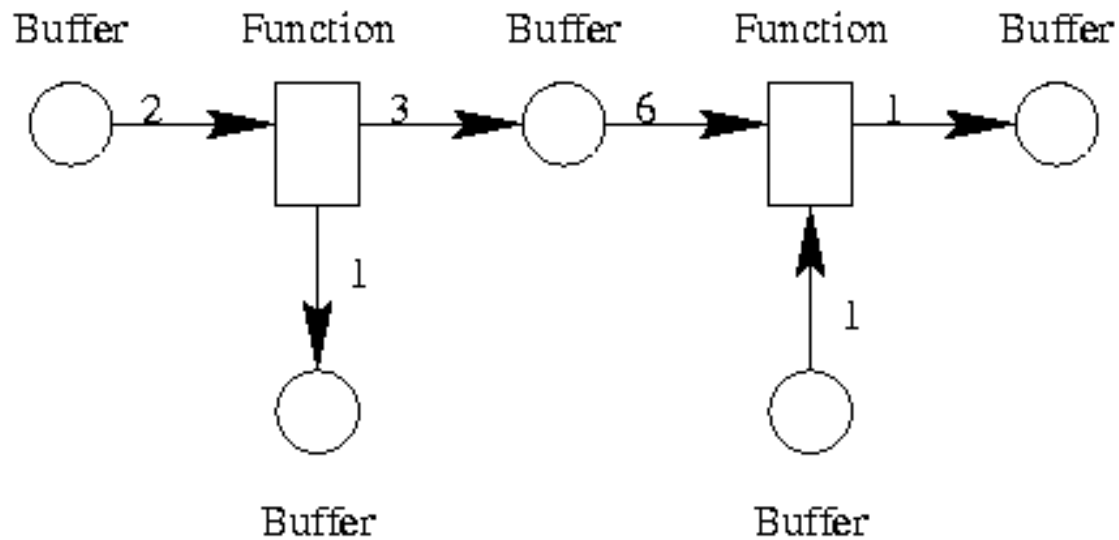
# Extending the Editor

Extending the Moses Editor to Accommodate the SDF  
Formalism

# What is SDF?



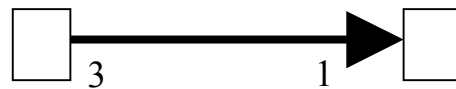
- ◆ Synchronous Data Flow
- ◆ Lee and Messerschmitt 86.
- ◆ Allows functional elements to be *statically* scheduled.



# SDF - Functions and Buffers



- ◆ Buffers are unbounded FIFO queues.
  - 1 input & 1 output per buffer.
  - L&M's SDF assumes implicit buffers on arcs
    - » buffers aren't explicitly depicted on arcs.



Implicit Buffer



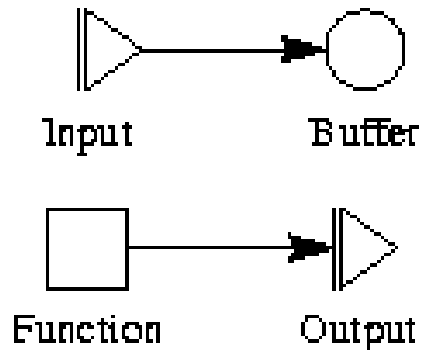
Explicit Buffer

- ◆ Markings (weights) on arcs represent # of tokens produced/consumed when the function attached to each end of the arc fires.
  - This is **NOT** optional - tokens must be consumed/produced from input/output arcs in the marked amounts when an attached function fires.
  - Functions can have an arbitrary number of input and output arcs.
- ◆ Functions may be scheduled in any order (at any time) after their firing conditions are met.
  - i.e. sufficient tokens on input buffers.

# SDF- Inputs and Outputs



- ◆ We wish to create Moses Hades components.
  - We need inputs and outputs!
- ◆ Inputs can produce tokens at any time.
  - Only 1 buffer can be connected to an input.
- ◆ Outputs can consume tokens at any time.
  - Only 1 function can be connected to an output.
- ◆ Arcs to/from outputs/inputs may have arbitrary weights.



