VDMPad: a Lightweight IDE for Exploratory VDM-SL Specification

Tomohiro Oda — Software Research Associates, Inc.
Keijiro Araki — Kyushu University
Peter G. Larsen — Aarhus University

This work is supported by Grant-in-Aid for Scientific Research (S) 24220001

Agenda

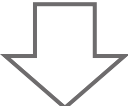
1. Exploratory specification
2. VDMPad
3. LIVE tastes of VDMPad
4. Lightweight IDE for lightweight modeling
5. Conclusion

Exploratory Specification

exploratory specification

pre-formal phase

informal requirements

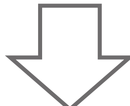


formal specification
which FM tools support

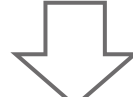
exploratory specification

the first step into formal spec

informal requirements



struggle to produce
an initial draft of formal spec



formal specification
which FM tools support

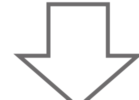
exploratory specification

Cycle of exploration

informal requirements

write a specification

by understanding the domain



understand a domain

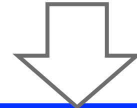
by writing the specification

rigorous formal specification

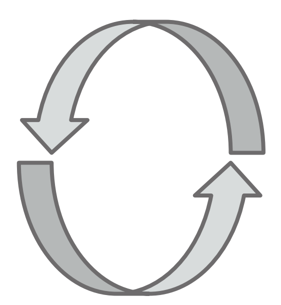
which FM tools effectively support

exploratory specification

informal requirements



exploratory formal specification

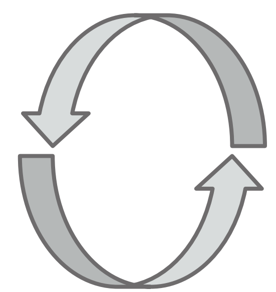


write a specification
by understanding the domain

understand a domain
by writing the specification

formal specification
which FM tools effectively support

exploratory specification
Challenges



write a specification
by understanding the domain

understand a domain
by writing the specification

Repeat trial and error

various abstraction of the domain

various constructs of the language

The problem definition is not clear.

Because we ARE defining it.

We learn the nature of the problem
from the spec you will write.

VDMPad

VDMPad

A lightweight VDM-SL IDE for

- exploratory formal specification
- introductory education of VDM-SL

with LIVE tastes

Quick overview of VDM-SL

- types
  - nat, real, char, seq, set, map, composite, token, ...
- values
  - constant values
- functions
  - pure (total / partial) functions
  - expressions (if-then-else, lambda, ...)
- states
  - variables
- operations
  - statements (assignments, while, ...)

VDM-SL

example: fibonacci numbers

```
1  state Fib of
2     n1 : nat
3     n2 : nat
4     init s == s = mk_Fib(0, 1)
5     inv mk_Fib(n1, n2) == n1 > 0 or n2 > 0
6  end
7  operations
8     next : () ==> nat
9     next() == (dcl n : nat := n1 + n2; n1 := n2; n2 := n; return n)
10    post RESULT = n1~ + n2~ and n2 = n1~ + n2~ and n1 = n2~;
11
12    prev: () ==> nat
13    prev() == (dcl n : nat := n2 - n1; n2 := n1; n1 := n; return n2)
14    pre n1 > 0
15    post RESULT = n2 and n1 + n2 = n2~ and n2 = n1~;
```

