Specifying Abstract User Interface in VDM-SL

Tomohiro Oda	**Software Research Associates, Inc.**
Keijiro Araki	National Institute of Technology, Kumamoto College
Yasuhiro Yamamoto	Future University Hakodate
Kumiyo Nakakoji	Future University Hakodate
Han-Myung Chang	Nanzan University
Peter Gorm Larsen	Aarhus University

Agenda

1. ViennaTalk
2. User Interface
3. ViennaVisuals
4. live demo
5. Summary

ViennaTalk

ViennaTalk

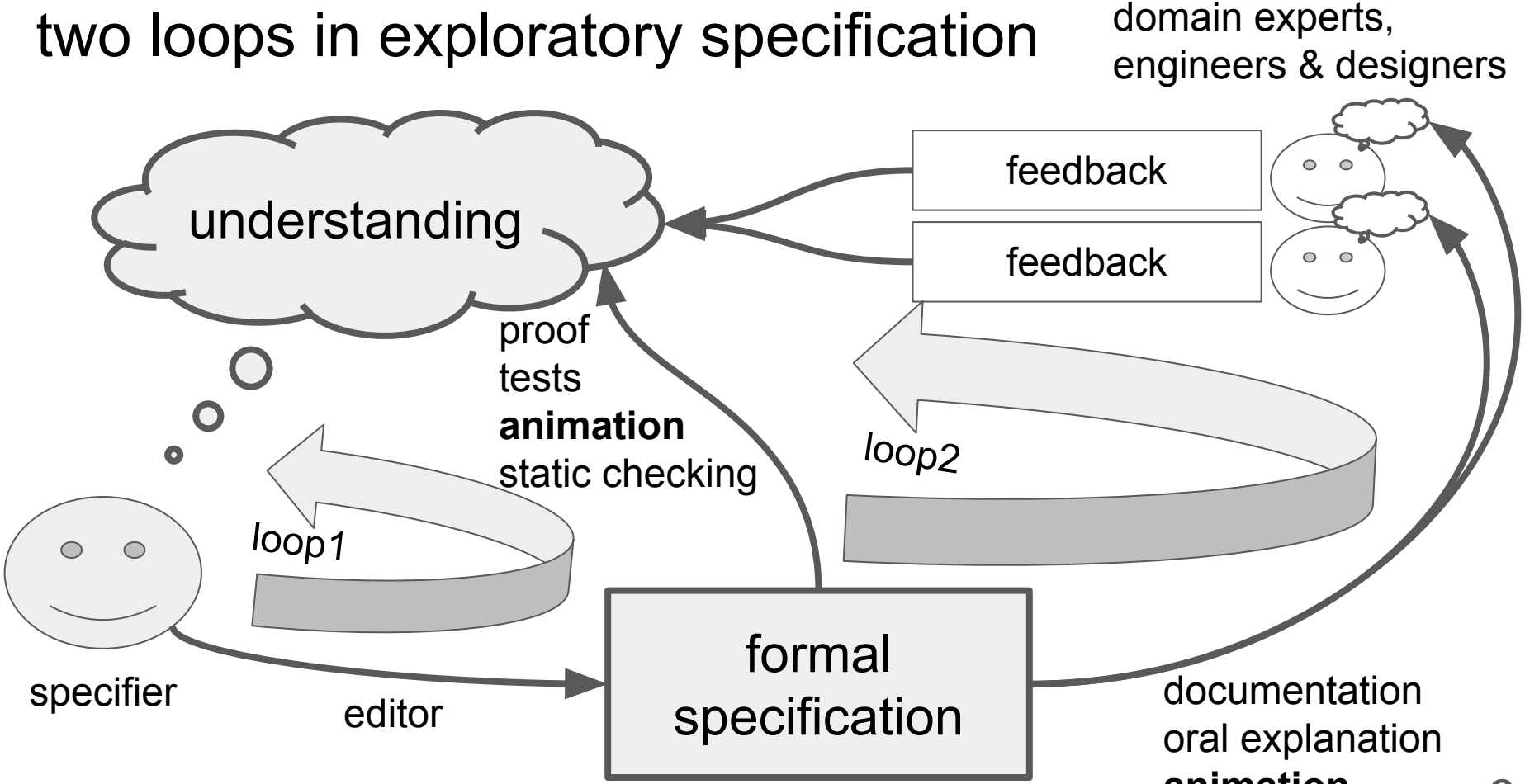
IDE for exploratory specification in VDM-SL

