Flow BP

Software Engineering Project
Maintenance Guide

Eilon Benami
Eyal Almog
Hadas Atiya
Gilad Abudi

Contents

Chapter 1 - Interpreter Maintenance

Adding external events:

In the **BPEngine.js**' **run** function, in case the function receives a null event, it means there are no events left to choose, and the function will yield until receiving a new event. In order to use external events, the StartRunning function in **rungraph.js** should support the external events mechanism by waiting for the external event after receiving the yield message from the BPEngine. After receiving the external events, the function should use the **next()** function for the BPEngine's run function generator.

Changing the multiple output mechanism:

_The entire mechanism of handling multiple outputs lies in the **GoToFollowers** function in **rungraph.js.** The function iterates over all the edges connected to the cell sent as a parameter to the function, and determines which payloads are sent to which outputs, and how many new BThreads are created.

Editing the behavior of the interpreter when reaching a node:

_The functions **runInNewBT** & **runInSameBT** in **rungraph.js** are responsible for handling the interpreter behavior while reaching a node, according to BP semantics and the functionality needed. **runInSameBT** is responsible for handling the continuation of some Bthread execution (meaning that the BThread was started on a different node). **runInNewBThread** is responsible for initiating a BThread execution. It registers a new BThread, with a function that is somehow similar to **runInSameBT**, only it refers to the first node in that BThread's flow of execution (scenario). One can change the interpreter behavior while reaching a node by changing one of these functions.

Changing transition of payloads between nodes connected with an edge:

_The function **GoToFollowers** in **rungraph.js** is responsible for the transition between nodes, and the continuation of a BThread's execution, while starting new BThreads if there are multiple edges going out of the cell currently being processed in the function. That is the function to go to in this case.

Changing the initial execution of a BP Flow program:

_The function _responsible for initiating the whole BP Flow program execution is the **StartRunning** function in **rungraph.js.** The function registers Bthreads according to the number of Start Nodes, and starts the BpEngine. In order to add conditions for executions or to change the initiations, this is the function to go to.

Chapter 2 - Debugger Maintenance

Intervals of the step-by-step execution:

In order to allow change of the time intervals on debug mode, a change is needed only in **getProgramRecord** function (or creating a similar one) in **..\editor\js\debuggerBP.js** file.

getProgramRecord function is implemented as single time unit record, where each step represents the execution state at its time unit (see getProgramRecord notes).

Syncing point intervals:

At time unit where eventSelected is not null, the execution is at syncing point, so the execution record should contain only these states.

For example, if the record looks like that:

```
rec = [{stages: {2: [{"X": 4}, {"Y": 6}]},
         eventSelected: null,
        blocked: {},
        messages:null,
        syncing: {}},
      {stages: {3: [{"X": 3}, {"Y": 4}], 4: [{"X": 5}, {"Y": 7}]},
        eventSelected: null,
        blocked: {},
        messages:["Hello", "World"]],
        syncing: {}},
      {stages: {3: [{"X": 3}, {"Y": 4}], 4: [{"X": 5}, {"Y": 7}]},
        eventSelected: null,
        blocked: {},
        messages: null,
        syncing: {3, 4}},
      {stages: {3: [{"X": 3}, {"Y": 4}], 4: [{"X": 5}, {"Y": 7}]},
        eventSelected: "X",
        blocked: {},
        messages: null,
        syncing: {3, 4}},
      {stages: {3: [{"X": 3}, {"Y": 4}], 4: [{"X": 5}, {"Y": 7}]},
        eventSelected: null,
        blocked: {5}},
        messages: null,
        syncing: {}}]
```

picking only rec[3] will results:

```
newRec = [{stages: {3: [{"X": 3}, {"Y": 4}], 4: [{"X": 5}, {"Y": 7}]},
           eventSelected: "X",
           blocked: {},
           messages: null,
           syncing: {3, 4}}]
```

and now the record represents the execution states at each syncing point (where this example shows only one step).

External events insertion while debugging a program:

In order to allow insertion of external events while executing a program in debug mode, several changes are needed:

1. In **..\Rungraph\BPEngine.js** file, the function **BPEngine.prototype.getEvent** is choosing a random event from a set of events that are legal to be selected at this point (requested events which are not blocked). The change needed in this function is one that allows the function to choose from external data structure which contains events that where inserted somehow.
2. In **..\Rungraph\BPEngine.js** file, in function **BPEngine.prototype.run**, the execution recording should be terminated (line of code- this.deb.endRecord()) without regarding the selection of an event, and needs to be performed before the event selection.
   It should look like that:
   ```
   if(this.deb!=null) {
       this.deb.endRecord();
   }
   **
   let e = this.getEvent();
   ```
3. After performing 1 and 2, the **getProgramRecord** function should return a record of the execution up to the time unit where an event has been selected. Now in line ** the program record should yield, after saving the record up to this point (by calling **getProgramRecord** function). Make sure that the data of the debuggerBP instance (deb) is initialized properly afterwards (don't use **initDebug** function, implement a new one where the fields scenarios, messages and the rest are set to values which won't cause problems on re-executing **run** function).
4. Now in **..\editor\js\debuggerBP.js** file, the function **debuggerBP.prototype.next** is responsible for making steps forward on debugging mode. When reaching the end of the debugging (line of code-
   else if (this.editor.undoManager.indexOfNextAdd == this.editor.undoManager.history.length))
   the **run** function (in **BPEnging.js** file) should be called again and update the record of the execution after picking an event to occur.

