ClickScript

a visual programming language in the browser

by lukas naef 2009

Index

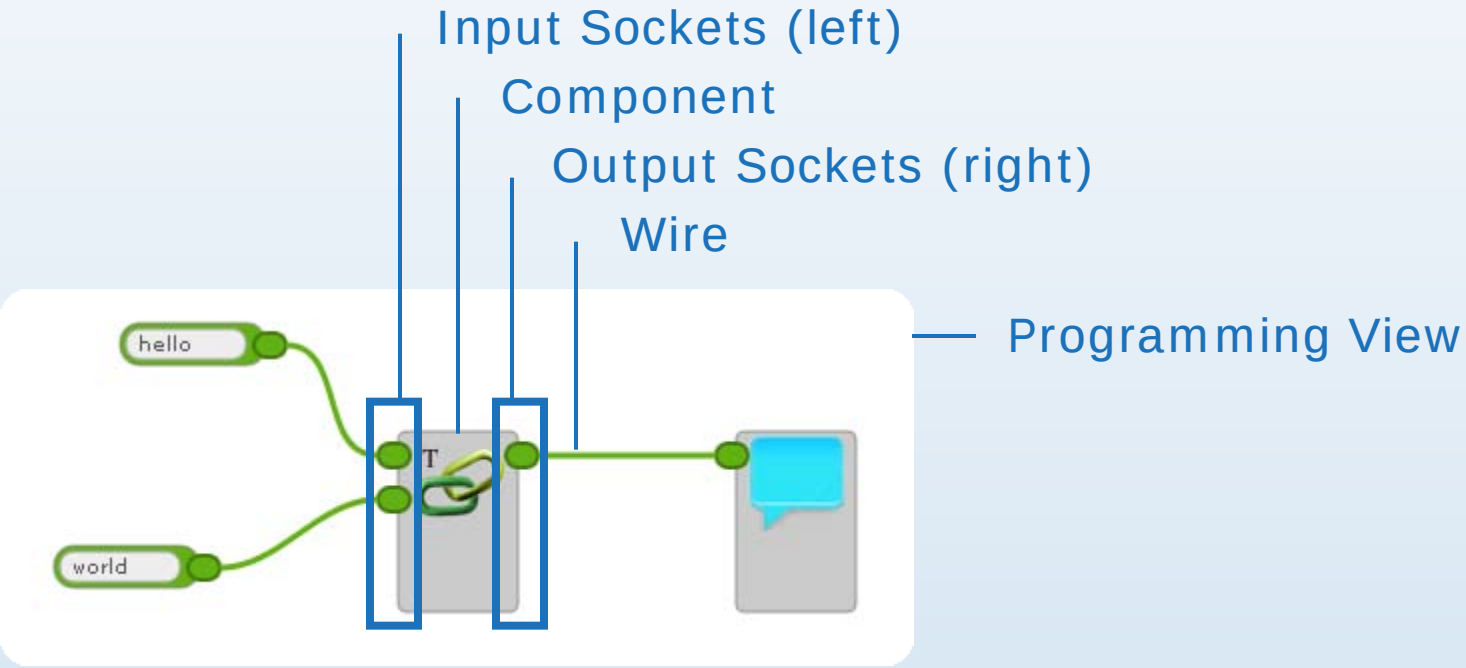
- Introduction
- Language
- Execution Strategy
- Extensibility
- Conclusion
- Demo

Goal

The goal of this thesis is to create an easy to use programming environment that motivates people to delve into the realm of software by developing their own programs.

