



# **VIEWS**

## **from the past**

Lambert Meertens

Declarative Amsterdam

4 October 2019



# “**VIEWS**” stands for

**V**isual

**I**nterfaces

**E**quipped

**W**ith

**S**emantics

# “**VIEWS**” stands for

**V**isual

**I**nterfaces

**E**quipped

**W**ith

**S**emantics

However, that is a *backronym*.

**backronym** *noun*.

an existing word turned into an acronym by creating an apt phrase whose initial letters match the word.



**In the beginning ...**



## In the beginning ...

.... was the ABC project.

The ABC project (then named *B* project) was started at CWI (then MC) in 1975 by Leo Geurts and myself.

## In the beginning ...

.... was the ABC project.

The ABC project (then named *B* project) was started at CWI (then MC) in 1975 by Leo Geurts and myself.

Aim: to design a new “beginners’ programming language”, which was to be

- suitable for structured programming,
- and yet very simple.



## What is a “beginner”?

- Basic assumption: someone who does not know any programming language and is not familiar with even basic programming concepts
- To serve such users:
  - hide low-level implementation details;
  - instead, provide powerful high-level (task-oriented) features;
  - make the implementation interactive.

# Design by iteration

$$B_0 \rightarrow B_1 \rightarrow B_2 \rightarrow \dots \rightarrow B_\infty.$$

1975:  $B_0$

1978:  $B_1$

1979:  $B_2$

1985:  $B_\infty = \text{ABC}$

## 1982: A new face appears

- In 1982 I accepted a one-year appointment as Visiting Associate Professor at NYU.
- So for that year we were looking for an interim replacement for the *B* project; someone understanding both programming language *design* and programming language *implementation* – someone who was not too senior.
- There was a young fellow in Brighton who had written a book on Pascal implementation and designed a language Newspeak.

# 1982: A new face appears



# 1982: A new face appears



# 1982: A new face appears



*(visual approximation)*



## Fast forward to 1985

- Steven is still with us!
- The ABC implementation is released (for Unix and IBM PC)
- We start designing a dedicated programming environment



# Design objectives

- As before, we wanted to:
  - hide low-level implementation details;
  - instead, provide powerful high-level (task-oriented) features.
- So start with a task analysis:
  - analyze what happens during a typical session.

\$ abc

ABC Release 1.01.01

Copyright (c) Stichting Mathematisch Centrum, Amsterdam, 1985.

>>> ■

\$ abc

ABC Release 1.01.01

Copyright (c) Stichting Mathematisch Centrum, Amsterdam, 1985.

>>> HOW TO REPORT is.palindrome word:

REPORT word = rev word

>>> █

\$ abc

ABC Release 1.01.01

Copyright (c) Stichting Mathematisch Centrum, Amsterdam, 1985.

>>> HOW TO REPORT is.palindrome word:

REPORT word = rev word

>>> PUT "malayam" IN w

>>> █

\$ abc

ABC Release 1.01.01

Copyright (c) Stichting Mathematisch Centrum, Amsterdam, 1985.

>>> HOW TO REPORT is.palindrome word:

REPORT word = rev word

>>> PUT "malayam" IN w

>>> CHECK is.palindrome█

```
$ abc
ABC Release 1.01.01
Copyright (c) Stichting Mathematisch Centrum, Amsterdam, 1985.
>>> HOW TO REPORT is.palindrome word:
        REPORT word = rev word
>>> PUT "malayam" IN w
>>> CHECK is.palindrome
*** There's something I can't resolve in your command
        CHECK is.palindrome
*** The problem is: is.palindrome is neither a refined test nor a
zeroadic predicate
>>> █
```

```
$ abc
ABC Release 1.01.01
Copyright (c) Stichting Mathematisch Centrum, Amsterdam, 1985.
>>> HOW TO REPORT is.palindrome word:
        REPORT word = rev word
>>> PUT "malayam" IN w
>>> CHECK is.palindrome
*** There's something I can't resolve in your command
    CHECK is.palindrome
*** The problem is: is.palindrome is neither a refined test nor a
zeroadic predicate
>>> CHECK is.palindrone w
```

```
$ abc
ABC Release 1.01.01
Copyright (c) Stichting Mathematisch Centrum, Amsterdam, 1985.
>>> HOW TO REPORT is.palindrome word:
        REPORT word = rev word
>>> PUT "malayam" IN w
>>> CHECK is.palindrome
*** There's something I can't resolve in your command
    CHECK is.palindrome
*** The problem is: is.palindrome is neither a refined test nor a
zeroadic predicate
>>> CHECK is.palindrone w
*** There's something I can't resolve in your command
    CHECK is.palindrone w
*** The problem is: is.palindrone hasn't been (properly) defined
>>> █
```

```
$ abc
ABC Release 1.01.01
Copyright (c) Stichting Mathematisch Centrum, Amsterdam, 1985.
>>> HOW TO REPORT is.palindrome word:
        REPORT word = rev word
>>> PUT "malayam" IN w
>>> CHECK is.palindrome
*** There's something I can't resolve in your command
    CHECK is.palindrome
*** The problem is: is.palindrome is neither a refined test nor a
zeroadic predicate
>>> CHECK is.palindrone w
*** There's something I can't resolve in your command
    CHECK is.palindrone w
*** The problem is: is.palindrone hasn't been (properly) defined
>>> CHECK is.palindrome w
```