5. When reaching the ** again, it means all bThreads are at syncing point, and now **getProgramRecord** function should return the execution record relevant to the time between the selection of the previous event and now (if you have initialized **deb** properly).
6. Finally, implement a function that updates the execution steps on the working sheet and the undoManager (see **debuggerBP.prototype.setDebuggingSteps** function in **..\editor\js\debuggerBP.js** file), but make sure you're doing it the right way, regarding the undoManager state at this point.

Expanding the program execution record information:

In **..\Rungraph\rungraph.js** file you can find the algorithm of performing an execution of BP program on flow chart based on mxGraph. Wherever **bpEngine.deb** (or **this.deb** in **..\Rungraph\BPEngine.js** file) is performing an action, it means that information at this point is being recorded. So, if you want to record any other information you can do it the same way, by adding other attributes to **debuggerBP**, just make sure your new attribute is recording respectively to the existing attributes. For example- **messages** and **syncing** attributes are being fixed to the current time unit of the execution when calling **fixStages** function, and i'th index inside them represent their corresponding data at time i.

Chapter 3 - Editor Maintenance

Adding a new kind of node(block) to the left-handed sidebar (Functional Requirement 1.2):

Adding a new node requires two steps:
1. Create a new shape in the file ShapesBP.js
2. Add the shape to the left side bar in the class SideBarBP

File: ShapesBP.js
- define a new shape function:

function *Shape_function_name*(){ *it is recommended to extend existing shapes from file Shapes*.js}
- register a new shape

mxCellRenderer.registerShape(*shape_name*, *Shape_function_name*);
File: SidebarBP.js
Functions:
1**. setCellAttributes**(value, geometry, style, type, bpCell, visible):
initial cell attributes
2. **createBPShape**(name, shape)
initial bp cell children (all bp shapes except start nodes contain 4 children):

a. divider: a line that is used to divide between the headline and the data
b. data: data of the bp node
c. divider2(visible only on debug mode): a line that is used to divide between the data and the payload
d. payloads (visible only on debug mode): the payload of the bp node

3. **addFlowBPPalette()**
insert all shapes into the sidebar.

Changing the number of outputs in a general node (Functional Requirement 1.12):

user choose output number of the general node.
File: FormatBP.js
Functions:
1. **computeNewPosition(cell, numOfOutputs, index)**
compute location of the connection point
2. **updateConnectionPoints(cell, numOfOutputs, graph):**
initially, new_constraints is filled with the new connection constraints by numOfOutputs (output constraint first, input constrainst last).
3. **adjustEdges(cell, numOfOutputs, graph):**
adjust output edges to fit new connection point locations when changing the number of outputs of the node.
File: GraphBP.js

Functions:
1. **getAllConnectionConstraints(terminal, source):**
use new_constraints of terminal if it exists.

Add labels for edges coming out of general nodes (Functional Requirement 1.13):

File: FormatBP.js
1**. deletePrevLabels (cell, value, graph)**
delete old labels of out edges
2. **updateLabelsPositions(graph, cell, newOutputNumber):**
Adjust output labels positions to fit new connection point locations when changing output number of node.
3. **updateEdgesLabels(cell, graph, cellValue):**
update output edges labels.
4. **updateOutputsLabels(graph, cell, labels):**
update output labels.
File: mxClient.js
Functions:
1. **fixConnectionPointsLabelLocation(Cell):**
relocate outputs labels according to connection points locations.

Set connection points of cells visible (not only while hovering on a cell):

In MxGraph, connection points are visible only while hovering on a shape.
In our project we want that the connection points will appear when adding new shape, And to distinguish between input and output connection point.
File: mxClient.js
Old attributes refactor:
focusIcons – refactored into a dictionary instead of a list. {key: cell id; value: connection point image}
Functions:
1. **getImageForConstraint(state, constraint, point**): **override**
Return connection point image by the connection point name.
2. **Reset(): override**
Reset shape focus and cancel deletion of connection points images
3. **destroyIcons(): override**
Cancel deletion of connection points images
4**. destroyIconsByState(state):**
delete connection points images of the input cell state (used after deletion of the cell)
5. **update(me, source, existingEdge, point): override**
Set focus on cell when hover a cell or hover the connection point of the cell.
6. **getConstraintLocation(state, constraints, size):**
get the connection point image location by the constraint.
7. **redraw(): override**
redraw all shapes connection points images.

