**Flow BP**

Software Engineering Project

Application Design Document

Tomer Bitran

Shir Markovits

Shahar Hazan

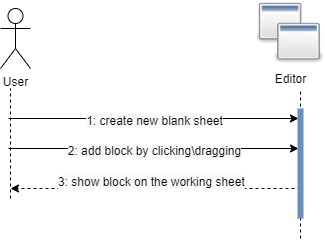
**Contents**

Chapter 1: Use Cases……………………………………………………………………………………………………………..2

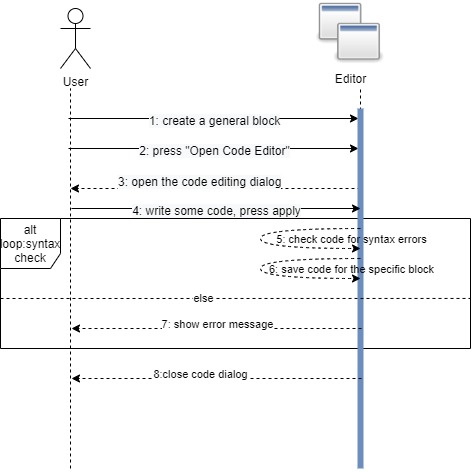
Chapter 2: System Architecture…………………………………………………………………..………………….…….12

# **Chapter 1- Use Cases**

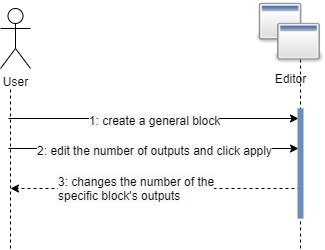
|  |  |
| --- | --- |
| Use-case name | Block addition |
| Description | The user opens the blocks pop-up menu by clicking the right mouse button and chooses a block. |
| Pre-conditions | None. |
| Post-conditions | The block will be visible on the working sheet. |



|  |  |
| --- | --- |
| Use-case name | Edit code section on a General/Console block |
| Description | After creating a general/console block, the user can edit the code on the block by clicking the block, then clicking the "Open Code Editor" button. After clicking the button, a new window with a text area will be opened, and in order to save the code the user should press the apply button. |
| Pre-conditions | A general/console block was created on the working sheet. |
| Post-conditions | The editor saves the code of the specific block. |
| Alternatives | If the user writes code with syntax errors, the editor will notify the user that an error has occurred, and allow the user to fix the code. |

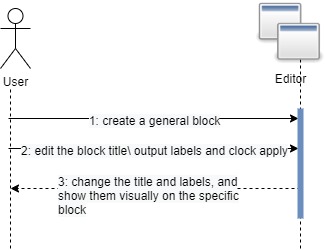


|  |  |
| --- | --- |
| Use-case name | Edit Number of outputs on a general block. |
| Description | After creating a general block, the user can edit the number of outputs the block has by clicking the block, then editing the "Number of outputs" field. Then pressing apply will change the number of outputs. |
| Pre-conditions | A General block was created on the working sheet |
| Post-conditions | The number of outputs on the specific block is changed accordingly |

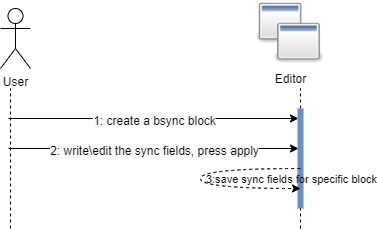


**3.1**

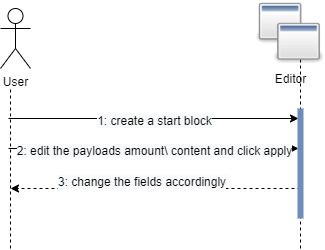
|  |  |
| --- | --- |
| Use-case name | Edit outputs labels \ block title on general block |
| Description | After creating a general block, the user can edit the output's labels and the block title by pressing the block and changing the field. |
| Pre-conditions | A general block was created on the working sheet |
| Post-conditions | The labels\title are showed visually on the block |



|  |  |
| --- | --- |
| Use-case name | Edit Request, Wait and Block fields on a Bsync block. |
| Description | After creating a Bsync block, the user can edit the Request, Wait and Block event fields on the block. |
| Pre-conditions | A Bsync block was created on the working sheet. |
| Post-conditions | The editor saves the sync fields of the specific block. |



|  |  |
| --- | --- |
| Use-case name | Edit Payloads on a start block. |
| Description | After creating a start block, the user can edit the number of payloads the block contains, and the payloads contents. The user clicks the block and edits the fields on the pop-up window. |
| Pre-conditions | A start block was created on the working sheet. |
| Post-conditions | The fields on the specific block are changed accordingly. |



|  |  |
| --- | --- |
| Use-case name | Loading a program |
| Description | The user is pressing the load-program button, picks a JSON file and opens it. |
| Pre-conditions | None. |
| Post-conditions | The working-sheet filled with the graph represented by the structure inside the JSON file. |

