Flow BP

Software Engineering Project
Application Requirements Document

Eilon Benami
Eyal Almog
Hadas Atiya
Gilad Abudi

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

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.
- Our objectives are:
  1. Examination of graphical tools from the software world relevant for visualization of BP flow
  2. Applying adjustments and modification for the tools found, while examining alternatives
  3. Development of an interpreter for BP flow programs
  4. Connecting the client side web application to a BPjs server

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-side browser-based web application, including an interpreter for executing BP flow programs.

The main user scenarios are:

1. BP flow program creation:
   - Dragging 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 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

Chapter 2

Usage Scenarios

2.1 User Profiles — The Actors

The actors of the system are programmers , developers and researchers.

2.2 Use-cases

I. Creation of a new BP flow program
   Main use-case flow:
   a. Creating a new blank worksheet
   b. Adding blocks for the flow runs and connecting them with the connection arrows
   c. Defining the initial payloads in the starting blocks
   d. Writing code in the designated code slots in the general and bSync blocks
   e. Deciding on relevant events for the waiting, requested and blocked code slots in the bSync blocks
   f. Editing the code on general blocks in order to determine outpus and scenario - splits

II. Executing an existing BP flow program
   Main use-case flow:
   a. Creating a new BP flow program / loading an existing one
   b. Execution of the BP flow program using the designated button / command
   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. Creating a new BP flow program / loading an existing one
   b. Execute the program, record each step of the execution
   c. The state of each scenario run at every step is displayed to the user visually (payload content in every block, current events status – waiting, requested and blocked events)
   d. When the user uses the designated button/ command for stepping forward/back – the display will change accordingly and describe the new state of the program

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
1.1.1	The system should enable saving a working sheet to the user's disk	MH	Low
1.1.2	The system should enable loading a saved working sheet	MH	Low
1.2	The system should provide a tool set of BP flow visual objects which corresponds to BP flow syntax (blocks)	MH	Low
1.2.1	The tool set should include the following objects: • General transformation block • Start block • B-Sync block • Console Block Directed connection arrow.	MH	Low
1.2.1.1	The system should provide an option to define initial payloads in a start node	MH	Medium
1.2.1.2	The system should provide a slot for a title for a General block on the working sheet	MH	Low
1.2.1.3	The system should provide a slot for writing js code for a general block	MH	Low
1.2.1.3.1	The system should enforce syntax check of the code written in every code slot	NTH	Medium
1.2.1.3.2	The system should apply syntax highlighting in every code slot	NTH	Medium
1.2.1.4	The system should provide slots for defining requested, waited for and blocked events on a Bsync node	MH	Medium
1.3	The system should enable to drag the objects from the tool set and drop them on the working sheet	MH	Medium

1.4	The system should enable moving objects from one place to another on the working sheet	MH	Low
1.5	The system should enable deletion of any object from the working sheet	MH	Low
1.6	The system should enforce connecting arrows between blocks only from the right side of a block to the left side of the other block	MH	Low
1.7	The system should provide the option to add a free-text box to the working sheet	NTH	Low
1.8	The system should provide the user an option to create and customize new kinds of blocks	NTH	High
1.9	The system should enforce a visual "sanity" check for the flow diagram (arrows not connected to anything, bad locations, etc)	MH	Low

2. **BP Flow program visualization:**

2.1	The system should provide the option to resize text and blocks located on the working sheet.	NTH	High
2.2	The system should provide the option to change the blocks' default shapes	MH	High
2.3	The system should provide the option to convert the current BP flow diagram to JS code	NTH	Low
2.4	The system should enable the option to display the execution of a program step-by-step(debug mode).	MH	High
2.4.1	The system should enable to display every payload content while in debug mode	MH	High
2.4.2	The system should enable the option to display each event's status in every B-Sync block (waiting, blocked or ready to run) in debug mode	MH	High
2.4.3	The system should provide the option to change the intervals of the step-by-step execution	NTH	High

3. <u>**BP Flow Program Execution:**</u>

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
3.1.3.1	The system should display the events taking place during the execution on the output log/console	MH	Low

4. <u>**Other:**</u>

4.1	The system should contain an event input console during execution	NTH	Medium
4.1.1	The system should provide the option to display all the possible events to enter on execution	NTH	Low
4.1.2	The system should enable the user to insert input – user-invoked events from the event list presented	NTH	Medium

Chapter 4

Non-Functional Requirements

1. Visualization:

1.1 Blocks 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 Blocks' size should fit to the text boxes inside of it.

1.4 The system should enforce the following arrow orientation:

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

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 design should enable replacing the client side interpreter (moving from in-place interpreting to server interpreting).