LIVE tastes

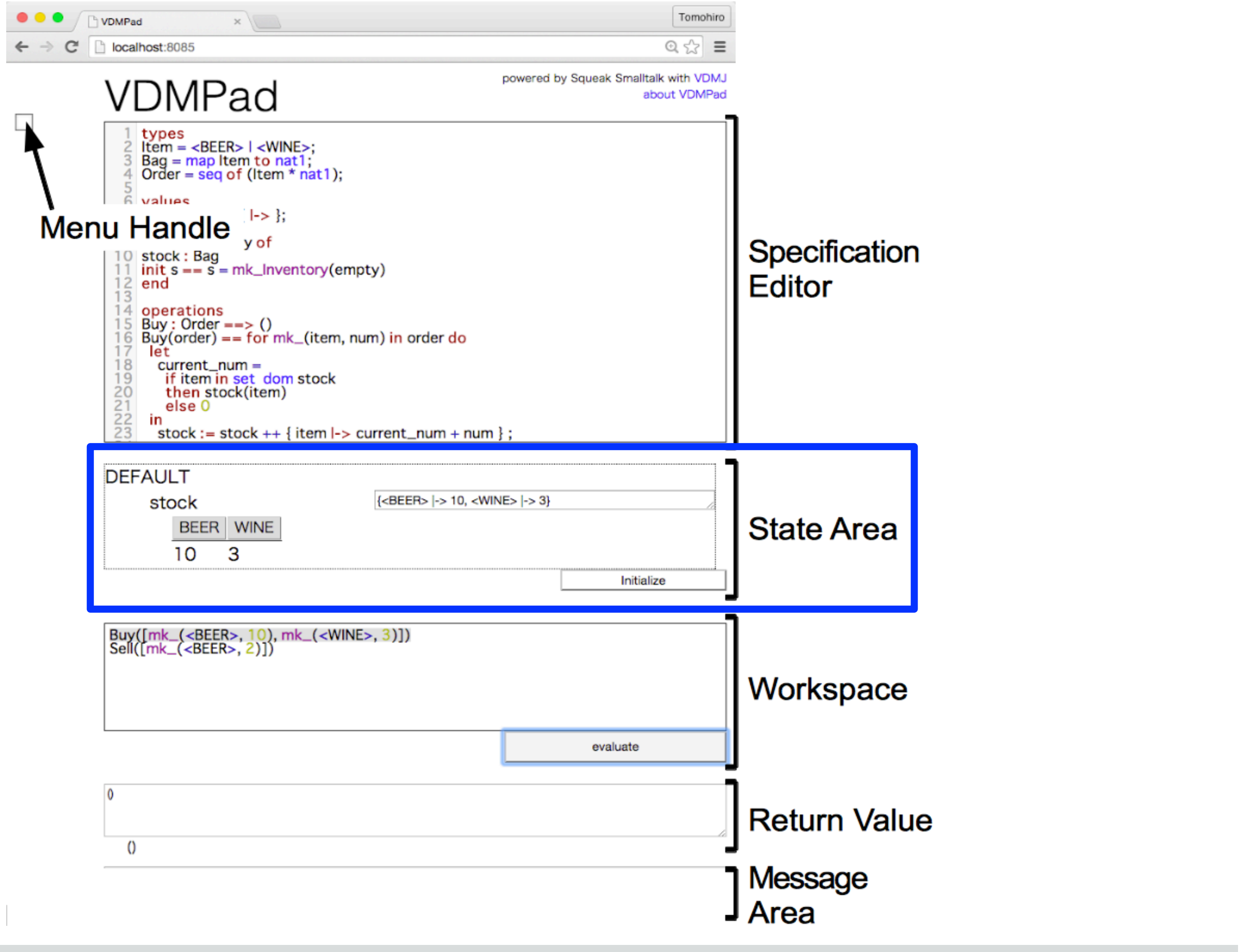
LIVE tastes of VDMPad

- state manipulation
- workspace
- animation over modifications
- visual presentation
- continuous unit testing
- permissive checking