- animation centric
- liveness
- meta-IDE

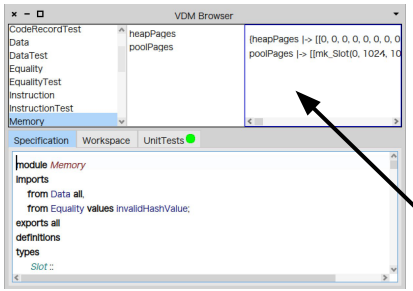
two phases in formal specification

1. exploratory phase
    - learn the problem
    - explore the design space
    - envision the goal
2. rigorous phase
    - refine the specification
    - eliminate ambiguity
    - ensure reliability

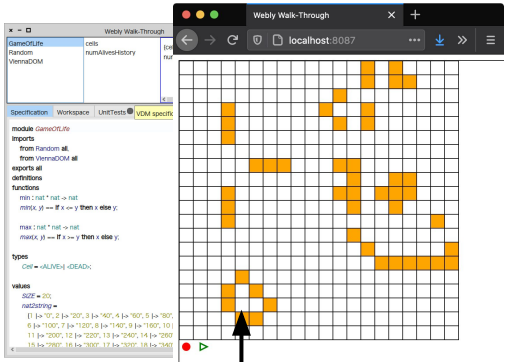
two loops in exploratory specification



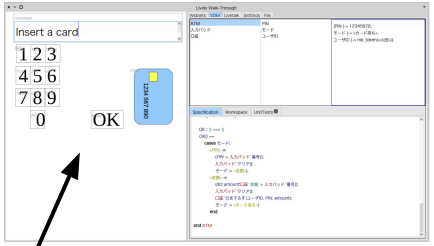
Animation as the first class object:
Every tool on ViennaTalk has an animation as its content



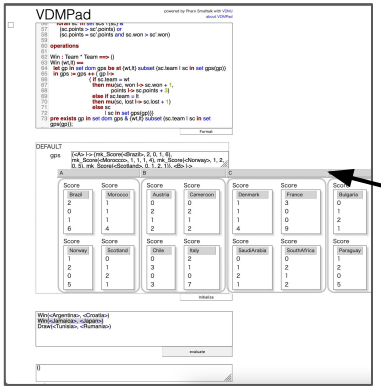
VDM Browser



ViennaVisuals
(Abstract UI with SVG)



Lively Walk-Through
(UI prototyping environment)

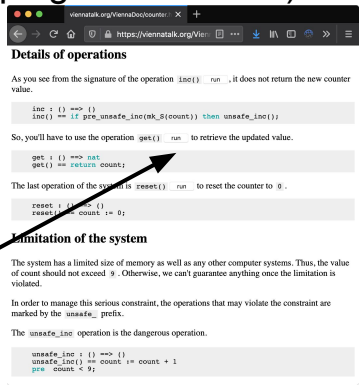


VDMPad (Web IDE)

