Executable graph models in Bloc

Louis Müller

Software Composition Seminar, fall 2019

Broader vision

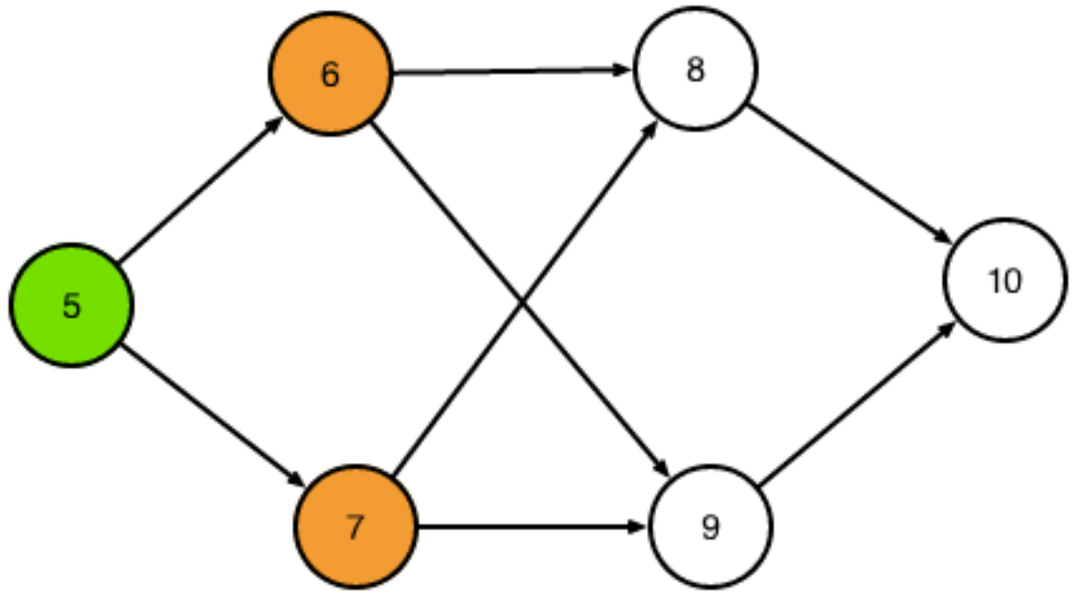
Many software models are based on graphs.

➡ Exploring possibilities in GT to interactively construct graphs.

Project goals

Experiment with ways to quickly build graph models using Bloc:

- interactively construct graphs (syntax)
- assign interpretation to graphs (semantics)
- execute graph models