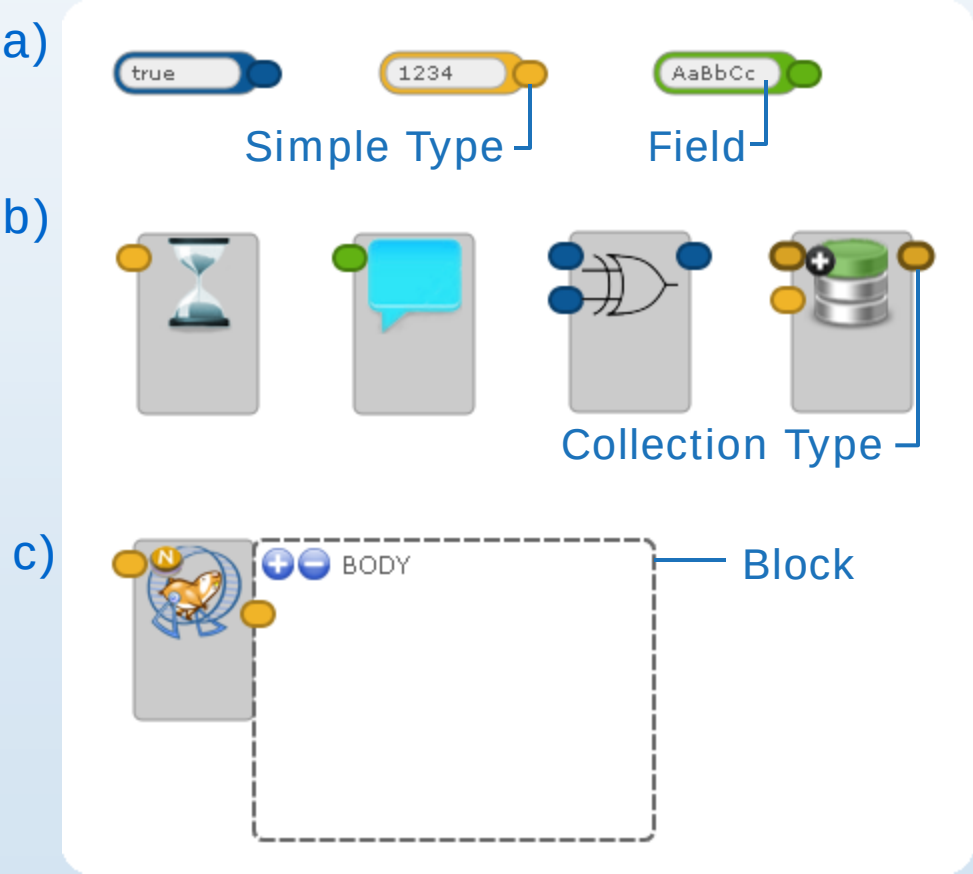
- visual programming language
- running entirely in the browser
- easy to use
- easy to extend