# SDF- Black Token



- ◆ We will implement black token SDF.
  - Tokens are un-typed, and have no value.
  - Functions perform no computation.
    - » Simply produce and consume the appropriate number of tokens.

# SDF- Element Types



- ◆ Within SDF we can identify the following entities.
  - Functions
  - Buffers
  - Arcs
  - InputConnectors
  - OutputConnectors
- ◆ We wish to build a graph out of these entities. In our graph, each element will have an associated state.

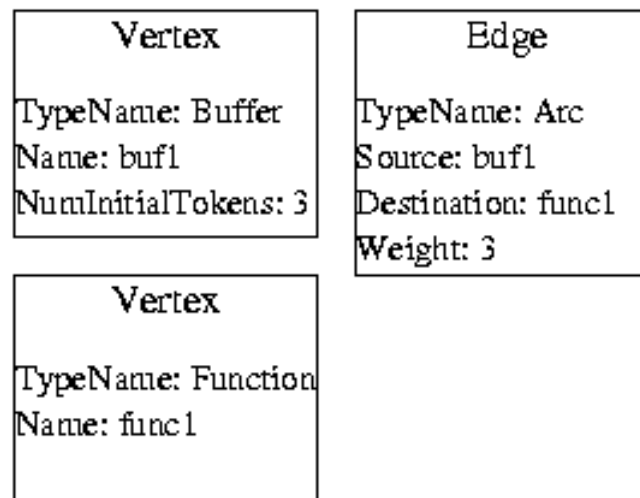


- ◆ The preceding entities can be broken into 2 types.
  - Edges (Arcs).
    - » Each edge connects 2 vertices.
  - Vertices (Functions, Buffers, InputConnectors & OutputConnectors).
- ◆ All elements (edges and vertices) have an associated TypeName (text).
  - E.G. text typename = “Buffer”.
- ◆ Each element (vertex or edge instance) can also have a name (text).
  - E.G. text name = “Buf1”
- ◆ Other State
  - Functions
    - » No associated user provided state.
      - ◆ Can be enabled - this isn't user provided - attained from environment.
  - Buffers
    - » Initial number of tokens can be set by the user (integer).
  - Arcs
    - » The arc weight can be set by the user (integer).

# Edges and Vertices



- ◆ An SDF graph can be entirely represented in a simple graph structure, built out of Edge and Vertex types.
  - Edges and Vertices have attributes describing:
    - » their element type (I.e. Buffer) (text).
    - » Name (text).
    - » Other attributes: (string, value) pairs.
  - Additionally, Edges have attributes describing:
    - » source and destination vertices.



# What does the Editor Produce?



- ◆ The Moses editor produces a graph . Graph object.
  - A simple structure containing a list of vertices and edges.
  - Also contains a set of graph attributes.
    - » E.G. string comment = “Our First SDF Graph”.
- ◆ This is all!
  - Graphs have no semantic meaning for the editor.
  - It is the task of components that consume the graph to infer their own meaning.
    - » Wait until we extend the simulator for this.

# What does the Editor Consume?



- ◆ **All formalism specific information is provided to the Editor by 2 files.**
  - The formalism's GTDL file.
  - The Editor Properties file.
- ◆ **These files must provide a complete description of SDF syntax.**
  - All edge and vertex types, with all associated attributes for each edge and vertex.
  - A mechanism for stating syntactical rules (e.g functions connect to buffers or outputs).
- ◆ **The editor also needs a graphical description of each vertex and edge type.**
  - This *is* graphical editing.
- ◆ **Finally, the editor requires a set of actions, tied to menu items and toolbar buttons.**
  - E.G. Insert a Buffer.
- ◆ **The Editor doesn't use the GTDL file directly - it is compiled by the GTDL compiler before Editor use.**

# Contents of the GTDL File



- ◆ To produce graphs, the Moses editor requires:
  - A name for the graph type.
  - A list of attributes for the entire graph
    - » i.e a textual comment

```
graph type SDF {  
    attribute text Comment.
```
  - A description of each *Vertex* and *Edge* type.
    - » Name of the type.
    - » Attributes (e.g. integer NumInitialTokens)
    - » Graphical description (we need to display vertices and edges on screen).
- ◆ We have the following Vertex and Edge types:
  - Buffer
  - Function
  - Arc
  - InputConnector
  - OutputConnector
  - Comment: A dummy vertex that **ONLY** displays text. It has no meaning for SDF.

