René Vincent Jansen

ELMO

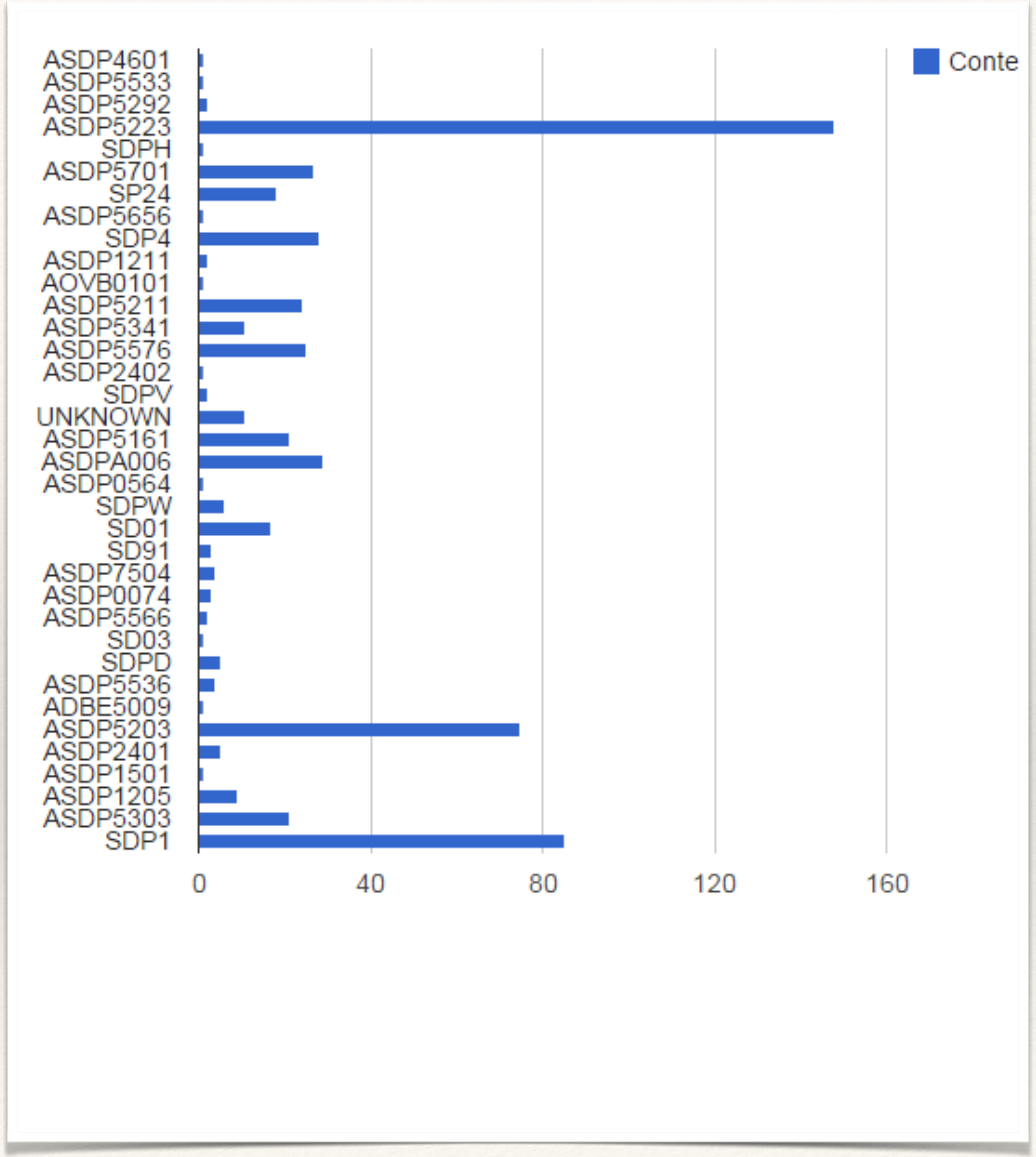
A live payment streams monitoring web application with NetRexx and JSON

