Flow BP

Software Engineering Project
Application Requirements Document

Group 4

Tomer Bitran
Shir Markovits
Shahar Hazan

Contents

Chapter 1

Introduction

1.1 The Problem Domain

"Behavioral programming is a novel, language-independent paradigm for programming reactive systems, centered on natural and incremental specification of behavior.

(Harel, D. , Marron,A. , Weiss, G. (2012). Behavioral Programming).

1.2 Context

Create a web-application based system, including a graphical editor.

1.3 Vision

- The main goal is to create a graphical IDE for creating and executing programs of BP flow, including a visualization of the execution by steps.
- Connecting the client-side web application to a BPjs server.
- Development with an emphasis on the user experience.
- Create a version control for the system's users.
- Share Flow-BP project between users.

1.4 Stakeholders

- Ben Gurion University BP research team.
- Programmers wishing to learn about and explore the BP paradigm.

1.5 Software Context

The system is a client-server browser-based web application, including a client as a web graph editor, and a BPjs based server.

The main user scenarios are:

1. BP flow program creation:
    - Create visual objects from a toolkit to a working sheet.
    - Connecting between the objects using connection arrows.
    - Writing code in the designated code slots.
    - Includes saving \ loading \ sharing a saved working sheet.
2. BP flow program execution:
    - After creating \ loading an existing program, the user can execute the BP flow program.
    - During execution, the events which take place will be displayed in a visual manner.
3. BP flow step by step program execution:
    - After creating \ loading an existing program, the user can see a visualization of the BP program execution in a step-by-step manner.
    - During the step-by-step execution, the current state of every flow run will be displayed visually.
4. External connection to BP flow program:
    - After creating a program, the user can use external events to trigger the BP flow program.

Chapter 2

Usage Scenarios

2.1 User Profiles — The Actors

The actors of the system are:

1. Programmers.
2. Developers.
3. Researchers.
4. External systems such as: robots, Tic-Tac-Toe program.

2.2 Use-cases

I. Creation of a new BP flow program
Main use-case flow:
   a. Create a new default worksheet.
   b. Add nodes for the flow diagram and connect them with arrows.
   c. Define the initial payloads in the starting nodes.
   d. Write code in the designated code slots in the general and bSync nodes.
   e. Decide on relevant events for the wait, request and block code slots in the bSync nodes.
   f. Edit the code on general nodes in order to determine output and scenario – splits.
   g. Save \ share the diagram.

II. Executing an existing BP flow program
Main use-case flow:
   a. Create a new BP flow program / load an existing one.
   b. Execute the BP flow program using the designated button.
   c. Invoked events are displayed in a visual manner to the user.

III. Debugging an existing BP flow program step-by-step
Main use-case flow:
   a. Create a new BP flow program / loading an existing one.
   b. Execute the program in debug mode.
   c. Display the state of each step, including the payload content in every.
   d. When the user uses the designated button for stepping forward/back – the display will change accordingly and describe the new state of the program.

IV. External connection to BP flow program
   a. Create a new BP flow program / loading an existing one.
   b. The BP flow program will be triggered by external program's start event.
   c. A two-way event-based connection will be established.

Chapter 3

Functional Requirements

1. BP Flow program creation:

No.	Description	Priority	Risk
1.1	The system should provide the option to create a new blank working sheet	MH	Low
1.2	The system should provide the option to create a new working sheet from the default examples.	NTH	Low
1.3.1	The system should enable saving a working sheet to the user's working space.	MH	Low
1.3.2	The system should enable saving a working sheet to the user's drive.	NTH	Low
1.4.1	The system should enable loading a saved working sheet from the user's working space.	MH	Low
1.4.2	The system should enable loading a saved working sheet from the user's drive.	NTH	Medium
1.5	The system should provide the option to add a free-text box to the working sheet.	NTH	Low
1.6	The system should provide a tool set of BP flow visual objects which corresponds to BP flow syntax (nodes)	MH	Low
1.6.1	The tool set should include the following objects: • General transformation node • Start node • B-Sync node • Console node Directed connection arrow.	MH	Low
1.6.2	The system should provide an option to define initial payloads in a start node	MH	Low
1.6.3.1	The system should provide a slot for a title for a General node on the working sheet.	MH	Low
1.6.3.2	The system should provide a slot for the number of output for a General node on the working sheet.	MH	High
1.6.4	The system should provide a slot for writing js code for a general node.	MH	Low

1.6.4.1	The system should enforce syntax check of the code written in every code slot.	NTH	High
1.6.5	The system should provide slots for defining requested, waited-for and blocked events on a Bsync node.	MH	Low
1.7	The system should enable to create the nodes from pop-up menu.	MH	Low
1.8	The system should enable to drag the objects from the tool set and drop them on the working sheet.	NTH	Medium
1.9	The system should enable moving objects from one place to another on the working sheet.	MH	Low
1.10	The system should enable deletion of any object from the working sheet.	MH	Low
1.11	The system should enable connect nodes by arrows.	MH	Low
1.11.1	The system should enforce connecting arrows between nodes only from the right side of a nodes to the left side of the other node.	MH	Low
1.12	The system should provide the option to add a free text box to the working sheet.	NTH	Medium