Animation

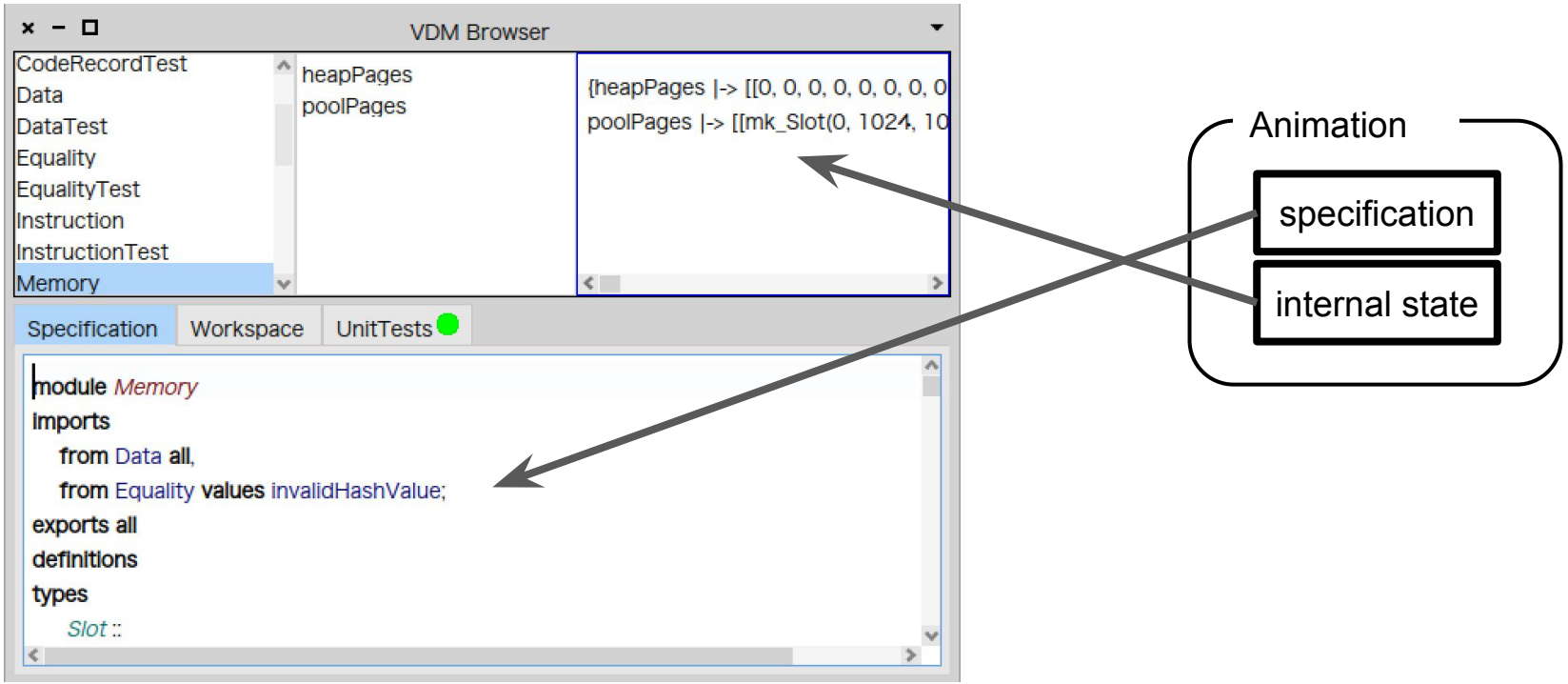
specification

internal state



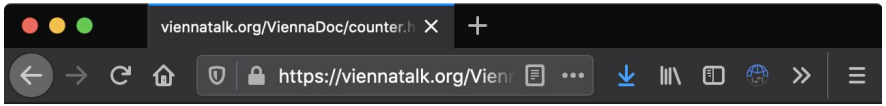
ViennaDoc
(executable
documentation)

Animation in VDM Browser



VDM Browser

Animation in ViennaDoc



Details of operations

As you see from the signature of the operation `inc()` run , it does not return the new counter value.

```
inc : () ==> ()
inc() == if pre_unsafe_inc(mk_S(count)) then unsafe_inc();
```

So, you'll have to use the operation `get()` run to retrieve the updated value.

```
get : () ==> nat
get() == return count;
```

The last operation of the system is `reset()` run to reset the counter to `0` .

```
reset : () ==> ()
reset() == count := 0;
```

Limitation of the system

The system has a limited size of memory as well as any other computer systems. Thus, the value of count should not exceed `9` . Otherwise, we can't guarantee anything once the limitation is violated.

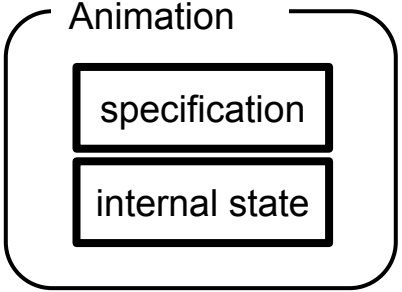
In order to manage this serious constraint, the operations that may violate the constraint are marked by the `unsafe_` prefix.

The `unsafe_inc` operation is the dangerous operation.