Simple Script



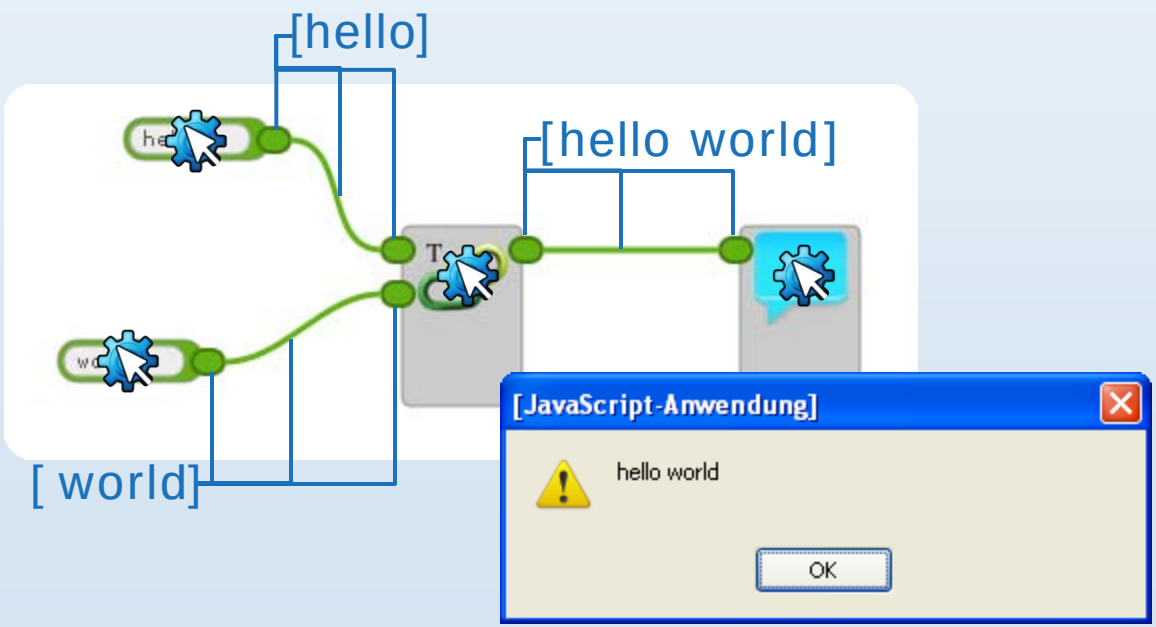
Components

- Types
  - a) Primitives
  - b) Modules
  - c) Statements
- Sockets
  - Colors
  - Borders



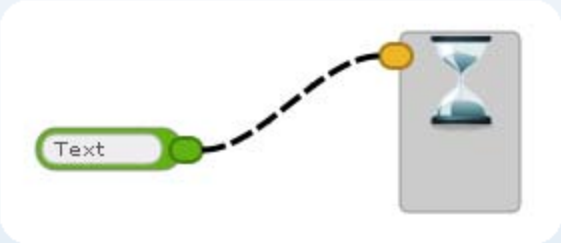
Simple Execution Strategy

- Components execute as soon as all their inputs are loaded.