8. **setFocus(me, state, source):** override
- Set the focus on the state cell when hovering on it.
- when hovering a shape as a target, hide input connection points.
9. **showConstraint(inputState):**
draw connection points of inputState if given, otherwise draw all shapes connection points.
File: Graph.js
Functions:
1**. shapeContains(state, x, y):** override
check if (x, y) contains a bp shape or its connection points

Adjust the right-sidebar toolkit format for the selected shape on the board:

File: FormatBP.js
Function: refresh
you can know from the "graph" global variable, what is the selected element on the board (edge, node or noting).

- If it is a vertex (node) you can get its element with this line of code:
  cell = **graph.getSelectionCell()**
  - for recognizing the selected node's type, use this line of code:
    cell.bp_type
  - To access the right-handed sidebar HTML element use this line of code:
    document.getElementsByClassName("geFormatContainer")[0];

General node number of outputs:

File: FormatBP.js
Function: **refresh()**
In this function HTML elements are created for the general node in the right-handed toolkit.

- The general node has an attribute called "NumberOfOutputs", this attribute is used to update the HTML element variable - NumberOfOutPutBox.
- NumberOfOutPutBox has the attributes “max” and “min” that their values represent the limit of the number that can be selected in the HTML “INPUT” element in the right-sidebar toolkit.

General node title:

File: formatBP.js
Function: refresh()
In this function HTML elements are created for the general node in the right-handed toolkit.

- The general node has an attribute called " NodeTitle this attribute is used to update the HTML element variable - titleBox.

Add new attribute for node that update from the right-sidebar toolkit:

File: FormatBP.js
Function: refresh
In this function HTML elements are created for the general node in the right-handed toolkit.
Identify the node that you want to add a new attribute for, with this line of code:
cell.bp_type

- Create the HTML elements that will serve as placeholders for the data given from the user.
- Add the HTML elements to the right-sidebar toolkit, to get this element use this line of code:
  var cont = document.getElementsByClassName("geFormatContainer")[0];

  for adding something to it use this line: **cont.appendChild(NEW_HTML_ELEMENT);**

- For updating or creating the node new attribute, use this line of code:
  cell.setAttribute("ATTRIBUTE_NAME", NEW_HTML_ELEMENT.value);
- After the attribute is updated on the cell you need to update the graph model, there are 2 options to do that:
  - Put the cell update attribute line between these lines:
    graph.getModel().beginUpdate();
    // all the updates that you made on the cells
    graph.getModel().endUpdate();
  - Use this line of code:
    graph.getModel().setValue(cell, value);
    when you choose this option, you need to update the cell value.
    Example:
    var value = graph.getModel().getValue(cell);
    value.setAttribute("ATTRIBUTE_NAME", NEW_HTML_ELEMENT.value);
    graph.getModel().setValue(cell, value);

Edge connection position:

File: mxClientBP.js
Function: checkAndFixBorder
Change the entry point of the edge on the target node, by changing the entryX and entryY in the style attribute of the edge. The style of the edge is loaded to a dictionary object – **stylesDic**. You can change the entryX or entryY values by using this line of code:
stylesDic["entryX"] = ___

Code Editor dialog:

File: DialogsBP.js
Function: CodeEditorDialog

- You can change the HTML element that is created for this dialog
- You can change the apply button function
- The code is being parsed using esprima package in order to detect syntax errors. Changing the package, removing it or adding another one is possible

BSync dialog:

File: DialogsBP.js
Function: BSyncFrom

- You can change the HTML element that is created for this dialog
- You can change the apply button action and the attributes of the Request/Wait/Block sections in the bsync node.

Enable/Unable connection of an edge to a start node (as a target):

File: mxClientBP.js
Function: terminalForCellChanged
you can determine if the target node is a “start node” with this line of code:
terminal.bp_type =="startnode"
and return the previous location of the edge.
Function: createEdge
you can determine if the target node is a “start node” with this line of code:
terminal.bp_type =="startnode"
and return null in order to cancel the option to create an edge that the target of this edge is a “start node”.

Enable/Unable creating an edge without a target node:

File: mxClientBP.js
Function: createEdge
check if the target node is null, and return null in order to cancel the creation of the edge.

Limit the amount of out edges:

File: mxClientBP.js
Function: createEdge
check the target or source nodes types by this line of code:
source.bp_type

- Get the number of out edges from the source node by this line of code:
  getNumOfOutEdges(source)
- For canceling the creation of the edge - return null.