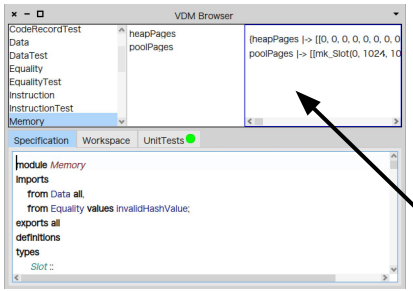
```
unsafe_inc : () ==> ()
unsafe_inc() == count := count + 1
pre  count < 9;
```

evaluatables

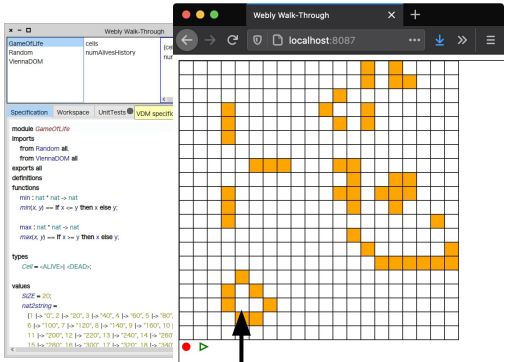
values validated by unit testing



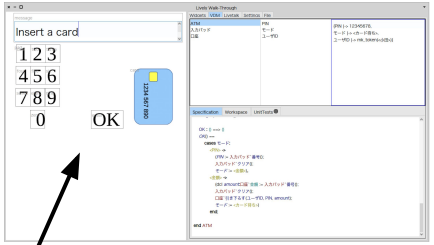
Animation as the first class object:
Every tool on ViennaTalk has an animation as its content



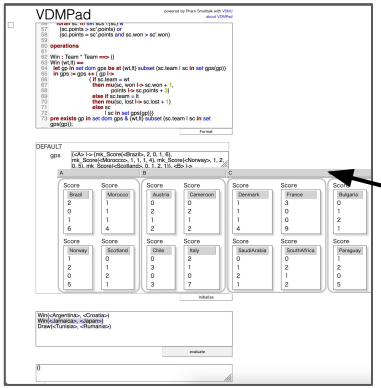
VDM Browser



ViennaVisuals
(Abstract UI with SVG)



Lively Walk-Through
(UI prototyping environment)

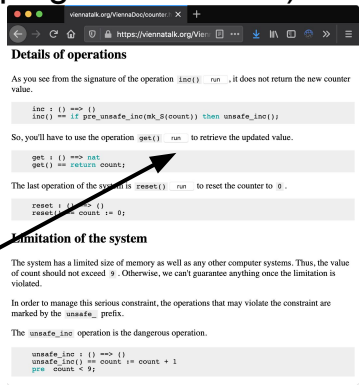


VDMPad (Web IDE)

Animation

specification

internal state



ViennaDoc
(executable
documentation)

User Interface

Modeling a User Interface

User Interface is the system from the user's viewpoint.

Example: review bidding

- A reviewer can make three bids.
- A reviewer cannot bid on the papers authored by the reviewer.
- A reviewer can bid on at most two papers authored by the same person.

functions

```
exceeds_max_papers_from_same_author : set of Paper -> bool
exceeds_max_papers_from_same_author(papers) ==
    exists author in set dunion {elems paper.authors
                                 | paper in set papers} &
        card {paper | paper in set papers
                    & author in set elems paper.authors}
        > MAX_PAPERS_FROM_AUTHOR;
```