2. **BP Flow program visualization:**

2.1	The system should provide the option to insert a node between two existing nodes connected by an arrow.	NTH	High
2.2	The system should enable the option to connect an arrow to a node in every point on the node's frame.	NTH	High
2.3	The system should enable moving existing nodes and arrows during debug mode execution, as long as the changes do not affect the program's semantic.	NTH	Medium
2.4	The system should display "…" when a text in a node exceeds its boundaries.	NTH	Medium
2.4.1	The system should display sub windows for further information as pop up window.	NTH	Medium
2.5	The system should enable the option to display the execution of a program step-by-step (debug mode).	MH	Medium
2.5.1	The system should display the payload of each node during each step in a step-by-step execution.	MH	Medium
2.5.2	The system should mark the current running node.	MH	Medium
2.6	The system should provide the option to resize text and nodes located on the working sheet.	NTH	High

3. **BP Flow Program Execution:**

3.1	The system should enable execution of the program described visually on the working sheet according to BP Flow semantics.	MH	High
3.1.1	The system should create a new B-Thread for each scenario described on the working sheet (a start node, or a new split).	MH	Low
3.1.2	While in sync point, the system should choose the event selected randomly in uniform distribution.	MH	Low
3.1.3	The system should contain an output log/console during execution.	MH	Low

4. **Other:**

4.1	The system should provide an API to interact with the existing BP flow program.	NTH	High
4.1.1	The system should provide an option to be triggered by external events.	NTH	High
4.1.2	The system should be able to send events to external programs.	NTH	High

Chapter 4

Non-Functional Requirements

Implementation Constraints

1. Visualization:

1.1 Nodes on the working sheet can't appear on each other or hide any other object.

1.2 Arrows on the working sheet can't appear on each other or hide any other object.

1.3 Nodes' shape would be with rounded corners.

1.4 Arrows shape should be rounded.

1.5 Each node's type will be tagged by an image in the left side of the node.

1.6 Each node's type color will be different and unique.

1.7 The system should enforce the following arrow orientation:

- Input arrow should be on the left side of a node.
- Output arrow should be on the right side of a node.

2. Portability:

2.1 The web application should be supported in all the popular web browsers (Explorer, Chrome, FireFox, Safari).

2.2 The system should support only UTF-8 in every textual input slot.

3. Usability:

3.1 In order to use the system, a user should be familiar with BP and flow paradigms.

3.2 In order to use the system, a user should be familiar with JS coding language.

4. Other:

4.1 The system should execute the flowBP program on a bpjs server.

4.2 The system's design should enable extending the nodes stock by developers.

Platform Constraints

1. We are limited to bpjs capabilities.
2. We are limited to an existing Diagram system (Rete.js).

Chapter 5

Risk assessment & Plan for the proof of concept

Risks:

- Lack of experience with graphical edit tools and their capabilities.
- Working with a new programing paradigm, which is not well known in the software world.
- Determining the criteria of optimal UX and UI of the system.
- Change an existing project made by others, understanding their way of thinking.

Plan for the proof of concept:

i. Research
   a. Understanding better the mx-graph and NODE-RED functionality for the visualization of BP flow.
   b. Improve our knowledge about UI and UX.
   c. Better understanding of BP paradigm.
   d. Understanding deeply the existing project code.

ii. Preliminary
   a. Determine the division of responsibilities between client and server.
   b. Decide which diagramming tool is better to the project by the following criteria:
      i. Pleasurable design.
      ii. Intuitive and easy for users.
      iii. Low coupling between the graph's execution and visualization.

iii. Proof of concept:
   Implementing a basic version of the system including:
   1) Well-designed flow BP diagram.
   2) External programs connect to a BP flow program.
   3) Simple execution option.

Appendices

Behavioral Programming:

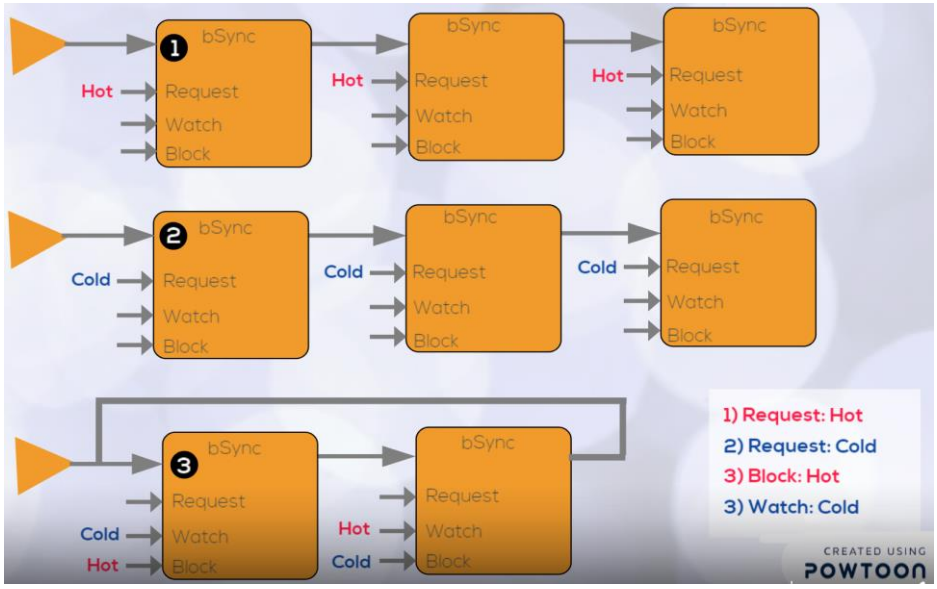
Behavioral programming is a novel, language-independent paradigm for programming reactive systems, centered on natural and incremental specification of behavior, and implemented in the visual formalism of live sequence charts (LSC), and in the BPJ Java package.

The approach allows coding applications as multi-modal scenarios, each corresponding to an individual requirement, specifying what can, must, or may not happen following certain sequences of events.

Each scenario is usually assigned with a single thread, also called as a b-thread (behavioral thread)

Flow BP example:

System that controls hot and cold water taps.



Let AddHotThreeTimes be a b-thread that requests three times the event of opening the hot water-tap (addHot), and then stops.

The b-thread AddColdThreeTimes performs a similar action on the cold water tap (with the event addCold).

We activate the above b-threads alongside a third one, Interleave, which forces the alternation of their events. Interleave repeatedly waits for addHot while blocking addCold, followed by waiting for addCold while blocking addHot.

Code example for Hot/Cold program:

```
bp.registerBThread("control-temp", function() {
        while ( true ) {
                bp.sync({waitFor:COLD, block:HOT});
                bp.sync({waitFor:HOT, block:COLD});
        }
});
```

```
bp.registerBThread("add-hot", function(){
        bp.sync({request:HOT});
        bp.sync({request:HOT});
        bp.sync({request:HOT});
});

bp.registerBThread("add-cold", function(){
        bp.sync({request:COLD});
        bp.sync({request:COLD});
        bp.sync({request:COLD});
});
```

The code consists of three b-threads: add-hot, which adds the hot water, add-cold, which adds the cold water and the control-temp that maintains the order between the hot and cold water, when we add cold water we block the hot water flow and after we finish to add the cold water the hot water can flow and the cold water are blocked until we finish the hot water request, the scenario continues in the same way.

Glossary:

1. Payload – a data dictionary for a specific scenario, holding all the variables that are relevant for that scenario.
2. Scenario - a sequence of actions or events that describe a certain behavior. Usually each scenario will be assigned with a different thread.
3. Node – a graphical object with placeholders for JS code, input and outputs.
4. General node- a node with one input entry, several output exits, a JS code slot and payloads content.
5. Start node – a node with no input entries, one output exit and a slot to insert initial payloads.
6. B-Sync node –a node with one input entry, one output, payloads content, as well as requesting, waiting and blocking (events) slots.
7. Console node – a node with one input entry, one output, payload content and a JS code slot, that the value which is returned from the JS code is printed to the output console.
8. Entries – a port for input of payloads that transfer from one node to another.
9. Output – a port of output of payload that transfer from one node to another (after changing- or not- in the node it goes out from).
10. Arrow – a connection between node output to a node entry (represent the possible path of a payload that moves from one node to another).
11. Requesting an event - proposing that the event be considered for Triggering and asking to be notified when it is triggered.
12. Waiting for an event - without proposing its triggering, asking to be notified when the event is triggered.
13. Blocking an event - forbidding the triggering of the event, vetoing requests of other behavior threads.
14. Requesting slot – code slot designate for Requesting event.
15. Waiting slot – code slot designated for a Waiting event.
16. Blocking slot – code slot designated for Blocking event.
17. Code slot – code frame designated for JS code.
18. Rete.js – a JavaScript based Vue diagramming library that provides a browser-based editor for wiring together flows.
19. mx-graph – a JavaScript diagramming library that enables interactive graph and charting applications to be quickly created.
20. NODE-RED- a programming tool provides a browser-based editor for wiring together flows using the wide range of nodes.
21. Flow BP – application's behavior as a network of "nodes". Each node has a well-defined purpose, it is given some payload, it does something with that payload and then it passes that payload on to the next nodes (which their inputs are connected to its outputs by arrows).