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 (small number of examples and existing tools).
- Determining the criteria of optimal UX and UI of the system.

Plan for the proof of concept:

i. Research
   a. Examination of graphical tools from the software world relevant for visualization of BP flow (API's, popular UI's etc.).
   b. Better understanding of BP paradigm.

ii. Preliminary
   a. After choosing a graphical tool, implementing the basic functionality of adding object and arrows to a working sheet.
   b. Implementing a conversion of the displayed diagram to a data structure.

iii. Proof of concept:
   Implementing a basic version of the system including:
   1) Toolkit- including start and bSync blocks and connection arrows.
   2) Working sheet- clean white area which the object will appear on.
   3) Simple execution option.

Appendices

Behavioral Programming:

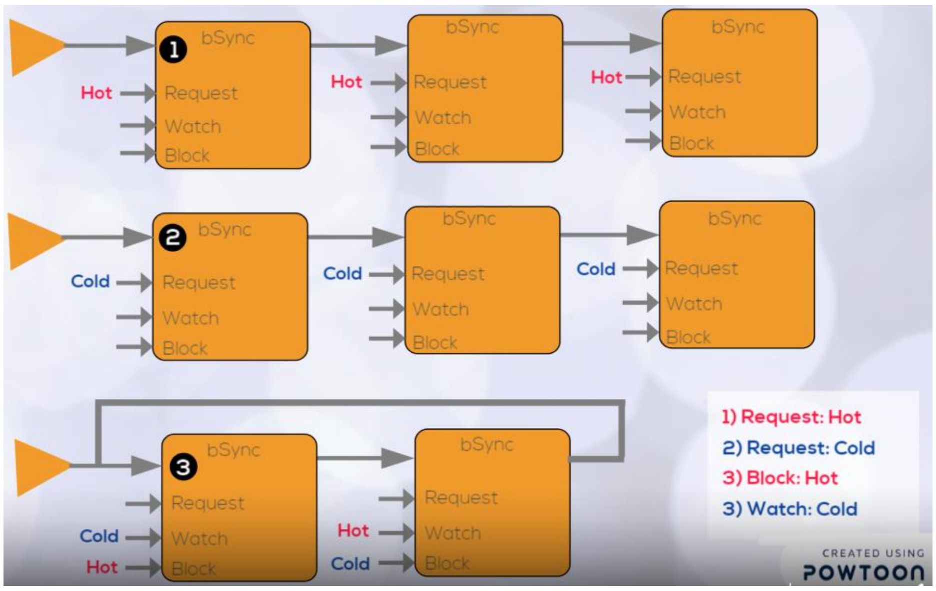
Behavioral programming is a novel, language-independent paradigm for programming reactive systems, centered on natural and incremental specification of behavior, and implemented in the visual formalism of live sequence charts (LSC), and in the BPJ Java package.

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. Block – a graphical object with placeholders for JS code, input and outputs
4. General transformation block- a Block with one input entry, several output exits, a JS code slot and payloads content.
5. Start block – a Block with no input entries, one output exit and a slot to insert initial payloads.
6. B-Sync block –a Block with one input entry, one output, payloads content, as well as requesting, waiting and blocking (events) slots.
7. Console Block – a Block 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 block to another.
9. Output – a port of output of payload that transfer from one block to another (after changing- or not- in the block it goes out from).
10. Arrow – a connection between block output to a block entry (represent the possible path of a payload that moves from one block 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. Toolkit – a toolbar that contains all existing blocks and the arrow.
19. Flow BP – application's behavior as a network of "Blocks". Each Block 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 blocks (which their inputs are connected to it's outputs by arrows).