Specifying a bidding system

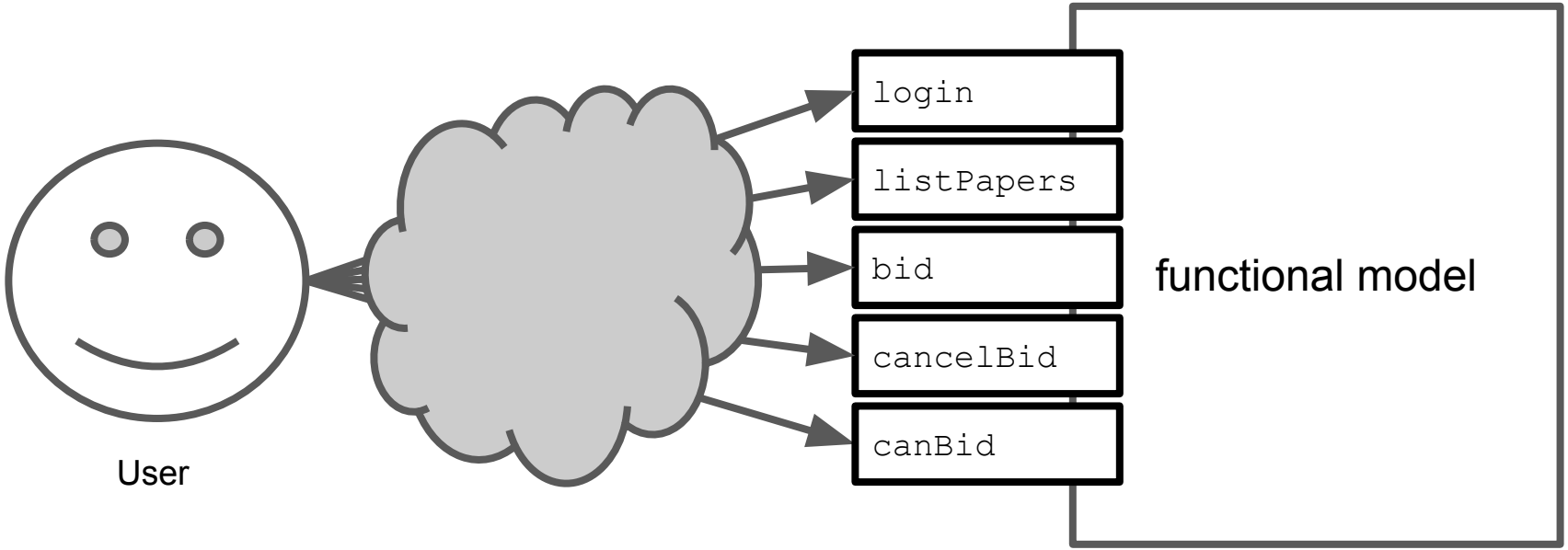
operations

```
login : Person * Password ==> bool
listPapers : () ==> seq of Paper
bid : Paper ==> ()
cancelBid : Paper ==> ()
pure canBid : Paper ==> bool
```

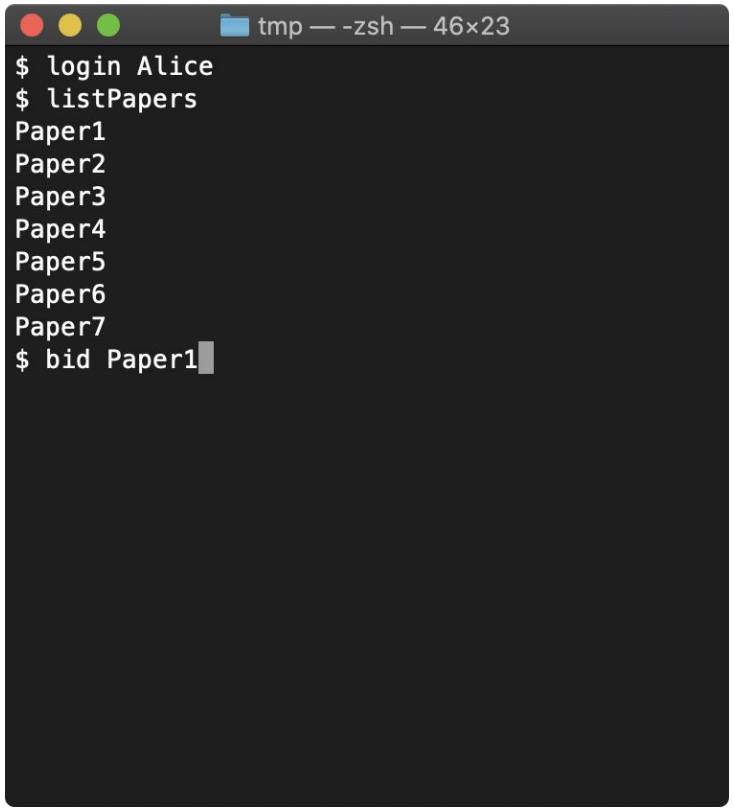
Will these APIs help the user to make an **affordable and legal** bid?