27th International Rexx Language Symposium - Tampa, Florida, August 2016

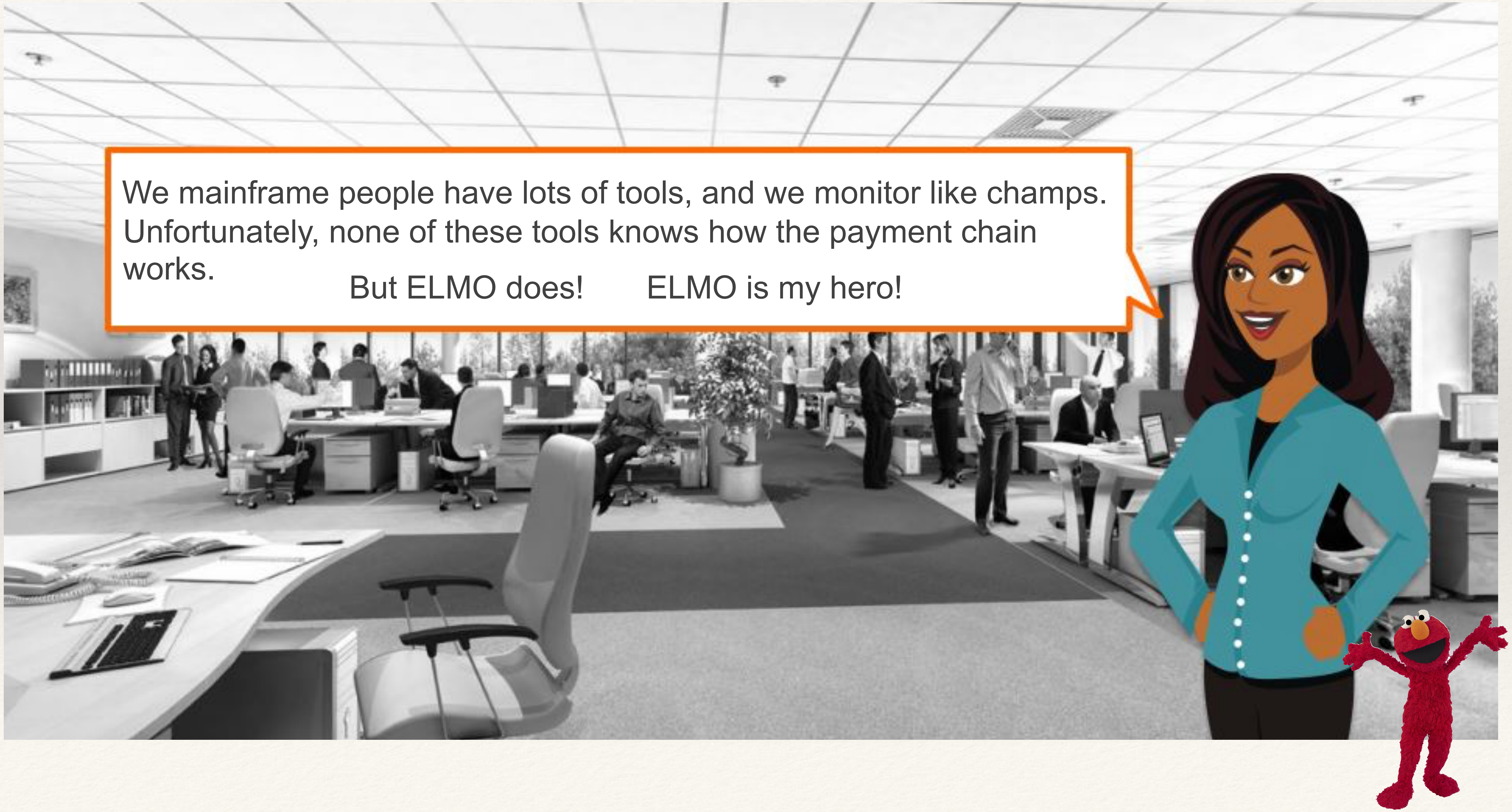
ELMO

Agenda

- What ELMO does
- Where is ELMO?
- Interacting with ELMO
- How to make ELMO do things



Linda says:


We mainframe people have lots of tools, and we monitor like champs. Unfortunately, none of these tools knows how the payment chain works.
But ELMO does!
ELMO is my hero!

ELMO's Purpose

- Capture application-level payment order status changes
- Monitor thresholds and show transgressions
- Assure the payment system delivers the required throughput
- Automate tedious manual work so there is more time to play!