# Vertex and Edge Descriptions



- ◆ Vertex Example:

```
vertex type Buffer (string Name, integer NumInitialTokens)
  graphics (hidden string Shape = "Oval",
    hidden color Color = "black",
    hidden color FillColor = "yellow",
    hidden integer ExtentX = 24,
    hidden integer ExtentY = 24).
```

Attributes

Graphical  
Attributes

- ◆ Edge Example:

```
edge type Arc (string Name, integer Weight)
  graphics (hidden string Head = "ClosedTriangle",
    hidden color Color = "black",
    hidden color FillColor = "black").
```



## ◆ Syntax Predicates

- Syntax checks in the editor are desirable.
- Editor checks a set of predicates after any graph modification.
- Predicates written in ELAN (Moses Expression Language).
- E.G.

```
predicate OneInputPerBuffer
  "Each buffer is only allowed one input"
  forall e1 in Arc, e2 in Arc :
    if (dst(e1) = dst(e2)) then
      ((e1 = e2) || ~(e1 in Buffer))
    else
      true
    end
  end
end
```



# Connector Handling



- ◆ The Moses editor allows the composition of components.
  - Must be able to extract the connectors from a Discrete Event Component without instantiating it.
  - The ELAN expression language is used.

```
declare {
  inputs = lambda (s)
    let ClassLoader = new moses.util.ClassReloader() :
    if (s <> null) && (s <> "") then
      let cd = new hades.DES.components.
        ComponentDescriptor(ClassLoader'loadClass(s)) :
      if cd'isDEC() then
        (cd'getSignature())'getAllAtomicInputConnectorNames()
      else
        {}...
```



- ◆ A final piece of information in the GTDL file.
  - Hooks that provide semantic meaning to the graph.
- ◆ Not necessary for graph **editing** (except EditorProperties).
  - Used to create simulation components, animate components, etc...
- ◆ EditorProperties
  - A file that provides a set of editor properties (menu-action bindings, etc...).
- ◆ Compiler
  - A compiler implements `moses.models.translator.ModelCompiler`
  - Allows a Graph to be compiled into a Java class implementing `DiscreteEventComponent` (the simulation component interface).
- ◆ AnimationAdapter
  - AnimationAdapters implement the `moses animator.AnimationAdapter` interface.
  - Allows `DiscreteEventComponents` to transmit state change events to the AnimationAdapter, which are displayed on-screen.

# Editor Properties



- ◆ Name of Editor Properties file specified in EditorProperties in GTDL file.

- Exists in the Moses code directory.
- Name has “.properties” appended.
- Specifies a menu bar (called insert) and a tool bar.

```
menubar=insert
```

```
insert=comment - arc - buffer function inputconnector outputconnector
```

```
toolbar=^ - comment - arc - buffer function inputconnector outputconnector
```

- Note: The ^ symbol means that all following buttons are *exclusive radio buttons* (only 1 can be selected at a time).

- Specifies a set of bindings

- » menu/button -> action

- » menu/button -> image

- » menu/button -> label

```
arcLabel=Arc
```

```
arcImage=images/petri-images/arc.gif
```

```
arcAction=ArcAction
```

# Exercise



- ◆ We have provided an SDF file and an EditorProperties file.
  - Run Moses using the Basic.rep repository.  

```
java -jar Moses.jar -d Basic.rep
```
  - The SDF file has an entry for *Function* missing.
    - » Similar to the Transition defined in the TimePetriNet GTDL file.
      - ◆ However, only has ‘Name’ as an attribute.
  - There are also 2 predicates missing (one output per inputconnector) and (one input per outputconnector). See if you can add these.
- ◆ Follow the available tutorial to add a new formalism to SDF.
  - Make your GTDL and optional EditorProperties modifications.
- ◆ You can now build SDF graphs using the moses editor.