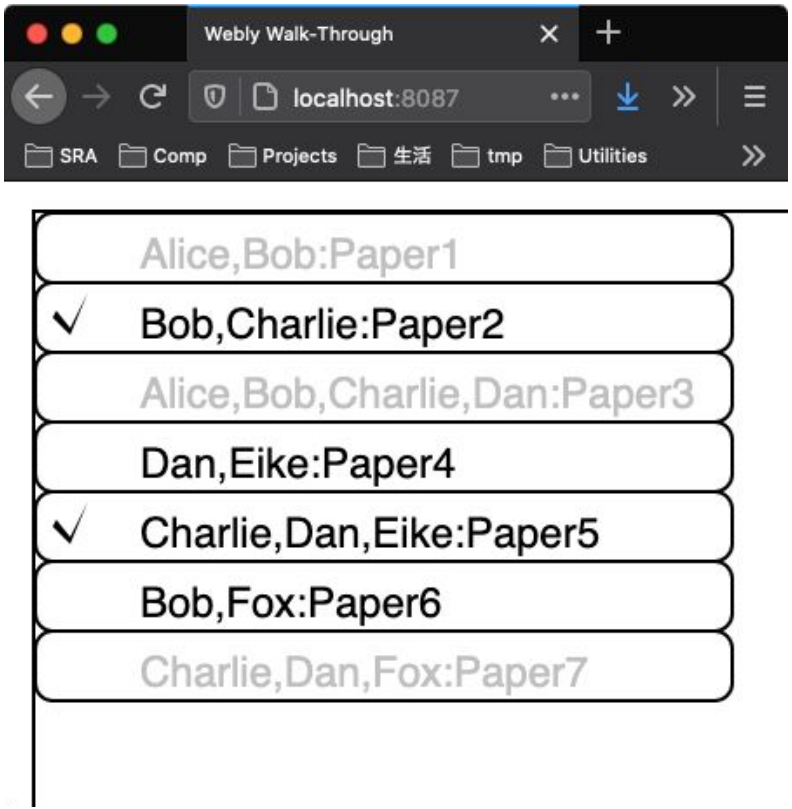
→ Can't tell unless how the information is presented to the user.

functional model in VDM-SL



The same API set, different user interfaces





ViennaVisuals

ViennaVisuals

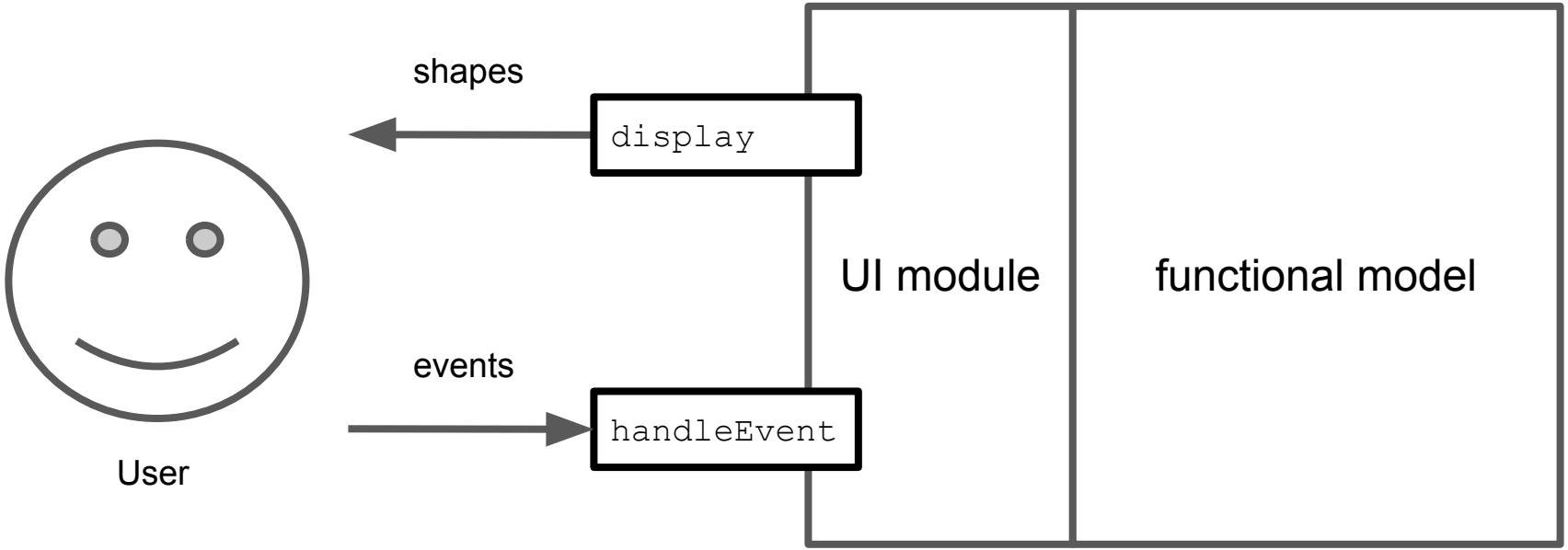
Objectives

- to specify GUIs in VDM-SL
  - using Scalable Vector Graphics (SVG)
- to animate the specified UI
  - on web browsers