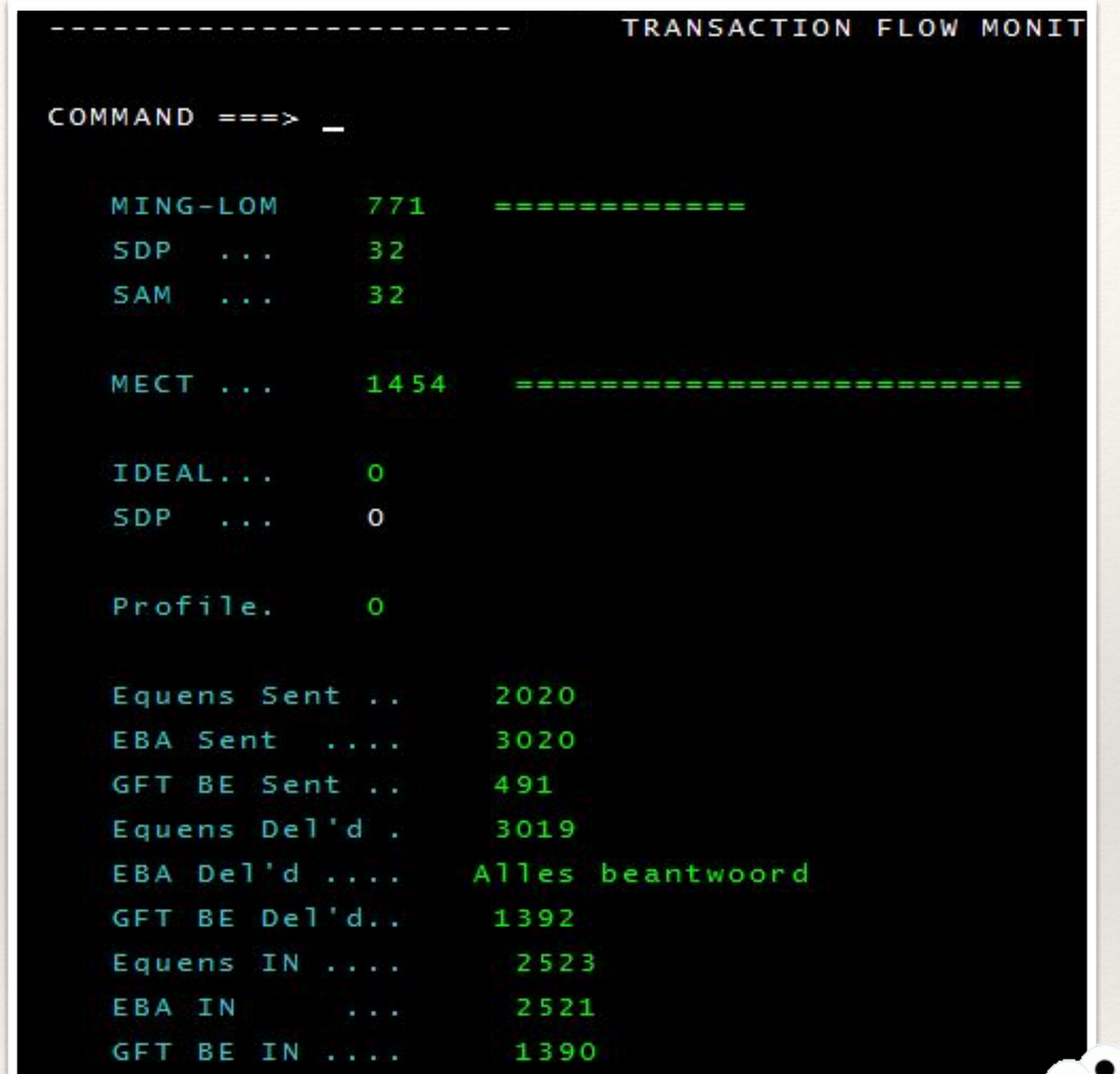
ELMO the 1st generation

A bit of history

The need for ELMO was identified in October 2014. The first version was delivered on November 12th, 2014, after a long night of Company Hackathon.

This version ran on a 3270 terminal as an ISPF application. Immediately, a version that runs on a smartphone was requested.

This version was written in Classic REXX. The DB2 queries were reused for ELMO-*ng* - the new generation, as was much of the logic.



```
----------------------      TRANSACTION FLOW MONIT

COMMAND ===> _

   MING-LOM     771   ============
   SDP   ...    32
   SAM   ...    32

   MECT ...     1454   ========================

   IDEAL...     0
   SDP   ...    0

   Profile.     0

   Equens Sent ..     2020
   EBA Sent   ....    3020
   GFT BE Sent ..     491
   Equens Del'd .     3019
   EBA Del'd ....    Alles beantwoord
   GFT BE Del'd..     1392
   Equens IN ....      2523
   EBA IN      ...     2521
   GFT BE IN ....      1390
```