**תמונה שמכילה צילום מסך

התיאור נוצר באופן אוטומטי**

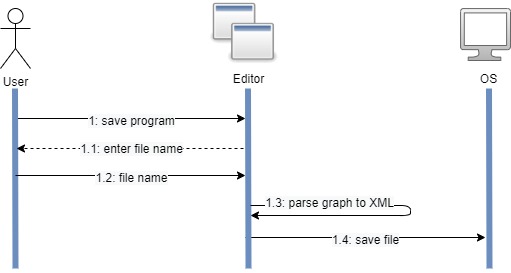
1.5: JSON file

1.6: loading JSON

to working sheet

**7.**

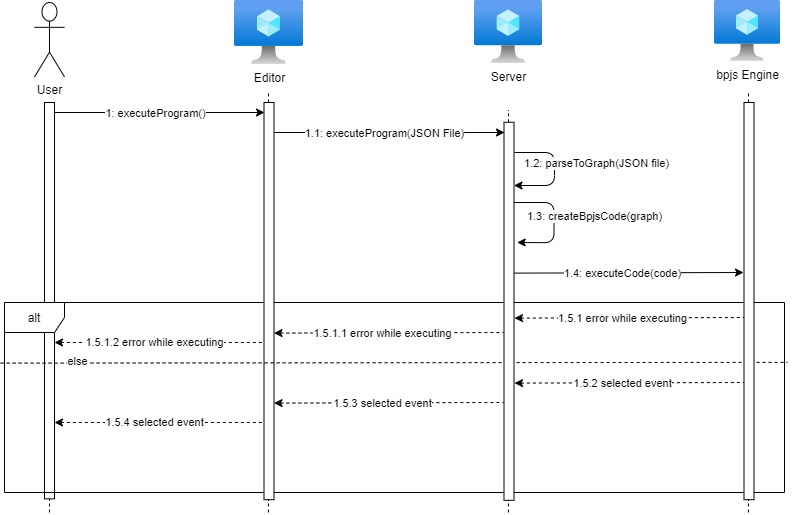
|  |  |
| --- | --- |
| Use-case name | Saving a program |
| Description | The user is pressing the save-program button, picks location and file name and saves it. |
| Pre-conditions | None |
| Post-conditions | An JSON file created on the selected location which contains all the graph details. |