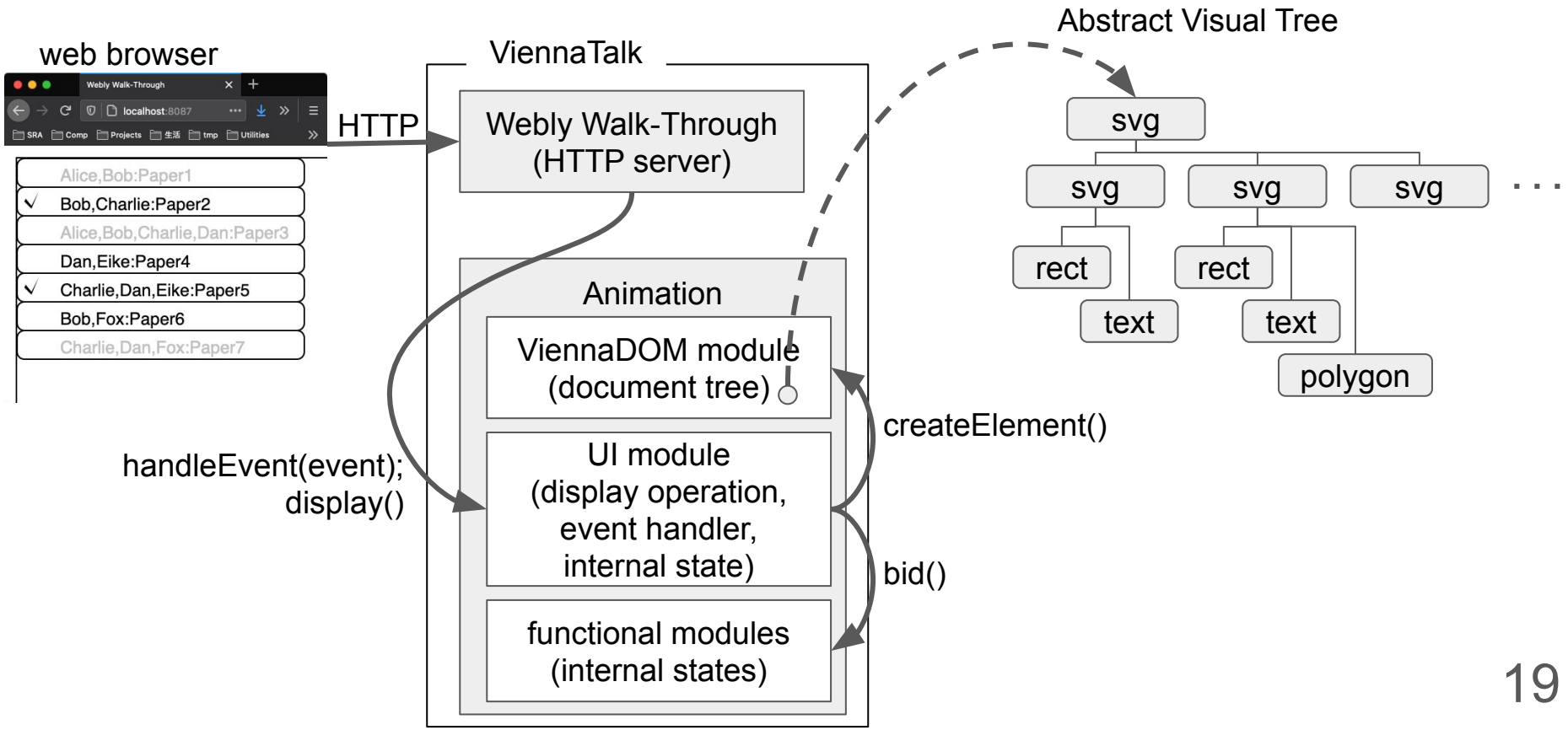
Components

- ViennaDOM module in VDM-SL
- JavaScript library to communicate with ViennaTalk server
- some extensions to HTTP server on ViennaTalk

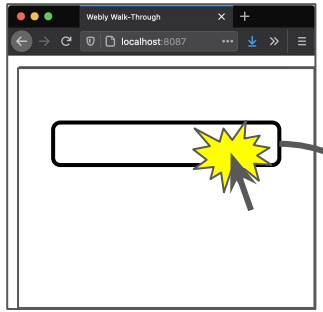
GUI model in ViennaVisuals



ViennaVisuals



Translating a JS event into VDM



event.type="click"
.target=...
.view=...
.buttons=...
.clientX=...
.clientY=...