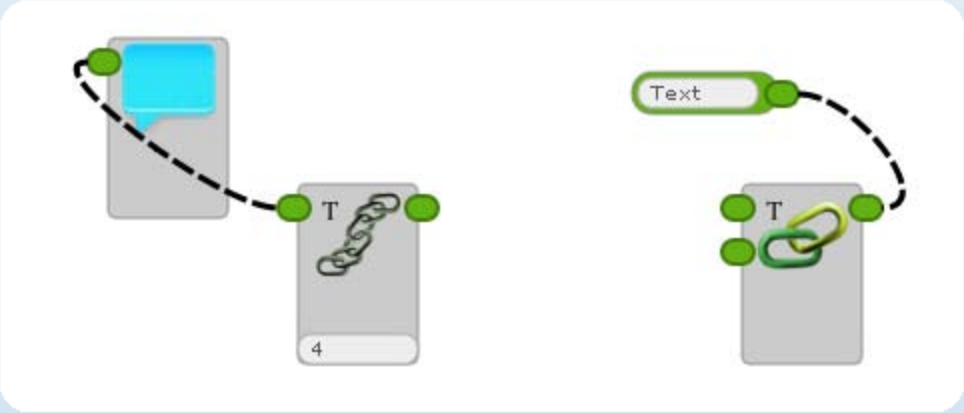


The Four Wiring Rules

- Rule 1 – Same Types

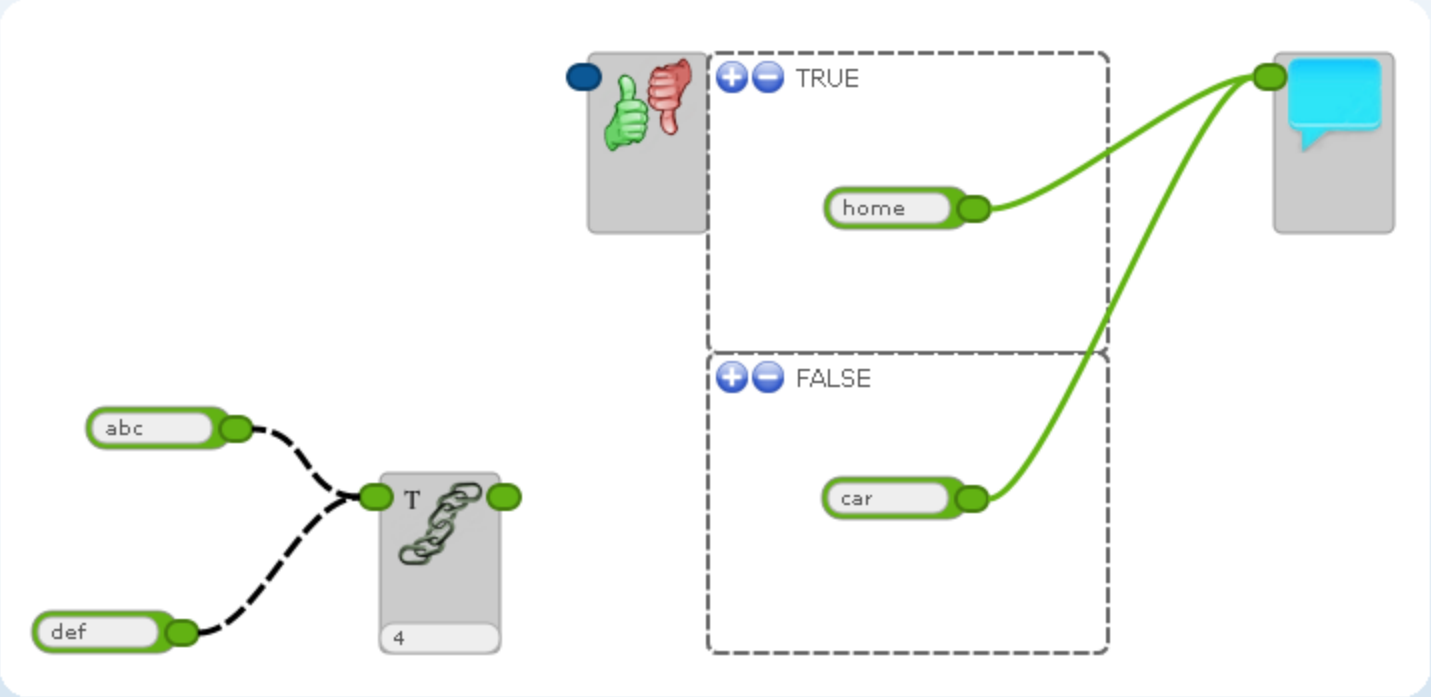


- Rule 2 – Input to Output



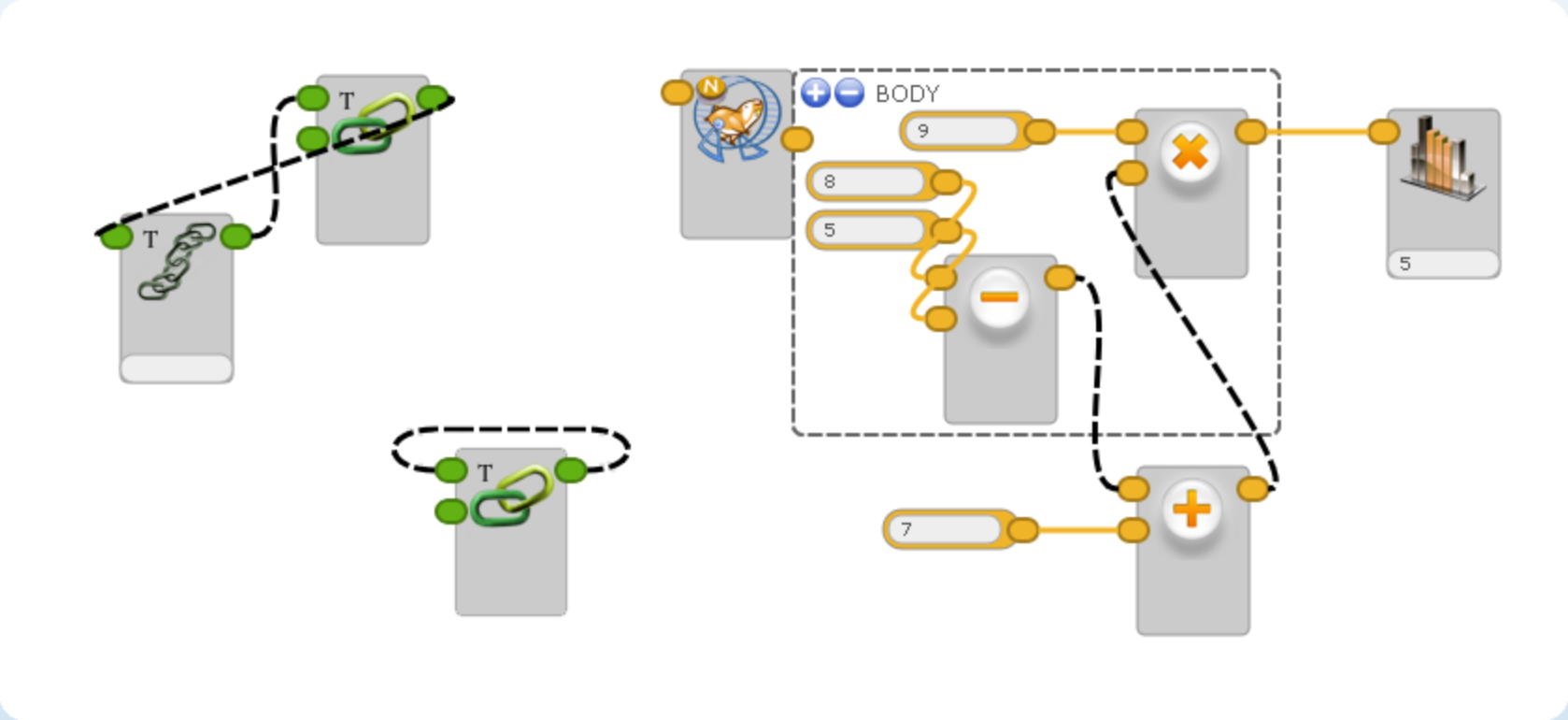
The Four Wiring Rules

- Rule 3 – One Wire per Input*

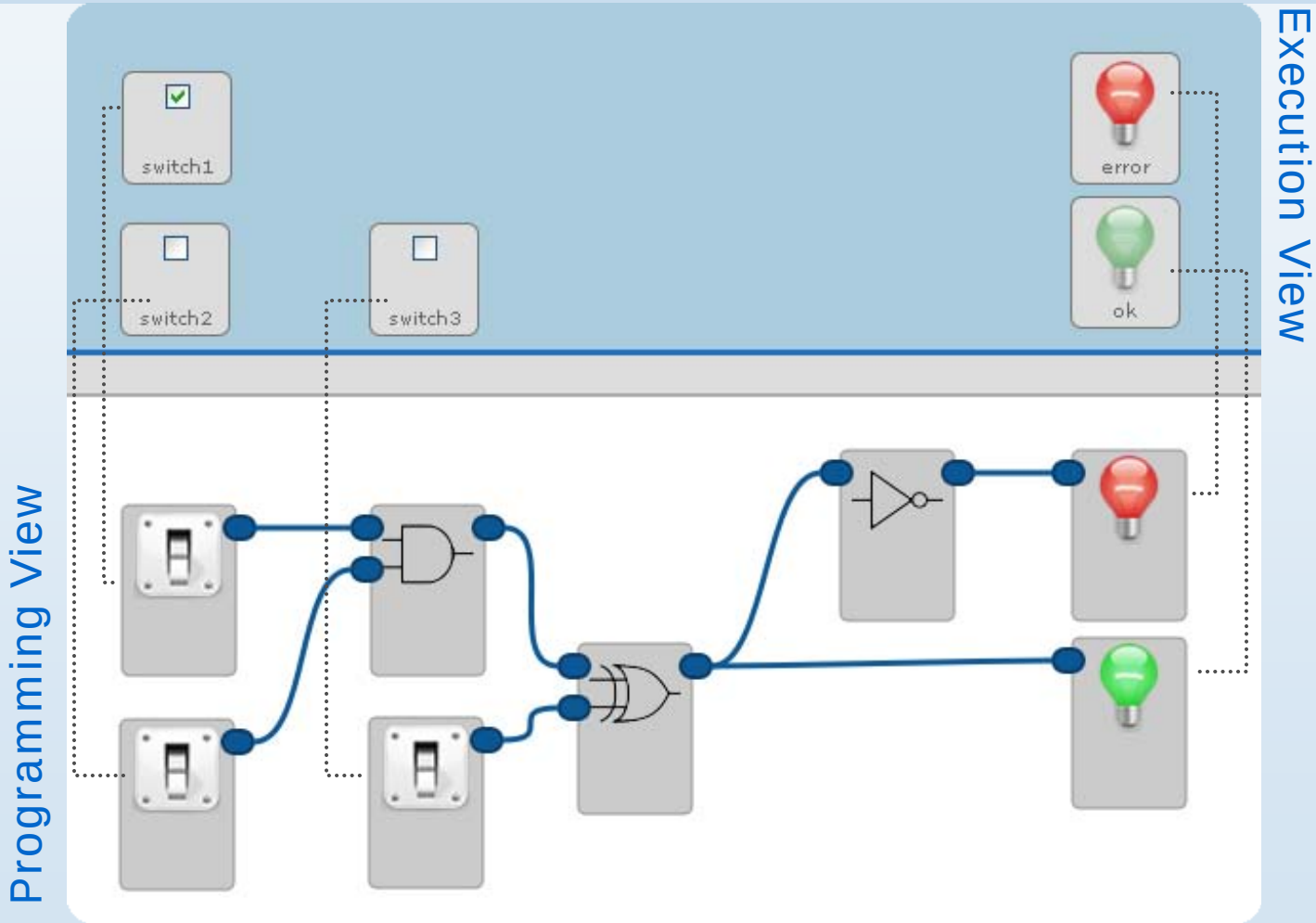


The Four Wiring Rules

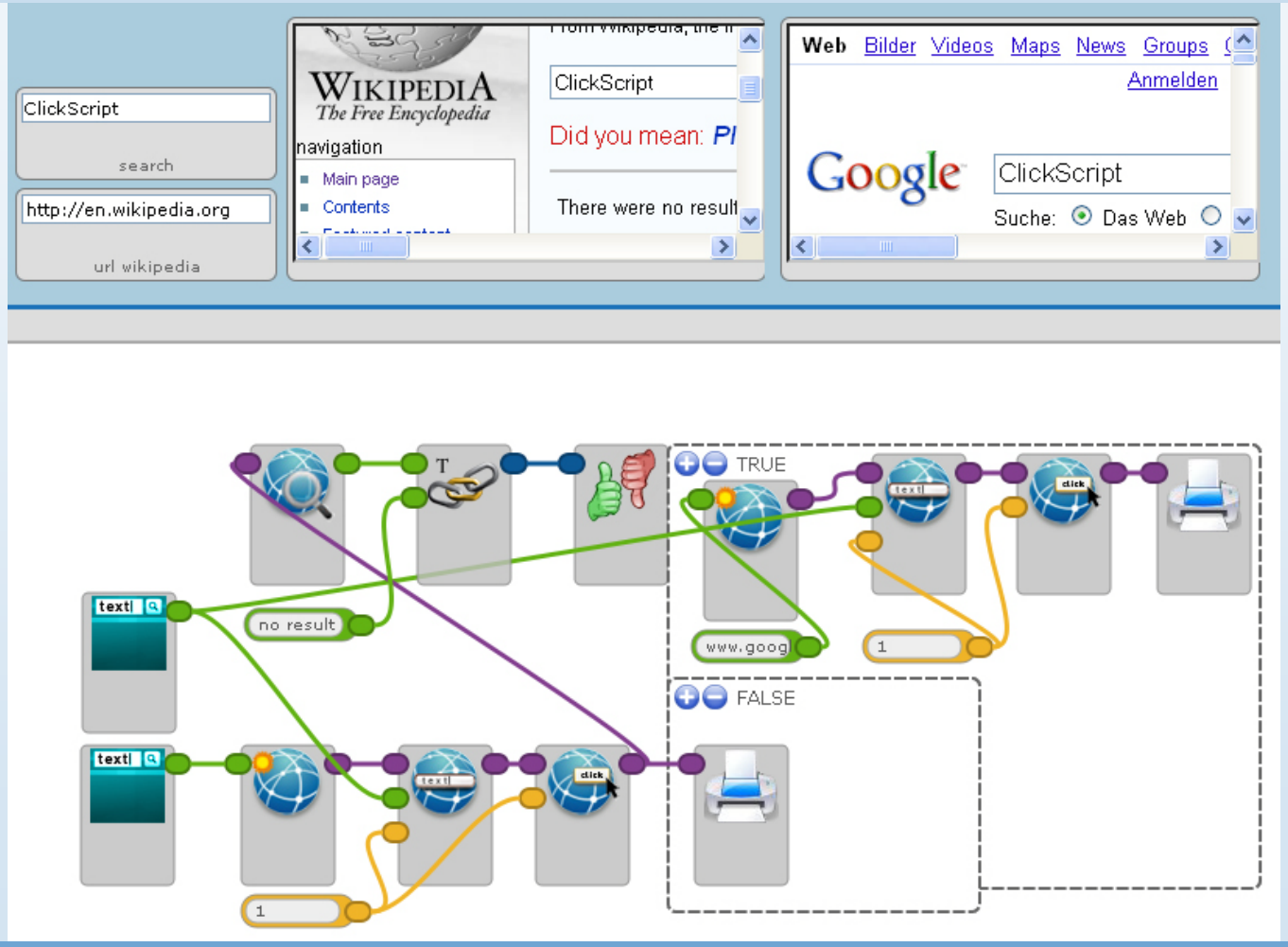
- Rule 4 – No Cyclic Graph



Execution View

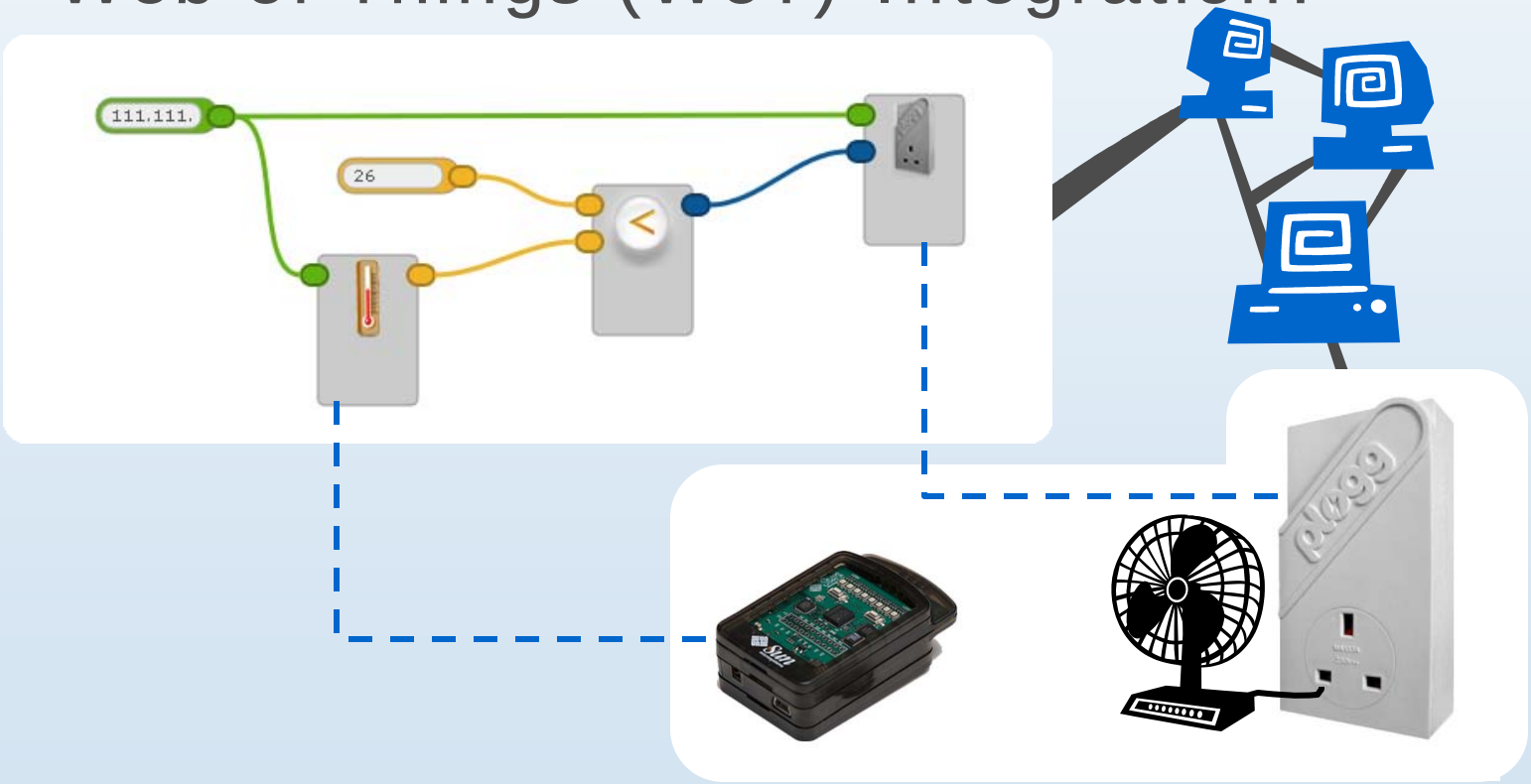


Example


ClickScript
search
http://en.wikipedia.org
url wikipedia
WIKIPEDIA
The Free Encyclopedia
navigation
Main page
Contents
ClickScript
Did you mean:
There were no result
Web Bilder Videos Maps News Groups
Anmelden
Google
ClickScript
Suche: Das Web
TRUE
FALSE
text
click
no result
www.goog
1

Extensibility