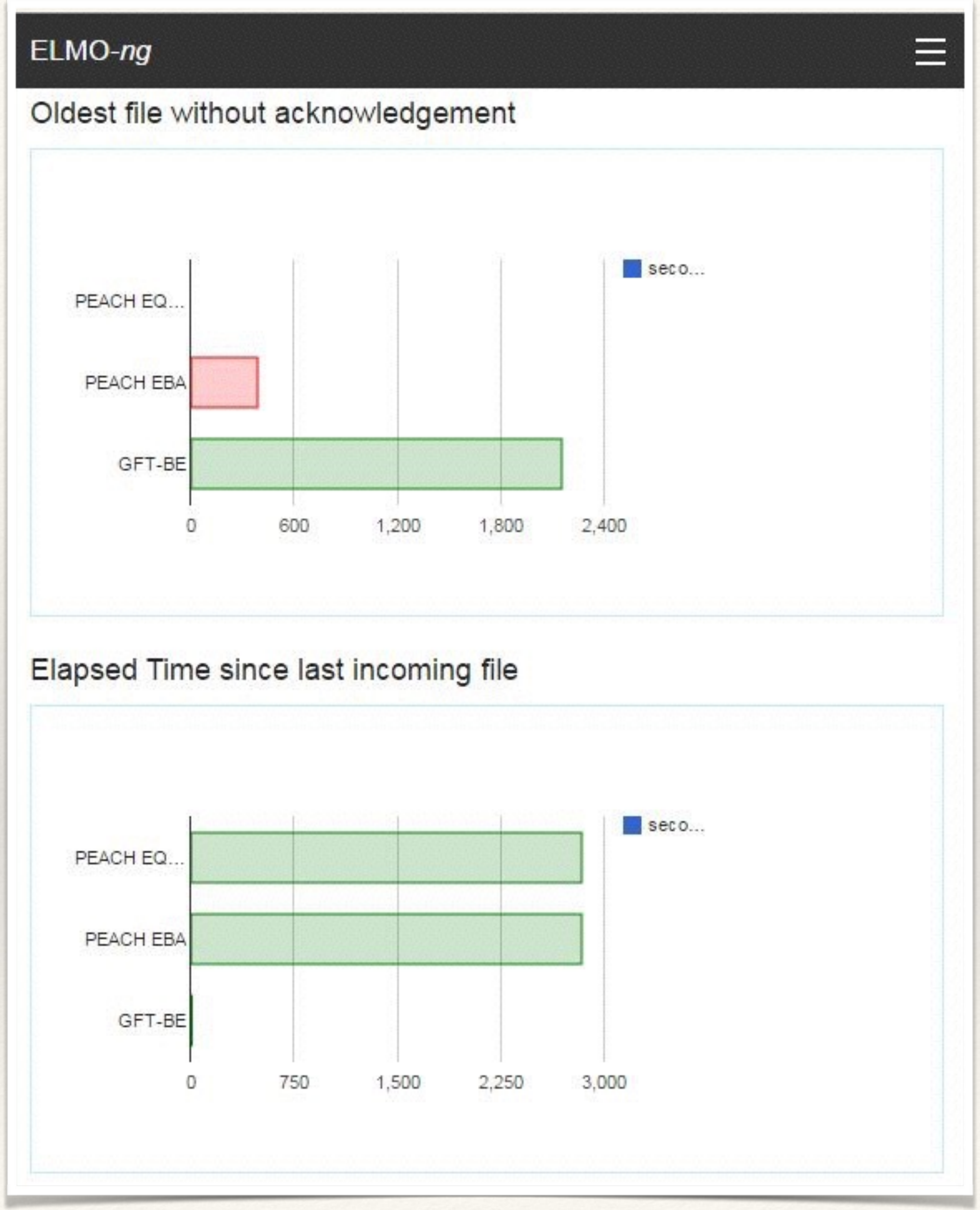
ELMO The New Generation

ELMO-*ng*

The new generation runs on open source Tomcat, with an HTML5 GUI and a Java backend, written in NetRexx.

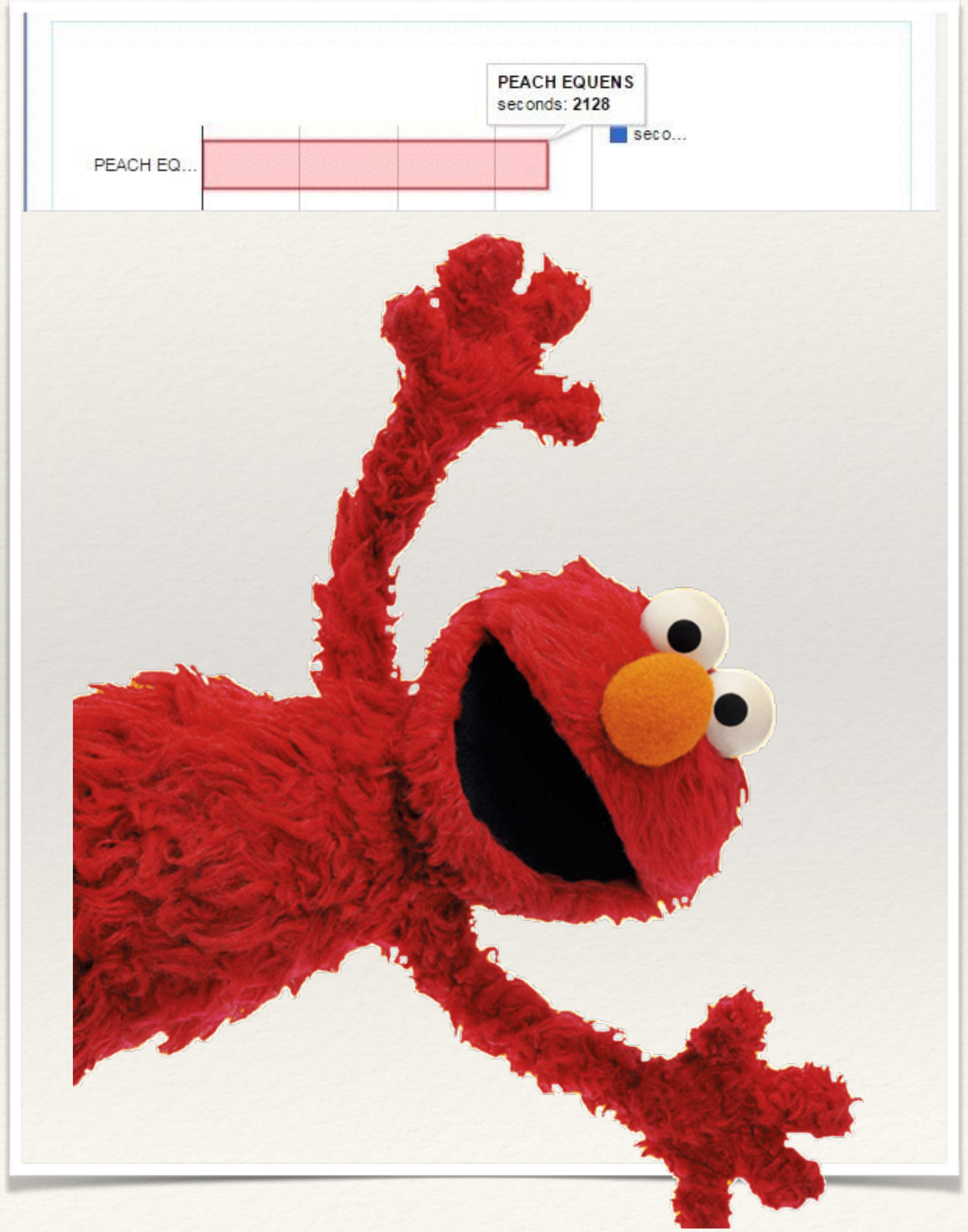
It tries to capture the status of the payments in clear graphics.

The picture on the right shows the status of file transfers to third parties.