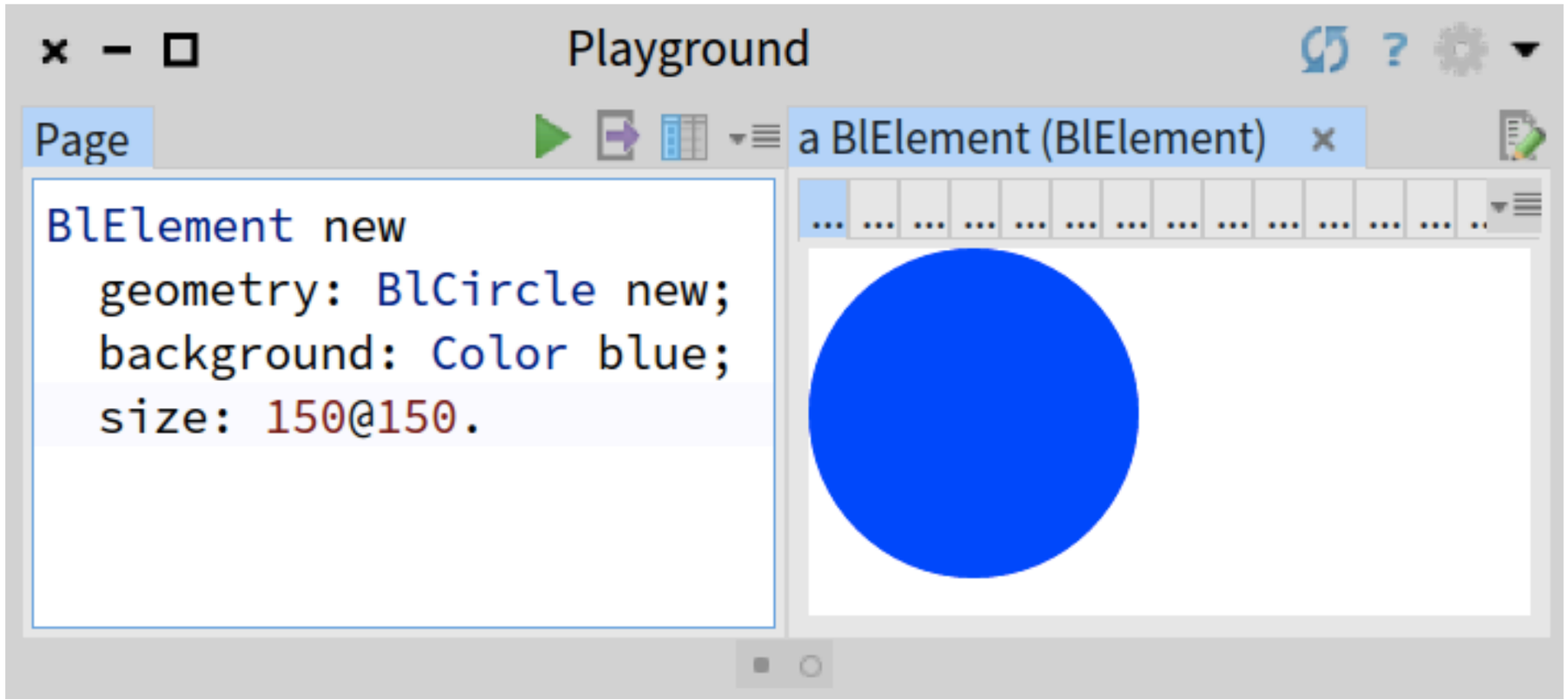
What is Bloc?

Glamorous Toolkit (GT) is a moldable development environment written in Pharo Smalltalk.

Bloc is the graphical framework delivered with **GT**.