- Web of Things (WoT) Integration:



Add Own Component

```
csComponentContainer.push({
  name : "cs.math.is_smaller",
  description : "RESULT is True if NUMBER1 is smaller than NUMBER2",
  inputs :[{ name: "NUMBER1",
             type: "cs.type.Number"},
           { name: "NUMBER2",
             type: "cs.type.Number"}],
  outputs:[{ name: "RESULT",
             type: "cs.type.Boolean"}],
  image: "math/smaller.png",
  exec : function(state){
    var number1 = parseFloat(state.inputs.item(0).getValue());
    var number2 = parseFloat(state.inputs.item(1).getValue());
    var result  = (number1<number2);
    state.outputs.item(0).setValue(result);
  }
});
```

1. Read Input Socket
2. Execute Component
3. Write Output Socket

Component Definition Object

Extensibility

- implemented so far:
  - **statements**: if, for, foreach, until-stop, sequence
  - **primitives**: string, number, boolean
  - **ide**: popup, timer, textfield, display
  - **string**: concatenation, match,...
  - **math**: add, subtract, random, diagram,...
  - **converter**: str2num, num2str
  - **browser**: open page, fill-form,...
  - **web of things**: switch, temperature
  - **collection**: pipe, add, has, remove
  - **logic**: and, or, xor, not,...
  - **robotic**: lights, heating, cooler, car, sound,...

Vision

- Exchange Scripts: Save and Load
- Compile – Execute somewhere else
- Introduction to Programming

Conclusion

- visual programming language
- running entirely in the browser
- easy to use
- easy to extend

We got:

- An interpreted visual data-flow programming language written in JavaScript.