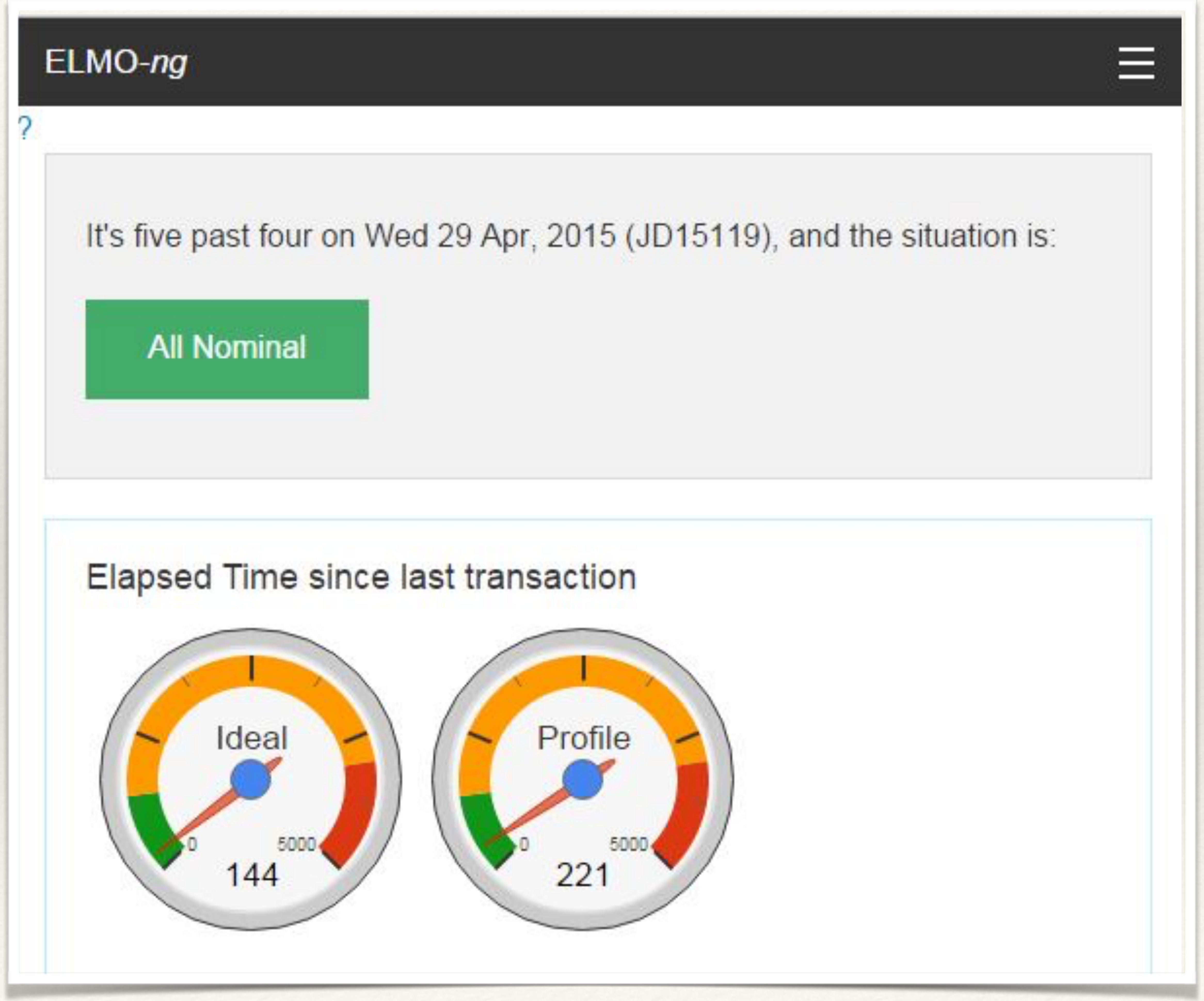
Delays are flagged

```
method fileconf() returns ArrayList
/* rexx die beoordeelt of er te lang geen confirm is binnengekomen */
/* op files die we uitgestuurd hebben*/
  equconfirm_waittime = 3600
  ebaconfirm_waittime = 1800
  gftbeconfirm_waittime = 5400
  EBAconf  = this.da.uitgb22(993)
  GFTBEconf = this.da.uitgb22(994)
  EQUENSconf = this.da.uitgb22(995)
  a = ArrayList()
  -- /*equens logica*/
  -- /*tussen 1700 en 0000 geen terugmeldingen van equens*/
  if EQUENSconf.getwaittime() <> 99999 & (date("W") <> "Saturday" & date("W") <> "Sunday" ) -
  & (time('S') > 3600 & time('S') < 61200) then do
        if EQUENSconf.getwaittime() < equconfirm_waittime then EQUENSconf.setcolor("green")
        if EQUENSconf.getwaittime() > 1000 then EQUENSconf.setcolor("orange")
  end
  else do
    EQUENSconf.setColor("green")
    EQUENSconf.setWaittime(0)
  end
  -- /*EBA logica*/
  if EBAconf.getwaittime() <> 99999 then do
    if (date("W") <> "Saturday" & date("W") <> "Sunday") then do
       if date("W") <> "Monday" 3 time('S') > 25200 then do
         if EBAconf.getwaittime() < ebaconfirm_waittime then EBAconf.setcolor("green")
         if EBAconf.getwaittime() > 1000 then EBAconf.setcolor("orange")
       end
    end
  end
  else do
    EBAconf.setColor("green")
    EBAconf.setWaittime(0)
  end
  -- /*gft be logica*/
  if GFTBEconf.getwaittime() <> 99999 then do
    if (date("W") <> "Saturday" & date("W") <> "Sunday" ) then do
      if GFTBEconf.getwaittime() < gftbeconfirm_waittime then GFTBEconf.setcolor("green")
      if GFTBEconf.getwaittime() > 3600 then GFTBEconf.setcolor("orange")
    end
  end
  else do
    GFTBEconf.setColor("green")
    GFTBEconf.setWaittime(0)
  end
  a.add(EQUENSconf)
  a.add(EBAconf)
```