LIVE tastes
state manipulation

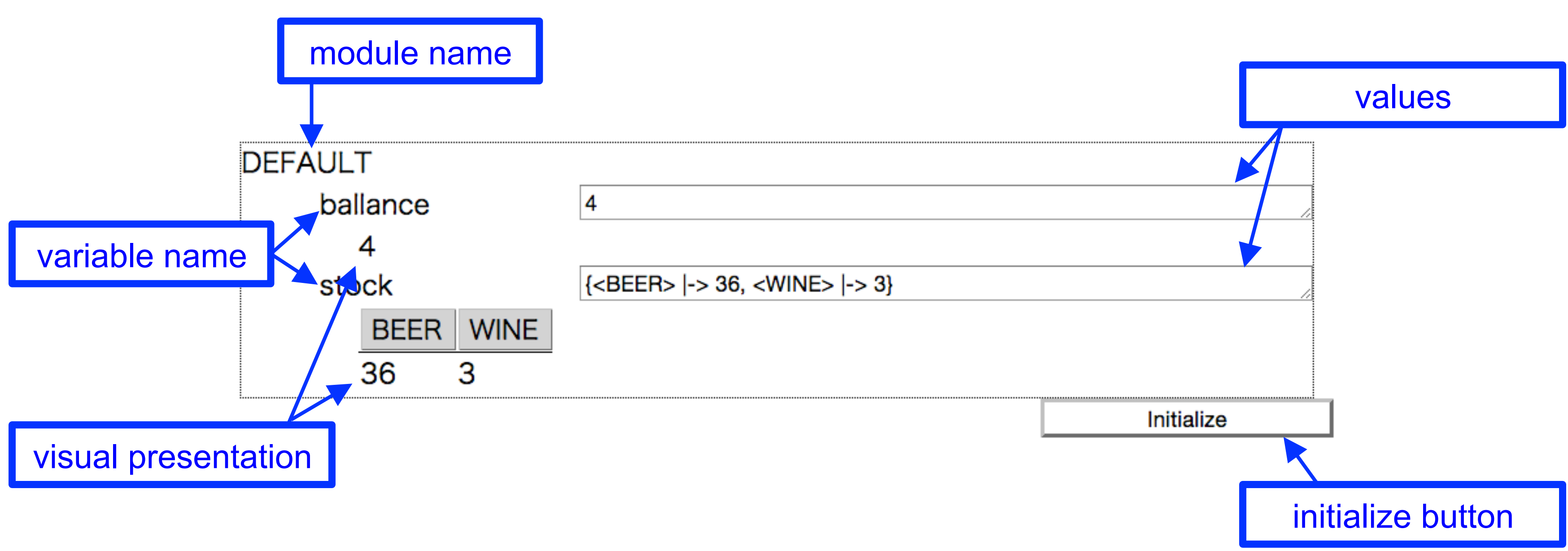
- The user can directly edit the state of the animated system.
    - to check if the given state satisfies invariants
    - to animate behavior of the system in the given hypothetical state
        - not always be realized by a series of operations
        - easy to reproduce the state of the concern.