ABC Release 1.01.01

Copyright (c) Stichting Mathematisch Centrum, Amsterdam, 1985.

>>> HOW TO REPORT is.palindrome word:

REPORT word = rev word

>>> PUT "malayam" IN w

>>> CHECK is.palindrome

\*\*\* There's something I can't resolve in your command

CHECK is.palindrome

\*\*\* The problem is: is.palindrome is neither a refined test nor a zeroadic predicate

>>> CHECK is.palindrone w

\*\*\* There's something I can't resolve in your command

CHECK is.palindrone w

\*\*\* The problem is: is.palindrone hasn't been (properly) defined

>>> CHECK is.palindrome w

\*\*\* Your check failed in your command

CHECK is.palindrome w

>>> █

```
>>> HOW TO REPORT is.palindrome word:
      REPORT word = rev word
>>> PUT "malayam" IN w
>>> CHECK is.palindrome
*** There's something I can't resolve in your command
      CHECK is.palindrome
*** The problem is: is.palindrome is neither a refined test nor a
zeroadic predicate
>>> CHECK is.palindrone w
*** There's something I can't resolve in your command
      CHECK is.palindrone w
*** The problem is: is.palindrone hasn't been (properly) defined
>>> CHECK is.palindrome w
*** Your check failed in your command
      CHECK is.palindrome w
>>> WRITE w
malayam
>>> █
```

```
>>> PUT "malayam" IN w
>>> CHECK is.palindrome
*** There's something I can't resolve in your command
    CHECK is.palindrome
*** The problem is: is.palindrome is neither a refined test nor a
zeroadic predicate
>>> CHECK is.palindrone w
*** There's something I can't resolve in your command
    CHECK is.palindrone w
*** The problem is: is.palindrone hasn't been (properly) defined
>>> CHECK is.palindrome w
*** Your check failed in your command
    CHECK is.palindrome w
>>> WRITE w
malayam
>>> WRITE rev w
mayalam
>>> █
```

```
>>> CHECK is.palindrome
*** There's something I can't resolve in your command
CHECK is.palindrome
*** The problem is: is.palindrome is neither a refined test nor a
zeroadic predicate
>>> CHECK is.palindrone w
*** There's something I can't resolve in your command
CHECK is.palindrone w
*** The problem is: is.palindrone hasn't been (properly) defined
>>> CHECK is.palindrome w
*** Your check failed in your command
CHECK is.palindrome w
>>> WRITE w
malayam
>>> WRITE rev w
mayalam
>>> PUT "malayalam" IN w
>>> CHECK is.palindrome w
```

```
*** There's something I can't resolve in your command
CHECK is.palindrome
*** The problem is: is.palindrome is neither a refined test nor a
zeroadic predicate
>>> CHECK is.palindrone w
*** There's something I can't resolve in your command
CHECK is.palindrone w
*** The problem is: is.palindrone hasn't been (properly) defined
>>> CHECK is.palindrome w
*** Your check failed in your command
CHECK is.palindrome w
>>> WRITE w
malayam
>>> WRITE rev w
mayalam
>>> PUT "malayalam" IN w
>>> CHECK is.palindrome w
>>> █
```



## Does this have to be so awkward?

Why can't the user just edit the incorrect input in the session record as shown on the screen, and let the system react accordingly?