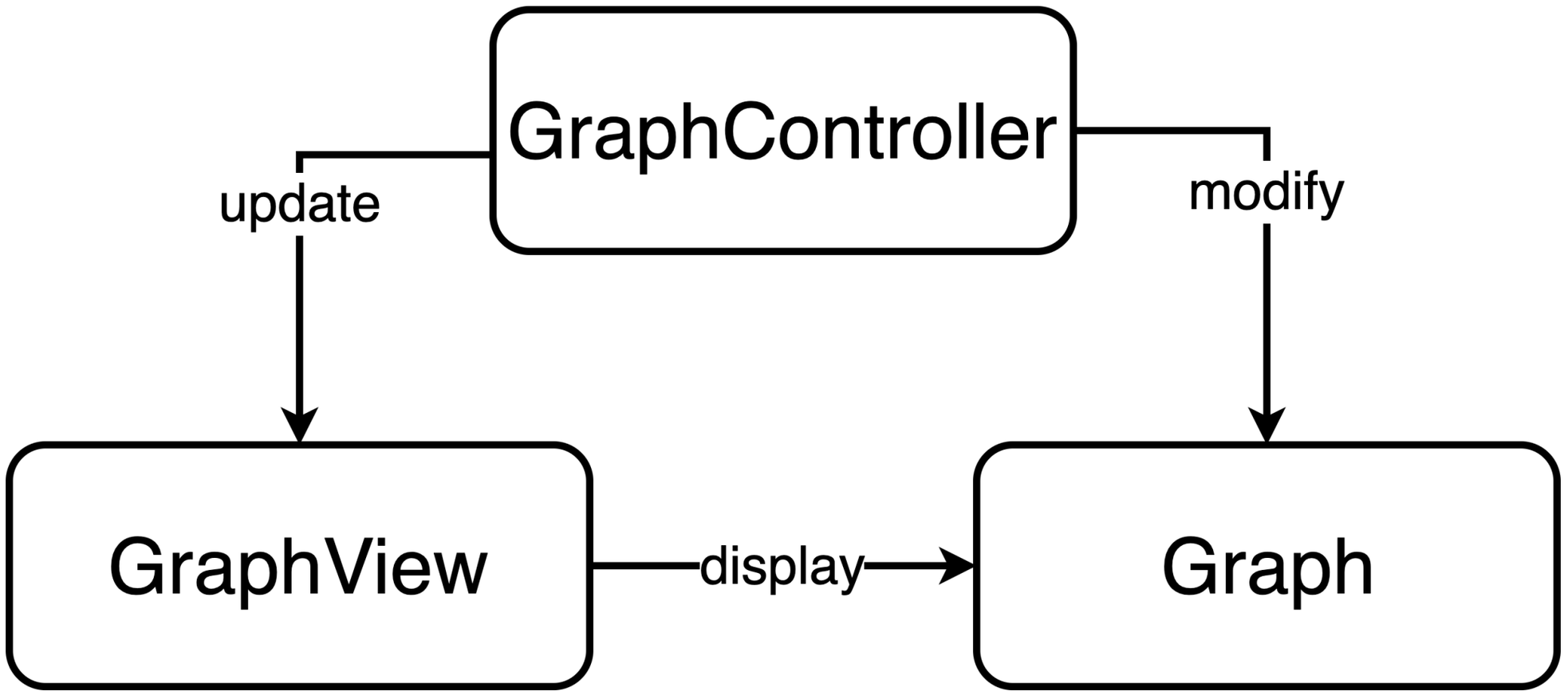
Bloc hierarchy

BlElement is the root class of all visual elements in Bloc



Implementation background

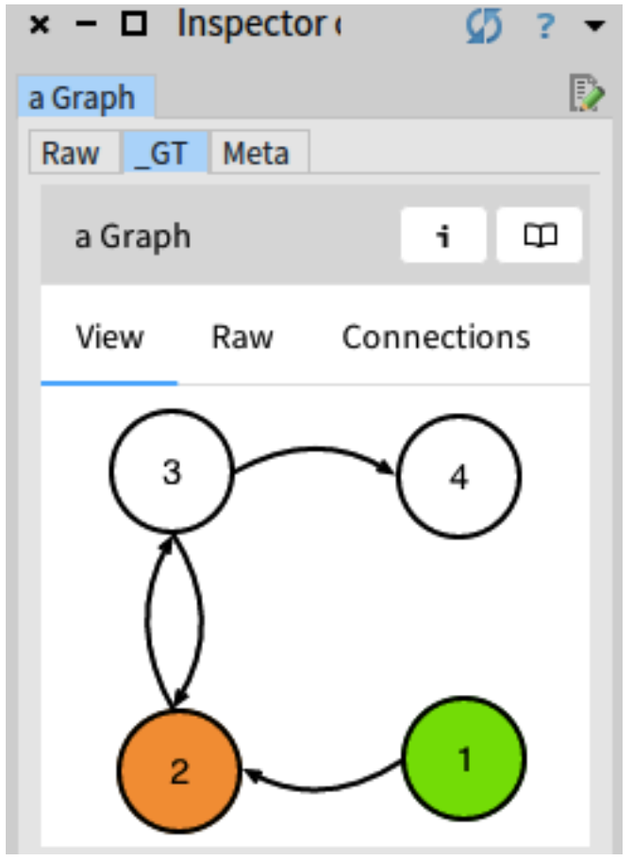
Applied model-view-controller (MVC) design pattern