LIVE tastes
state manipulation



LIVE tastes

state manipulation

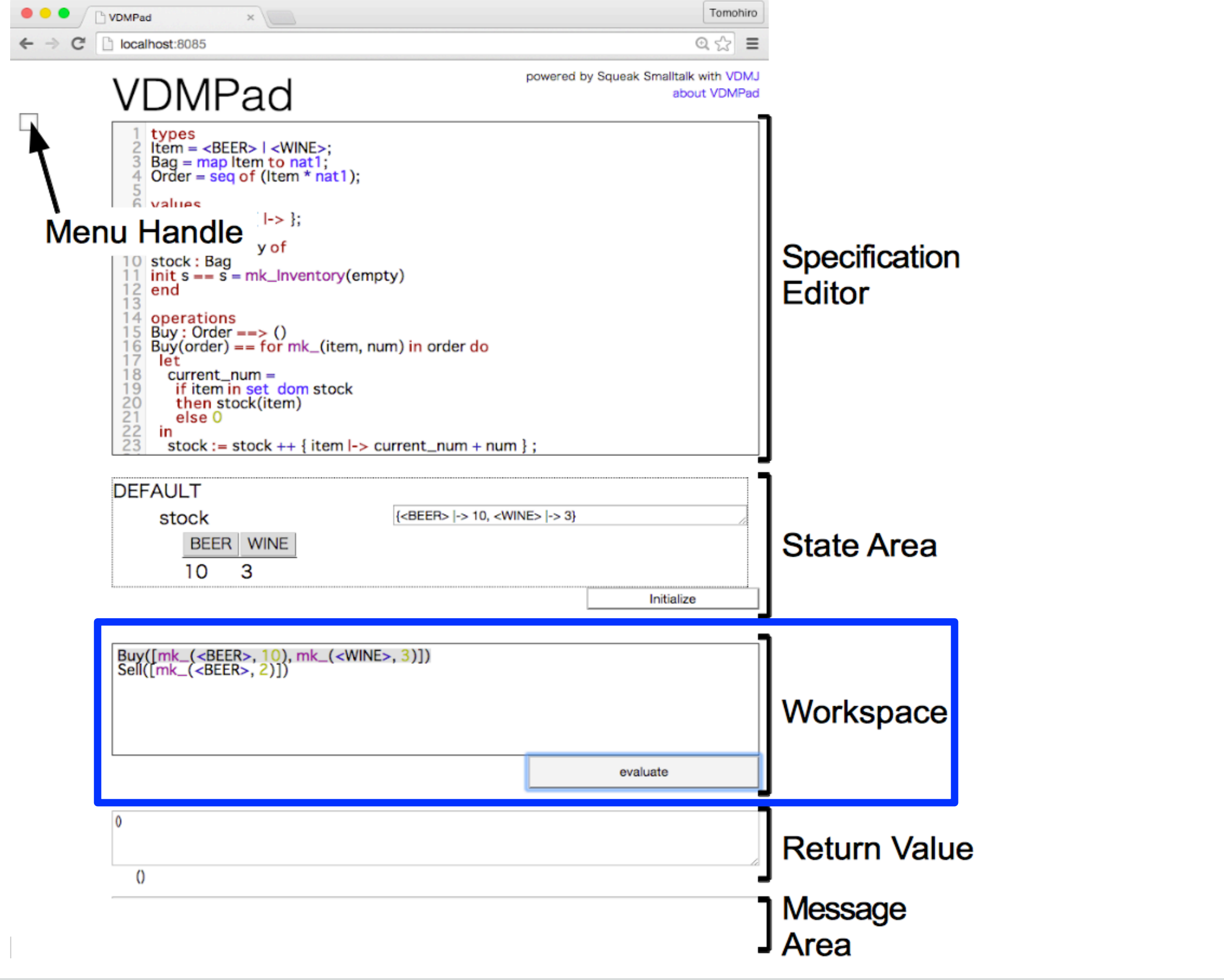


LIVE tastes

workspace

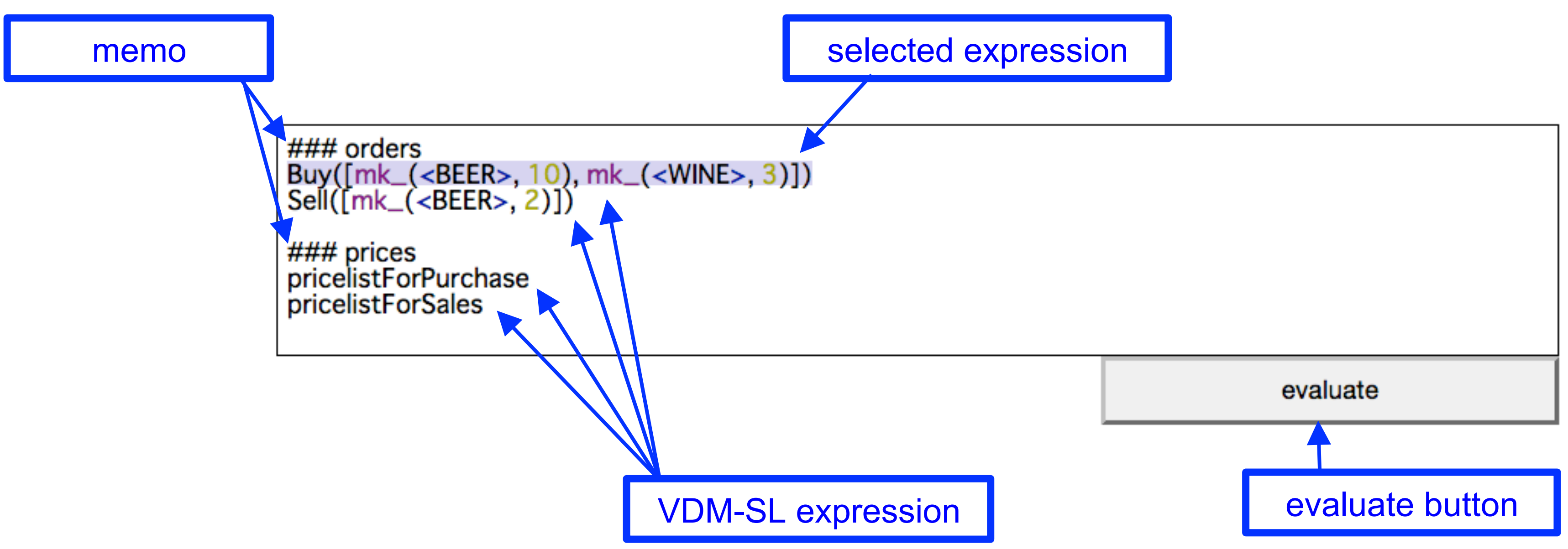
- workspace is a free text editor
    - to list and evaluate
        - a series of operations in a scenario.
        - a set of basic operations to drive the animated system in exploratory ways.
    - to leave memos in natural languages.