Let's go back to the earliest error message...

```
$ abc
ABC Release 1.01.01
Copyright (c) Stichting Mathematisch Centrum, Amsterdam, 1985.
>>> HOW TO REPORT is.palindrome word:
        REPORT word = rev word
>>> PUT "malayam" IN w
>>> CHECK is.palindrome
*** There's something I can't resolve in your command
    CHECK is.palindrome
*** The problem is: is.palindrome is neither a refined test nor a
zeroadic predicate
>>> █
```

```
$ abc
ABC Release 1.01.01
Copyright (c) Stichting Mathematisch Centrum, Amsterdam, 1985.
>>> HOW TO REPORT is.palindrome word:
        REPORT word = rev word
>>> PUT "malayam" IN w
>>> CHECK is.palindrome
*** There's something I can't resolve in your command
        CHECK is.palindrome
*** The problem is: is.palindrome is neither a refined test nor a
zeroadic predicate
>>>
```

```
$ abc
ABC Release 1.01.01
Copyright (c) Stichting Mathematisch Centrum, Amsterdam, 1985.
>>> HOW TO REPORT is.palindrome word:
        REPORT word = rev word
>>> PUT "malayam" IN w
>>> CHECK is.palindrome w
*** There's something I can't resolve in your command
    CHECK is.palindrome
*** The problem is: is.palindrome is neither a refined test nor a
zeroadic predicate
>>>
```

```
$ abc
ABC Release 1.01.01
Copyright (c) Stichting Mathematisch Centrum, Amsterdam, 1985.
>>> HOW TO REPORT is.palindrome word:
        REPORT word = rev word
>>> PUT "malayam" IN w
>>> CHECK is.palindrome w
*** Your check failed in your command
    CHECK is.palindrome w
>>> █
```

```
$ abc
ABC Release 1.01.01
Copyright (c) Stichting Mathematisch Centrum, Amsterdam, 1985.
>>> HOW TO REPORT is.palindrome word:
        REPORT word = rev word
>>> PUT "malayam" IN w
>>> CHECK is.palindrome w
*** Your check failed in your command
        CHECK is.palindrome w
>>> WRITE w
malayam
>>> WRITE rev w
mayalam
>>> █
```

```
$ abc
ABC Release 1.01.01
Copyright (c) Stichting Mathematisch Centrum, Amsterdam, 1985.
>>> HOW TO REPORT is.palindrome word:
        REPORT word = rev word
>>> PUT "malaya■m" IN w
>>> CHECK is.palindrome w
*** Your check failed in your command
        CHECK is.palindrome w
>>> WRITE w
malayam
>>> WRITE rev w
mayalam
>>>
```

```
$ abc
ABC Release 1.01.01
Copyright (c) Stichting Mathematisch Centrum, Amsterdam, 1985.
>>> HOW TO REPORT is.palindrome word:
        REPORT word = rev word
>>> PUT "malayalam" IN w
>>> CHECK is.palindrome w
*** Your check failed in your command
        CHECK is.palindrome w
>>> WRITE w
malayam
>>> WRITE rev w
mayalam
>>>
```

```
$ abc
ABC Release 1.01.01
Copyright (c) Stichting Mathematisch Centrum, Amsterdam, 1985.
>>> HOW TO REPORT is.palindrome word:
        REPORT word = rev word
>>> PUT "malayalam" IN w
>>> CHECK is.palindrome w
>>> WRITE w
malayalam
>>> WRITE rev w
malayalam
>>> █
```

## How to model this?

- The session record is formed by the zipped merge of the (user-supplied) session input and the (system-produced) session output.
- The session output is a function of the session input.
- The user can edit the session record, but only the input part.
- Upon a user edit (of the screen), the input changes, and then the output also changes.
- Then their zip as seen on the screen is updated.



## More generally

- There is a collection of objects.
- Their values are related by *invariants*.
- An autonomous agent (such as the user) can modify the value of some of these objects.
- If thereby invariants are violated, the system updates other objects to restore them.
- This can trigger a cascade of updates.

# Insights

- Not only can the session be modelled this way, but in fact a complete operating environment.
- By doing this systematically, you obtain a system that can be driven fully by direct manipulation using familiar editing methods.
- Example: no more typing arcane commands like

```
$ mv decalration-08 declaration-08
```

Instead, a user just edits the file name in a table of contents (directory listing).

# Observations

- Applications that follow this paradigm are easy to learn and to use.
- Applications that follow this paradigm but with some exceptions are confusing precisely at these exceptions.
- So it appears that the “Views paradigm” is a good paradigm for application design, not only for an ABC environment, but in general.



## Relationship with the MVC pattern

- The Model–View–Controller pattern is a special, simple case:
  - there are just two objects: the model and the view, whereas the Views paradigm allows a complex network through which updates can cascade;
  - the model–view relation is one-way, whereas the relations of the Views paradigm are generally maintained bidirectionally.



# What about implementing applications?

- Designing an application this way is one thing; implementing the design is another thing.
- The point is that it is very easy to overlook some case where a change would invalidate an invariant. This leads to an inconsistent system state, causing confusion or worse.
- It is also costly to define all possible updates, also for cases that the designer expects to be rare, so it is alluring to disallow some types of changes for no particular good reason, thus creating a potential source of user frustration.