Implementation results

1. Interactive graph editor

2. Export created graph as code

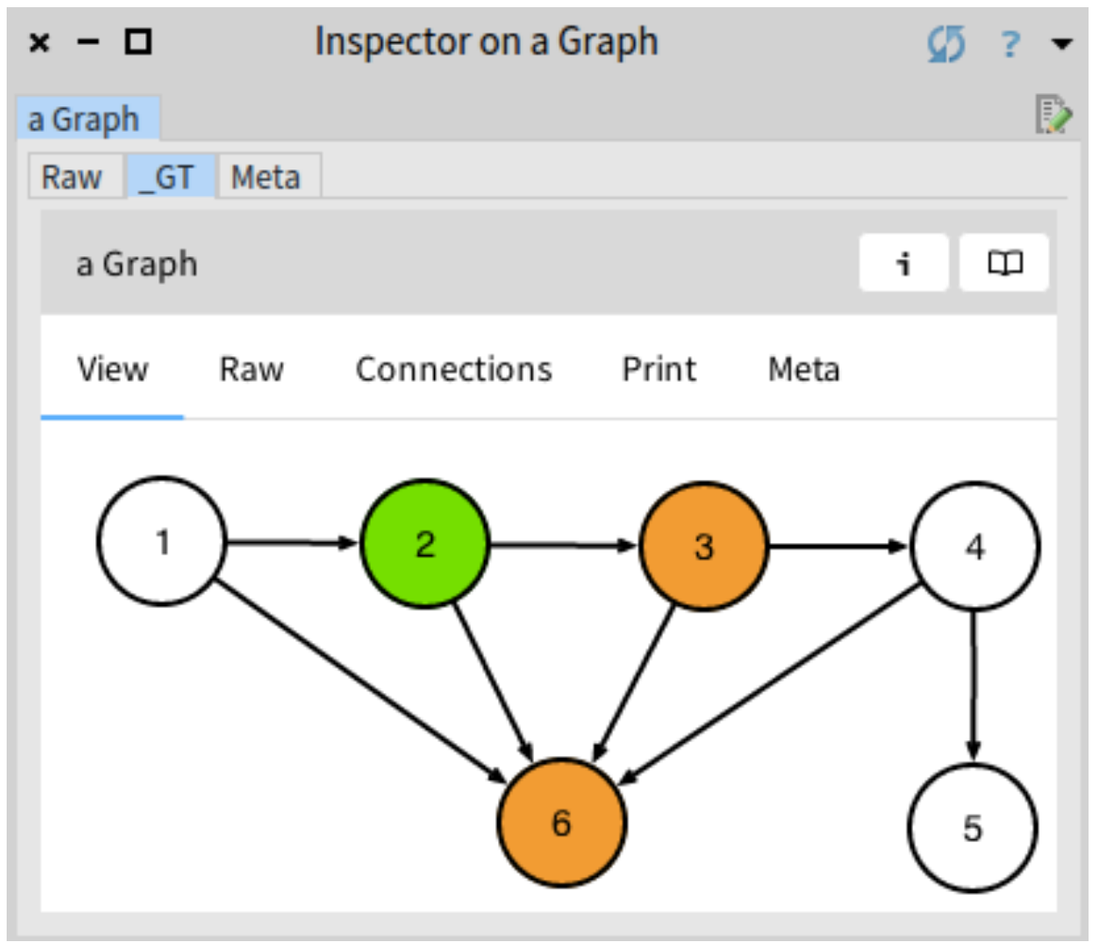


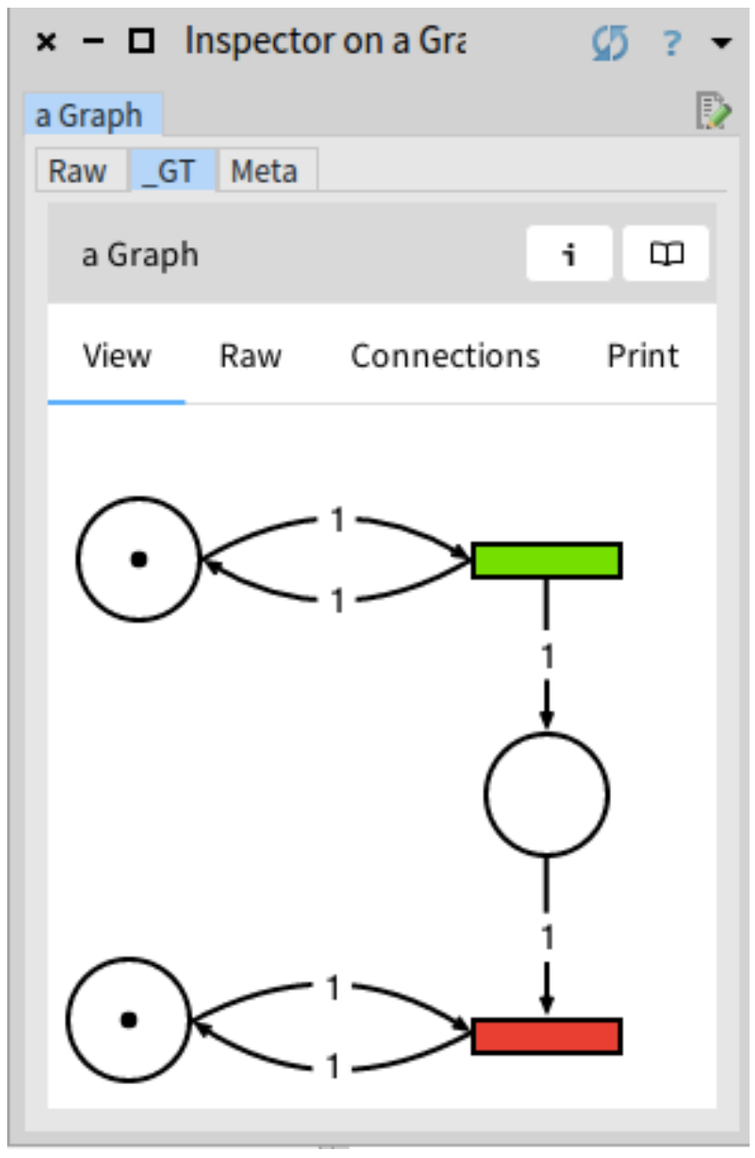

```
demo1
<gtExample>
self nodes: #(1 2 3 4) asOrderedCollection.
self places: #() asOrderedCollection.
self transitions: #() asOrderedCollection.
self edges: {1->2. 2->3. 3->2. 3->4} asOrderedCollection.
self currentState: 1.
self weights: {(1->2)->0.(2->3)->0.(3->2)->0.
(3->4)->0} asOrderedDictionary.
self tokens: {} asOrderedDictionary.
self nodeLocations: { 1->(143.0@123.0). 2->(30.0@128.0).
3->(27.0@9.0). 4->(141.0@11.0)} asOrderedDictionary.
```