LIVE tastes
workspace



LIVE tastes

workspace



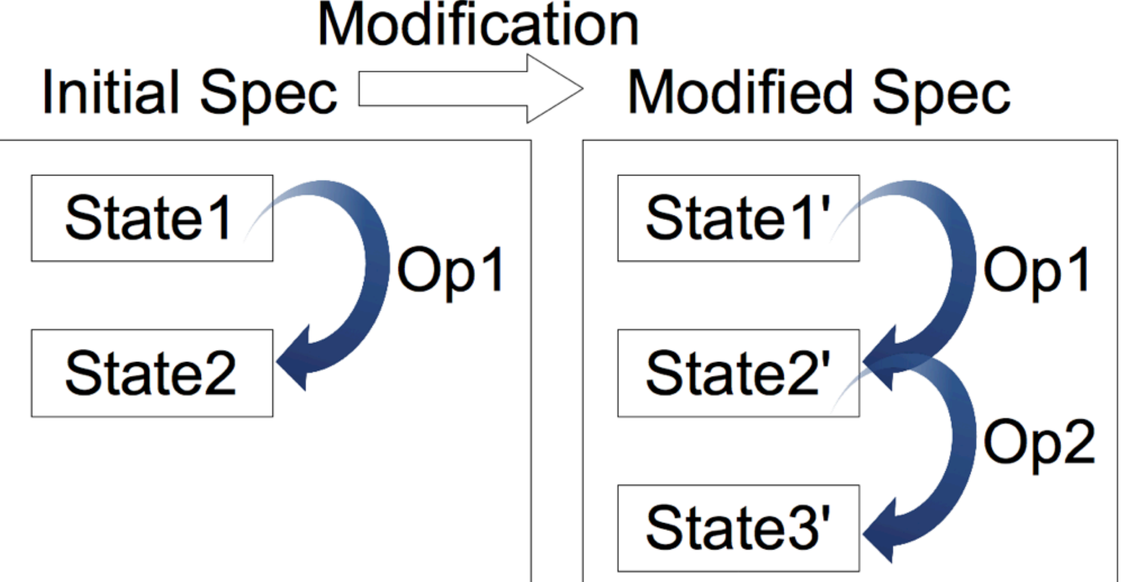
More freedom than
REPL (Read-Eval-Print Loop) console!