# What about implementing applications?

- Designing an application this way is one thing; implementing the design is another thing



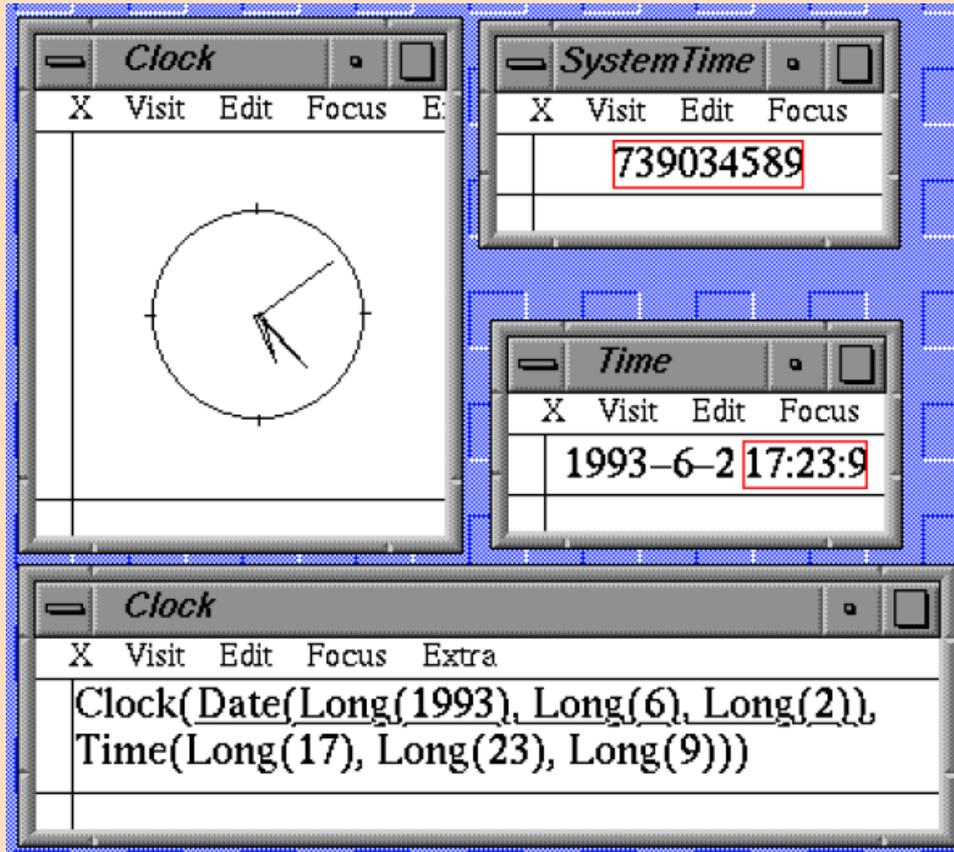
# What about implementing applications?

- Designing an application this way is one thing; implementing the design is another thing – or is it?

# What about implementing applications?

- Designing an application this way is one thing; implementing the design is another thing – or is it?
- Why not derive the implementation automatically from a description of the design? Surely, that should be possible.
- Thus, the Views project was born.

# Multiple views on one entity



The views correspond to presentations of different objects linked by invariants.

# Clock example

- The analog clock presentation could have been defined by (simplified):

```
AnalogClock(h, m, s, size) =  
  Combine[ClockFace size, ClockHands(h, m, s, size)]  
  
ClockHands(h, m, s, size) = Combine[  
  UprightLine 0.80*size RightRotated s*(360/60),  
  UprightLine 0.65*size RightRotated m*(360/60),  
  UprightLine 0.50*size RightRotated (h + m/60)*(360/12)  
]
```

- This is not a one-way street; the definition defines a network schema that links objects

# How to specify a new application

- Define any new data types, including their default representations
- Declare objects as instances of types
- Specify invariants between objects in the form of equations:

*expression1 = expression2*

# Views on Views

- Steven has written:

In the late 80's I built a system with my group, Views, that if you saw it now you would call a browser. It had extensible markup, stylesheets, vector graphics, client-side scripting, everything you would recognise as the web now (though it didn't run over TCP/IP).

- While not untrue, I feel that this does not do justice to how innovative the whole approach was.



# Views was an early example of a declarative paradigm

- Not “how”, but “what”; the only code that was “executed” were the invariants.
- The system had many aspects of the Web, such as hyperlinks, but many other aspects of Views were not present in the Web technology, such as the ease of interoperability of independently developed applications.
- Applications following the Views paradigm (but implemented with ad hoc methods) have become more consistent, but exceptions still abound.

# Further reading

<https://homepages.cwi.nl/~steven/views/>