Elmo Speed Gauge

- The green status button changes color and links to the problem when somewhere in ELMO a threshold has tripped
- Two large speed gauges indicate the number of milliseconds since the last transaction of the specified type entered SDP



It's five past four on Wed 29 Apr, 2015 (JD15119), and the situation is:

Elapsed Time since last transaction

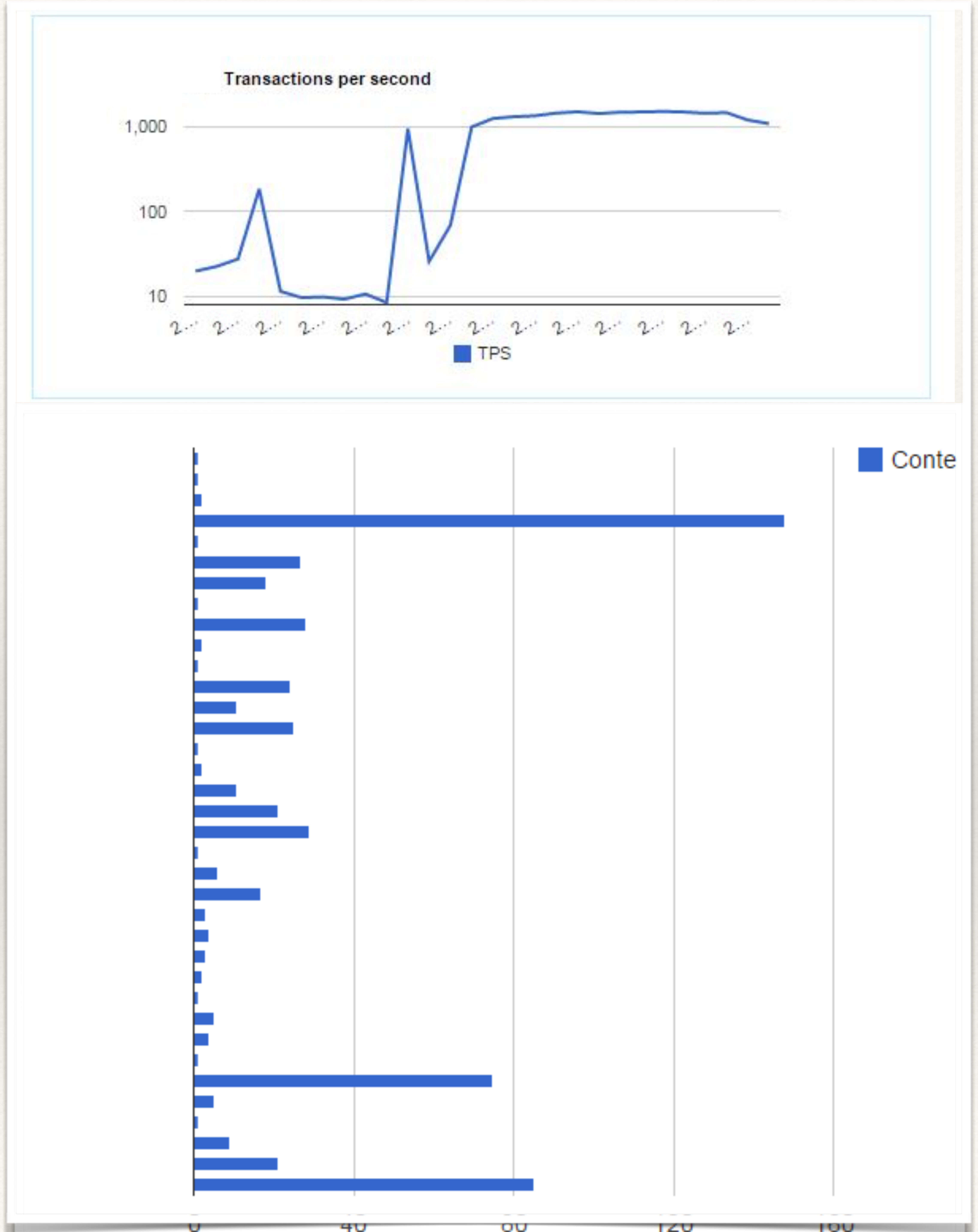
Velocity and Contention Graphs

Velocity and locking