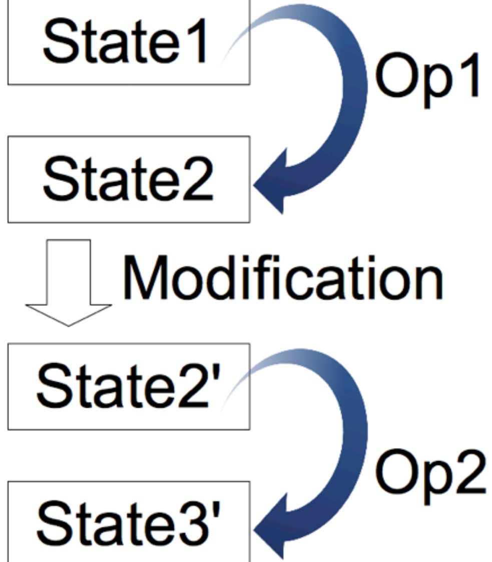
LIVE tastes
animation over modifications

- Keep the state of the animated system when modifying the spec.
    - to continue the on-going scenerio after fixing a minor bug.
    - for immersive modeling.

LIVE tastes

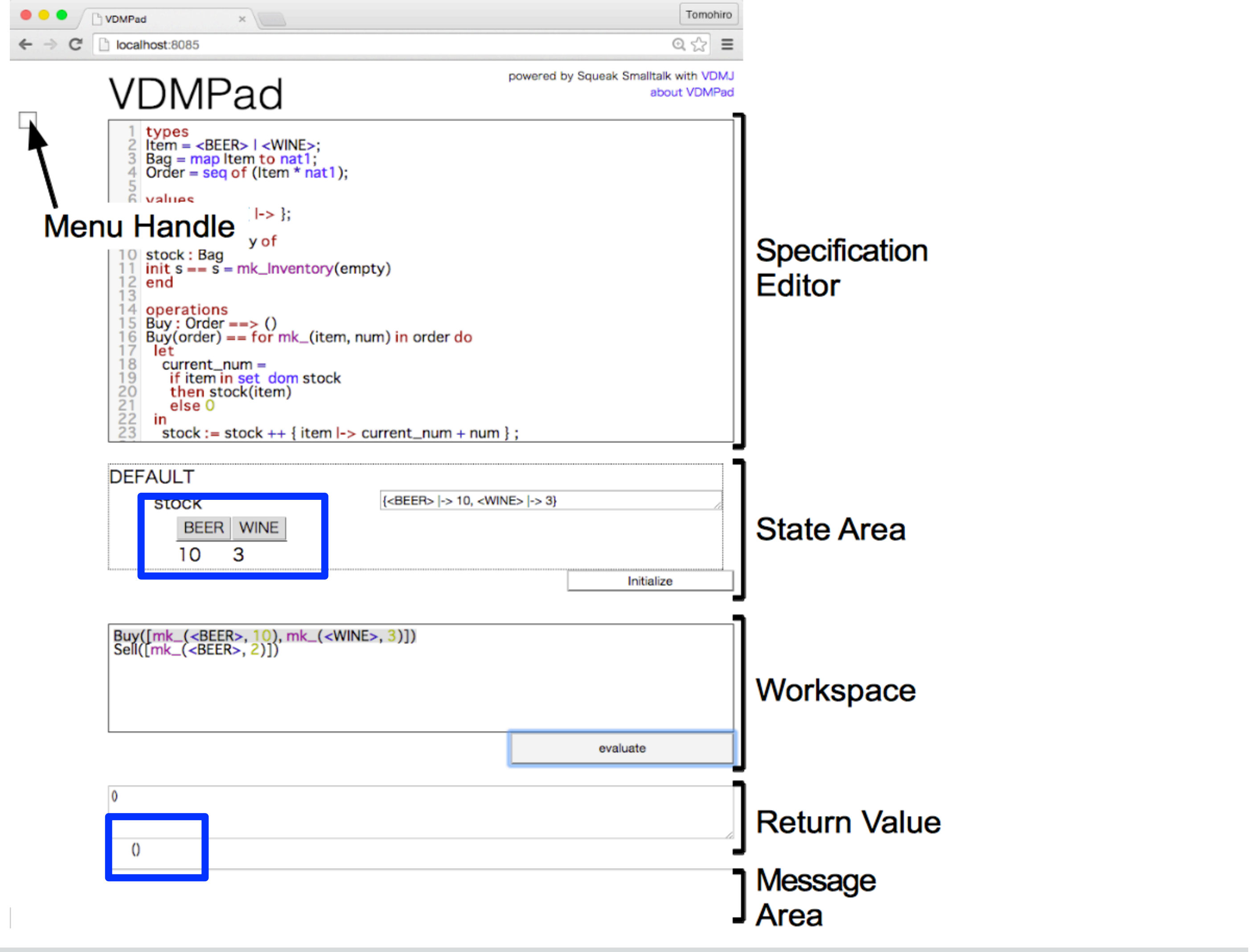
animation over modifications





LIVE tastes

visual presentation



LIVE tastes

visual presentation

<table>
<tr><th>type</th><th>value</th><th>diagram</th></tr>
<tr><td>real</td><td>1.0</td><td>1</td></tr>
<tr><td>symbol</td><td><symbols></td><td>symbols</td></tr>
<tr><td>seq of char</td><td>"abc"</td><td>"abc"</td></tr>
<tr><td>seq</td><td>[1, 2, 3, 4]</td><td>1 2 3 4</td></tr>
<tr><td>set</td><td>{1, 2, 3, 4}</td><td>1 2
3 4</td></tr>
<tr><td>map</td><td>{<one> |-> 1, <two> |-> 2 }</td><td>one two
1 2</td></tr>
<tr><td>product</td><td>mk_tuple(1, "abc")</td><td>1
"abc"</td></tr>
<tr><td>composite</td><td>mk_Record(1, "abc")</td><td>Record
1
"abc"</td></tr>
<tr><td>token</td><td>mk_token(0)</td><td>token
0</td></tr>
</table>

LIVE tastes continuous unit testing

- always run unit tests after evaluation
  - as a discipline in trial and error process
  - to detect degrading by trial and error

LIVE tastes

continuous unit testing

```