- Introduction of 2h → create own scripts.
- Included WoT Components in 2h
- Intrinsically motivated ClickScript programmers.

Demo

- ClickScript Firefox Extension vs. Online Version
- IDE
  - Menu Bar
    - Options
    - Tutorial
    - Exercises
  - Library
  - Execution View
  - Programming View
  - Console
- Features
  - Repeated Run
  - Labeling
  - Component Description
  - Execution View Highlighting
- Example Script

Questions?

- do not resist – get it now…

www.clickscript.ch

Additional Slides

Advanced Execution Strategy

1. All input wires are loaded (its predecessors has been executed)
2. Its parental block is activated
3. For all input wires originating in a block, the block has to be finished

Advanced Component Def Obj

```
csComponentContainer.push({
name : "cs.web.things.switch",
description : "switch on or off",
inputs : [{ name: "IP",
            type: "cs.type.String"},
          { name: "on/off",
            type: "cs.type.Boolean"}],
outputs: [],
image: "web/things/plogg.png",
exec : function(state){
  this.setAsync();
  var ip = state.inputs.item(0).getValue();
  var aurl = "http://"+ip+":8082/EnergyMonitor/ploggs/Laptop/status.html";
  var onoff = state.inputs.item(1).getValue() ? "on" : "off";
  var component = this;
  $.ajax({
    url: aurl,
    type: "POST",
    data: ({status : onoff}),
    success: function(html){
      alert("status of fan : " +onoff);
      component.finishAsync();},
    error: function(msg){
      throw Error("Error on: "+aurl);}
  });}
});
```