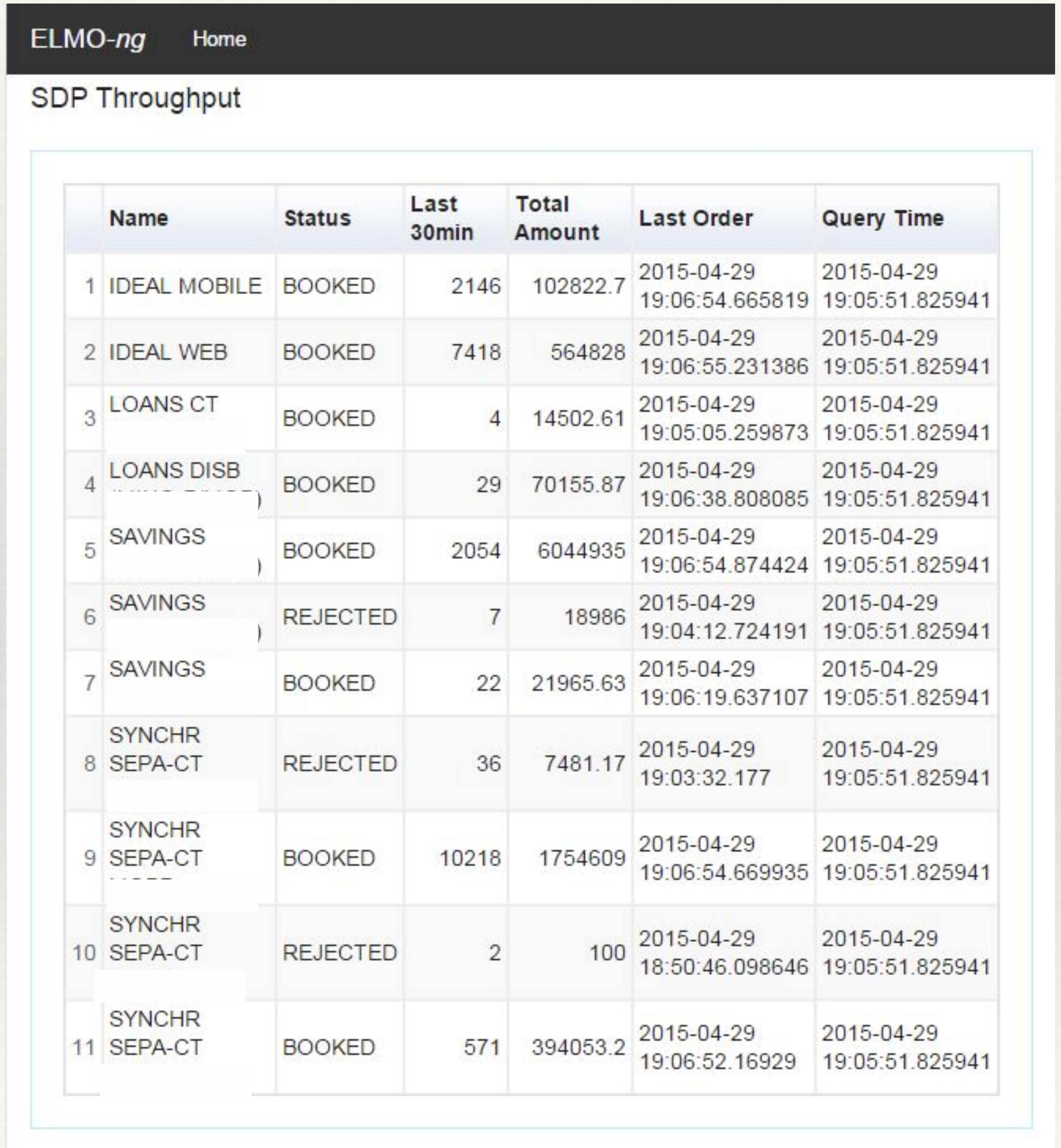
A modern mainframe is capable of sustained periods of high-velocity transaction processing, necessitated by the nature and volume of *batch payments* and *direct debits*.

At any moment we can see the transaction rate in created *payment_id's* per second, and the incurred database contention, split out in locking winners and victims.

This forms the base for ongoing database maintenance and tuning; also needed program changes are identified.