Buy([mk_(<BEER>, 10), mk_(<WINE>, 3)])
Sell([mk_(<BEER>, 2)])
```

evaluate

make it a testcase

()

“ make it a testcase ” Button

()

OK: Buy([mk_(<BEER>, 10), mk_(<WINE>, 3)])

OK: Sell([mk_(<BEER>, 2)])

Results of Unit Tests

LIVE tastes
continuous unit testing

OK: Buy([mk_(<BEER>, 10), mk_(<WINE>, 3)])

prestates : {"DEFAULT`stock":"{|->}"}
expression : Buy([mk_(<BEER>, 10), mk_(<WINE>, 3)])
value : ()
poststates : {"DEFAULT`stock":"{<BEER> |-> 10, <WINE> |-> 3}"}

delete

OK: Sell([mk_(<BEER>, 2)])

prestates : {"DEFAULT`stock":"{<BEER> |-> 10, <WINE> |-> 3}"}
expression : Sell([mk_(<BEER>, 2)])
value : ()
poststates : {"DEFAULT`stock":"{<BEER> |-> 8, <WINE> |-> 3}"}

delete

LIVE tastes
permissive checking

- can optionally disable runtime checking
  - to simulate "bad" scenario
  - to focus on more important issue

not for regular use!

Lightweight

Lightweight IDE

VDMPad is lightweight in the senses of

- no installation, less footprints, quick launch
- less setup to start with a new model
- simple user interfaces
- small and focused functionality

Lightweight IDE
no installation, less footprints, quick launch

- Web-based IDE
  - a free server available online.
  - open http://vdmpad.csce.kyushu-u.ac.jp/
    and then you have the IDE before your eyes.