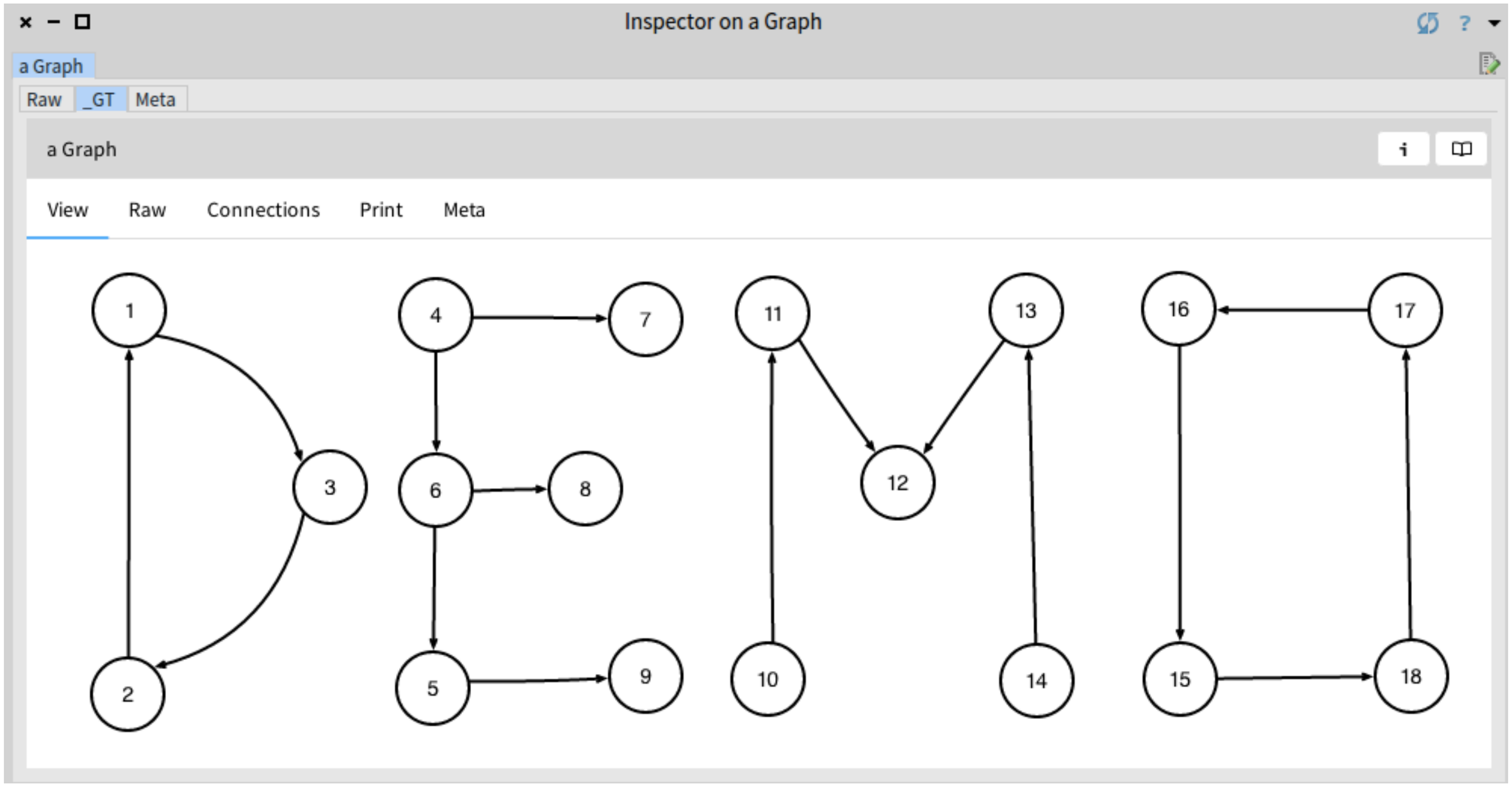
Implementation results

3. Implementation of finite state automata (FSA)

4. Implementation of petri net






Inspector on a Graph
a Graph
Raw
_GT
Meta
a Graph
View
Raw
Connections
Print
Meta
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18

Challenges

- Finding a stable GT version

- Limited sources of knowledge about Bloc

Possible future work

- Implement further graph models, i.e. extend syntax & semantics.

- Add algorithms for solving graph problems (e.g. max flow problem).

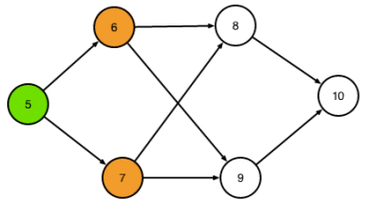
- Integration into the interactive actor modeling project.

Summary

Project goals

Experiment with ways to quickly build graph models using Bloc:

- interactively construct graphs
- assign interpretation to graphs (semantics)
- execute graph models



3

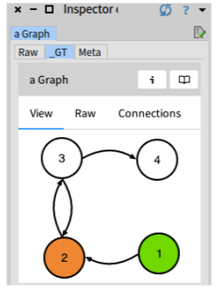
Possible future work

- Implement further graph models, i.e. extend syntax & semantics.
- Add algorithms for solving graph problems (e.g. max flow problem).
- Integration into the interactive requirements modeling project.

11

Implementation results

1. Interactive graph editor (syntax)
2. Export created graph as code

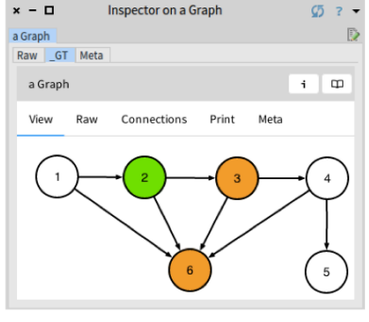


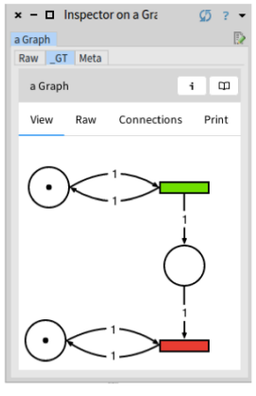

```
demo1
<gtExample>
self nodes: #(1 2 3 4) asOrderedCollection.
self places: #() asOrderedCollection.
self transitions: #() asOrderedCollection.
self edges: {1->2. 2->3. 3->2. 3->4} asOrderedCollection.
self currentState: 1.
self weights: {(1->2)->0.(2->3)->0.(3->2)->0.
(3->4)->0} asOrderedDictionary.
self tokens: {} asOrderedDictionary.
self nodeLocations: { 1->(143.0@123.0). 2->(30.0@128.0).
3->(27.0@9.0). 4->(141.0@11.0)} asOrderedDictionary.
```

7

Implementation results

3. Implementation of finite state automata (FSA)
4. Implementation of petri net





9