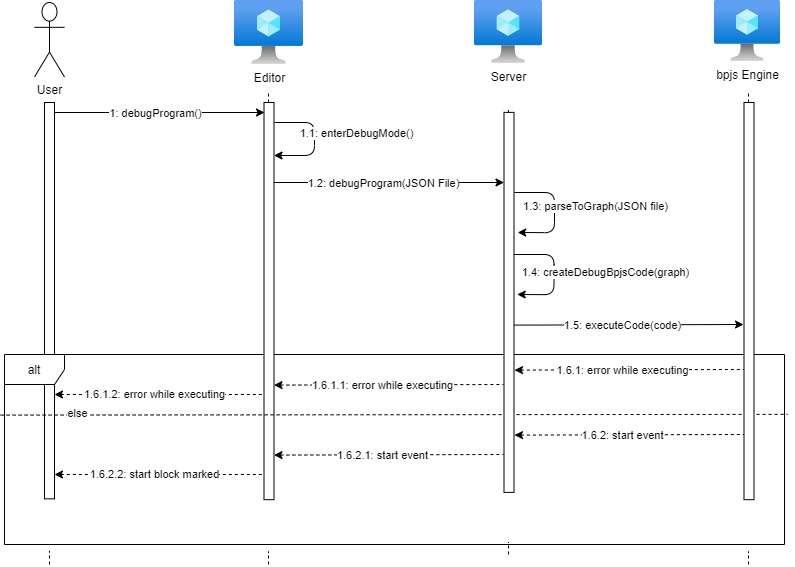
JSON

**8.**

|  |  |
| --- | --- |
| Use-case name | Executing a program |
| Description | Pressing the run button after a graphical BP Flow program is located on the working sheet. A JSON presents the graph being sent to the server.  The server parses and converts the JSON object to a graph-like object. The server creates a bpjs code file based on the graph structure. The server executes the bpjs code using bpjs server. |
| Pre-conditions | Loaded/Created graph bases on the working-sheet. |
| Post-conditions | The output console filled with events occurred according to the program restrictions. |
| Alternatives: | One of the general blocks' code section has code that breaks in run-time. The execution will stop, and an appropriate message will be shown to the user. |

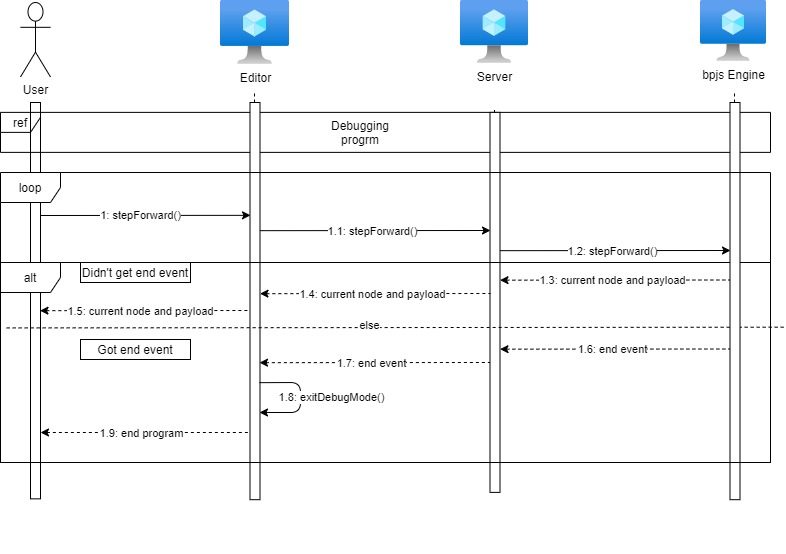


|  |  |
| --- | --- |
| Use-case name | Debugging a BP-Flow program |
| Description | After a graphical program is located on the working sheet, pressing the debugging button will transform the UI into debug mode. A JSON presents the graph being sent to the server.  The server parses and converts the JSON object to a graph-like object. The server creates a **debug** bpjs code file based on the graph structure. The server executes the bpjs code using bpjs server. |
| Pre-conditions | Loaded/Created graph bases on the working-sheet. |
| Post-conditions | Step Forward and stop buttons are available, action that changes the program semantics are blocked until exiting debug mode. Start block of the executing flow is marked. |
| Alternatives: | There was an error while executing the code. an appropriate message will be shown to the user |



**9.1**

|  |  |
| --- | --- |
| Use-case name | Step forward in debug mode. |
| Description | While in debug mode, the user can click on the step – forward button, to see the next step of the program that lies on the working sheet. |
| Pre-conditions | Debug button was clicked, and the editor is in debug mode. |
| Post-conditions | The next step of the program execution will be visible to the user. |
| Alternative | The bpflow program has reached the end of the execution, and therefore editor exits from debug mode. |