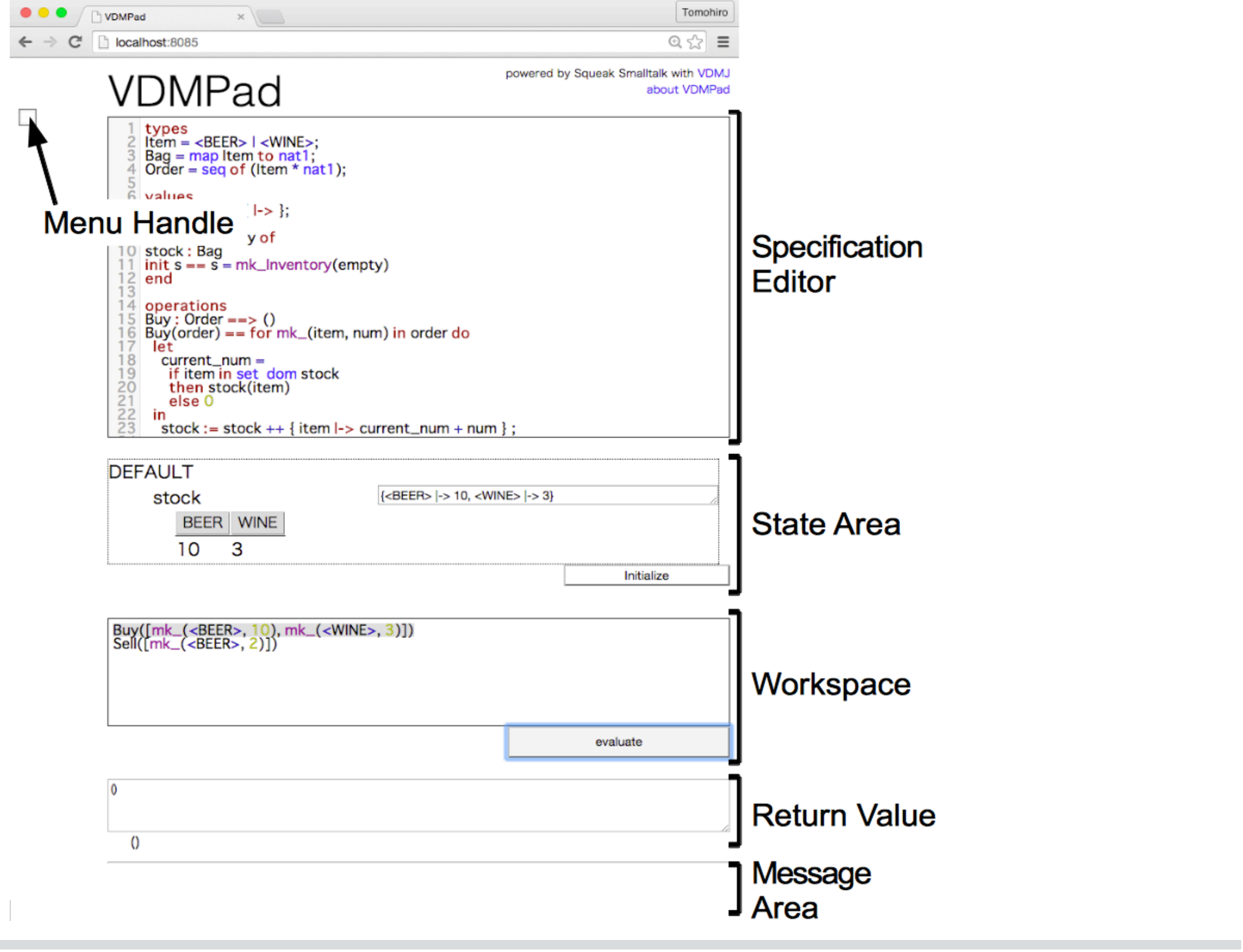
- runs on Firefox browser and Google Chrome

Lightweight IDE
less setup to start with a new model

- no need for user registration
    - Nothing is stored on the server.
- no need for source trees
    - Everything is stored in your browser.
- spec and animation contexts are automatically saved into your browser

All you need to write a spec is on the browser's localStorage (HTML5's standard key-value DB)

Lightweight IDE
simple user interface



Lightweight IDE
small and focused functionality

- The "evaluate" button is the only operation to invoke functionality.
  - edit a specification
  - change the state
  - eval an expression

- menu to manage stored animations and options
  - animations: load, save, delete, export
  - options: 5 checkboxes

Conclusion

Conclusion

- exploratory specification
  - trial and error
    to obtain the first grip on the right abstraction
- LIVE tastes
  - more freedom to try
  - immersive modeling
  - discipline by continuous unit testing
  - occasionally permissive
- lightweight IDE
  - good for introductory education
  - always ready to go

Thank you.