html5 +
JavaScript

webly.js

GET /event HTTP/2.0

{"type":"click","target":"163","x":"300","y":"10"}

Smalltalk

Webly Walk-Through
(HTTP server)

VDM-SL

Animation

`handleEvent`

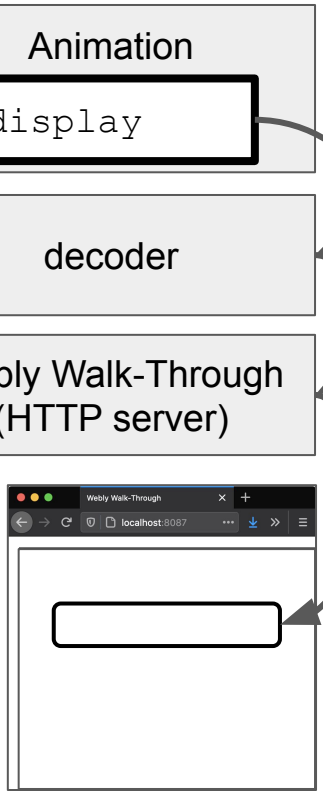
```
mk_MouseEvent(<click>,
    ViennaDOM`getElement(163),
    300, 10)
```

Translating AVT into SVG

VDM-SL

Smalltalk

html5 + JavaScript + SVG




```
mk_TaggedElement("rect", {"fill" |-> "white", "stroke"
|-> "black", "x" |-> "0", "y" |-> 20, "width" |-> 200,
"height" |-> 20, "rx" |-> 5, "ry" |-> 5}, {<click>}, [],
{mk_token(2)}, 163)
```

```
<rect x="0" y="20" width="200" height="20" rx="5"
ry="5" fill="white" stroke="black" identifier_="163" />
```

HTTP response

```
<rect x="0" y="20" width="200" height="20" rx="5"
ry="5" fill="white" stroke="black" identifier_="163" />
```

AST of shape elements

```
types
    Element = TaggedElement | String;
    TaggedElement ::
        name : String
        attributes : map String to [String| real]
        eventHandlers : set of EventType
        contents : seq of Element
        tokens : set of token
        identifier_ : nat;

-- example: mk_TaggedElement("rect", {"fill" |-> "white", "stroke" |-> "black", "x" |->
"0", "y" |-> 20, "width" |-> 200, "height" |-> 20, "rx" |-> 5, "ry" |-> 5}, {<click>}, [],
{mk_token(2)}, 163)
```

Event

```
types
        EventType = <click> | <change>;
        Event = MouseEvent | ChangeEvent;
        MouseEvent :: type : EventType target : TaggedElement x : nat y : nat;
        ChangeEvent :: type : EventType target : TaggedElement value : String;

-- example: mk_MouseEvent(<click>, mk_TaggedElement(...), 40, 30)
```

Example `display` operation

```
operations
    display : () ==> TaggedElement
    display() ==
        (dcl list:TaggedElement := createElement("svg");
        for index = 1 to  len papers
        do
                let paper : Paper = papers(index)
                in
                (dcl itemText:TaggedElement, itemRect:TaggedElement;
                itemRect := ...;
                list := appendChild(list, itemRect);
                if paper in set elems bids then (list := appendChild(list, check(index)));
                list := appendChild(list, itemText));
        return list);
```

Example `handleEvent` operation

```
operations
   handleEvent : Event ==> ()
   handleEvent(event) ==
      cases event:
         mk_MouseEvent(<click>, target, -, -) ->
            let index in set {1, ..., len papers}
            be st hasToken(target, mk_token(index))
            in toggleBid(papers(index)),
         others -> skip
      end;
```

live demo

Limitations

- response time
  - response time = VDM evaluation + transmission latency
- the `display` operation is impure
  - due to management of the mapping between AVT in VDM and SVG elements.
- no concurrency
  - no way to actively push the change of shapes

Summary

ViennaVisuals

- is a tool and a library to **specify** GUIs,
- uses SVG to **animate** the GUI on web browsers,
- requires a GUI module to publish `display` and `handleEvent` operations, and
- defines AST for view and event.