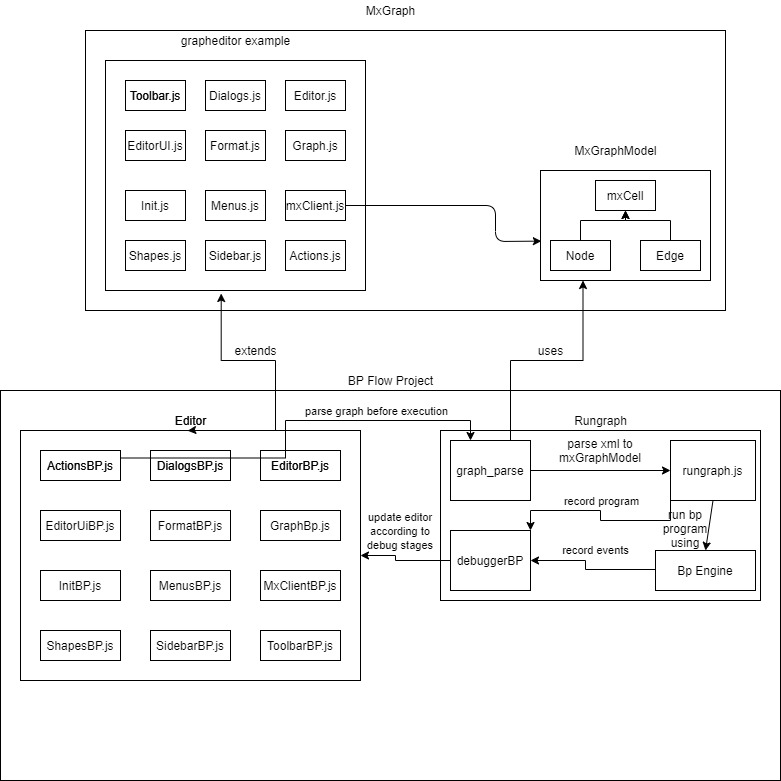
**Chapter 2**

# **System Architecture**

The system is based on one of the MxGraph javascript graphical editor examples. MxGraph is an open – sourced repository for defining graphical objects, with a model called .

For more information about mxGraph:

* Main repository: <https://github.com/jgraph/mxgraph>
* Graph editor example (being used in this project): <https://github.com/jgraph/mxgraph/tree/master/javascript/examples/grapheditor>



1: MxGraphModel - A graph-like model in the MxGraph repository. Takes every shape and arrow from the graph and converts it into nodes and edges, using the MxCell object as a node\edge accordingly

The editor part of the project is responsible for the graphical additions and changes to MxGraph's "grapheditor" example.

Editor Files:

* **ActionsBP**.js – extends Actions.js of MxGraph. Responsible for defining different actions (Execution, debugging, etc.).
* **DialogsBP**.js – extends Dialogs.js of MxGraph. Responsible for defining dialogs and pop up windows opened (Code editor, Output console, etc.).
* **EditorBP**.js - extends Editor.js of MxGraph.
* **EditorUiBP**.js – extends EditorUi.js of MxGraph.
* **FormatBP**.js – extends Format.js of MxGraph. Responsible for the right-hand menu opened while clicking on a certain graphical object.
* **GraphBP**.js – extends Graph.js of MxGraph.
* **InitBP**.js – A new file(not an extension) some like similar to Init.js of MxGraph, but with the certain definitions and file paths needed for the current project to run correctly.
* **MenusBP**.js – extends Menus.js of MxGraph. Responsible for the upper menu ( File,Edit,View etc.).
* **MxClientBP**.js – extends MxClient.js of MxGraph. The main file which defines the entire MxGraph Model used for parsing diagrams and graphical objects.
* **ShapesBP**.js – extendsShapes.js of MxGraph. Responsible for the graphical objects' behavior.
* **SidebarBP**.js – extends Sidebar.js of MxGraph. Responsible for the left-hand menu where the shapes and graphical objects lie. In order to create a shape, you need to click it \ or drag it from that sidebar.
* **ToolbarBP**.js – extends Toolbar.js of MxGraph. Responsible for the lowest upper toolbar, with the icons. In this toolbar there are default MxGraph actions like undo and zoom in\out, and added actions like execute \ debug.

The RunGraph part of the project is responsible for Executing and Debugging BP Flow programs, which are created according to the syntax of the graphical objects created in the working sheet by the user.

Rungraph Files:

* **Graph\_parse.**js – responsible for converting the XML that describes the diagram into a MxGraph Model. When the user clicks the 'Execute' button, the action written in ActionsBP.js which is connected to that button, sends the xml to this file. Then the XML is being transformed into a MxGraph Model, and sent to rungraph.js for execution.
* **Rungraph**.js – Responsible for executing and debugging BP Flow programs. The file holds an interpreter inside, which traverses over the entire graph given by graph\_parse (MxGraph Model) and executes actions according to the BP semantics. Each scenario is handled separately. Events selected and String given from Console nodes are written into the console Window ( defined in DialogsBP.js)
* **BPEngine**.js – responsible for registering B threads, selecting events in every bsync point, and continuing bthreads execution according to the BP program flow
* **debuggerBP**.js – responsible for the BP flow debugger definition and functionality. When clicking on the debug button, this file runs the specified BP flow programs and records every step. On every step the debugger saves the state of each scenario's nodes which are relevant to the time of the step, and additionaly writes to the console the events selected in the specific step the debugger is currently on.