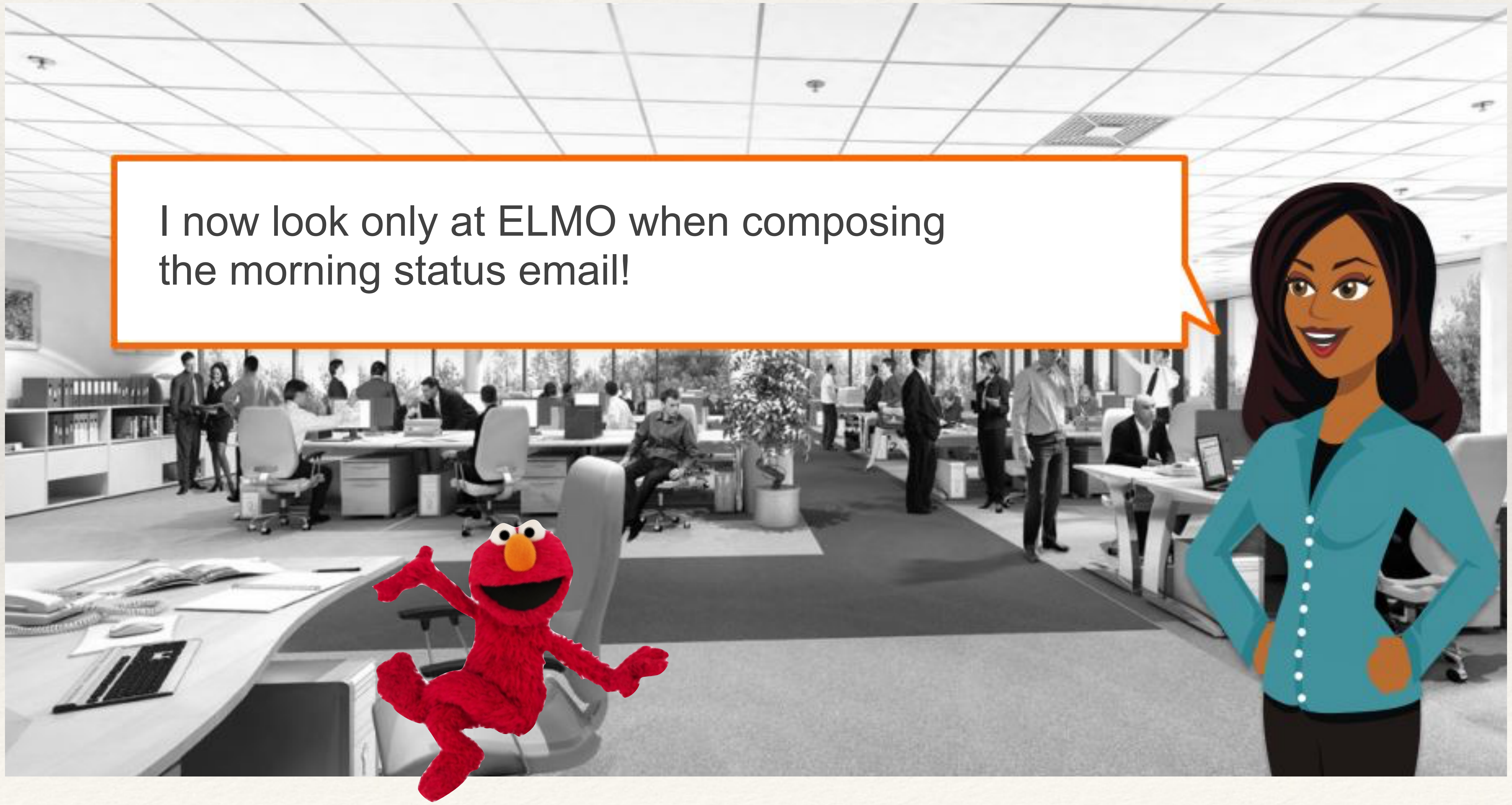
Tabular formats



SDP Throughput

	Name	Status	Last 30min	Total Amount	Last Order	Query Time
1	IDEAL MOBILE	BOOKED	2146	102822.7	2015-04-29 19:06:54.665819	2015-04-29 19:05:51.825941
2	IDEAL WEB	BOOKED	7418	564828	2015-04-29 19:06:55.231386	2015-04-29 19:05:51.825941
3	LOANS CT	BOOKED	4	14502.61	2015-04-29 19:05:05.259873	2015-04-29 19:05:51.825941
4	LOANS DISB	BOOKED	29	70155.87	2015-04-29 19:06:38.808085	2015-04-29 19:05:51.825941
5	SAVINGS	BOOKED	2054	6044935	2015-04-29 19:06:54.874424	2015-04-29 19:05:51.825941
6	SAVINGS	REJECTED	7	18986	2015-04-29 19:04:12.724191	2015-04-29 19:05:51.825941
7	SAVINGS	BOOKED	22	21965.63	2015-04-29 19:06:19.637107	2015-04-29 19:05:51.825941
8	SYNCHR SEPA-CT	REJECTED	36	7481.17	2015-04-29 19:03:32.177	2015-04-29 19:05:51.825941
9	SYNCHR SEPA-CT	BOOKED	10218	1754609	2015-04-29 19:06:54.669935	2015-04-29 19:05:51.825941
10	SYNCHR SEPA-CT	REJECTED	2	100	2015-04-29 18:50:46.098646	2015-04-29 19:05:51.825941
11	SYNCHR SEPA-CT	BOOKED	571	394053.2	2015-04-29 19:06:52.16929	2015-04-29 19:05:51.825941

There also is a tabular format for lists that are used for specific reports, like the “online” query that is used for the checklist and the 07.15 AM morning call.


I now look only at ELMO when composing the morning status email!

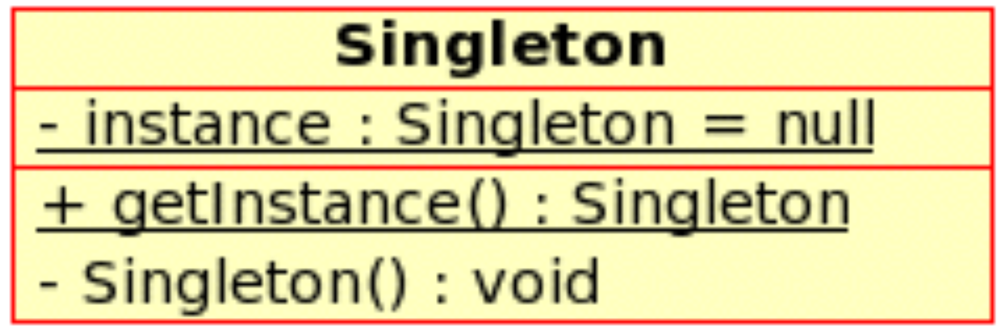
ELMO Architecture

Asynchronous

- Where older (3270-ISPF) ELMO fired DB2 queries for every user to draw the lines, ELMO-*ng* uses an asynchronous model
- The user looks via a web page served by an Apache Tomcat instance into a set of memory buffers
- These buffers are asynchronously updated by a set of monitor threads

Two Patterns

- ELMO uses, mainly, two software patterns
  - Singleton
  - Observer / Observable



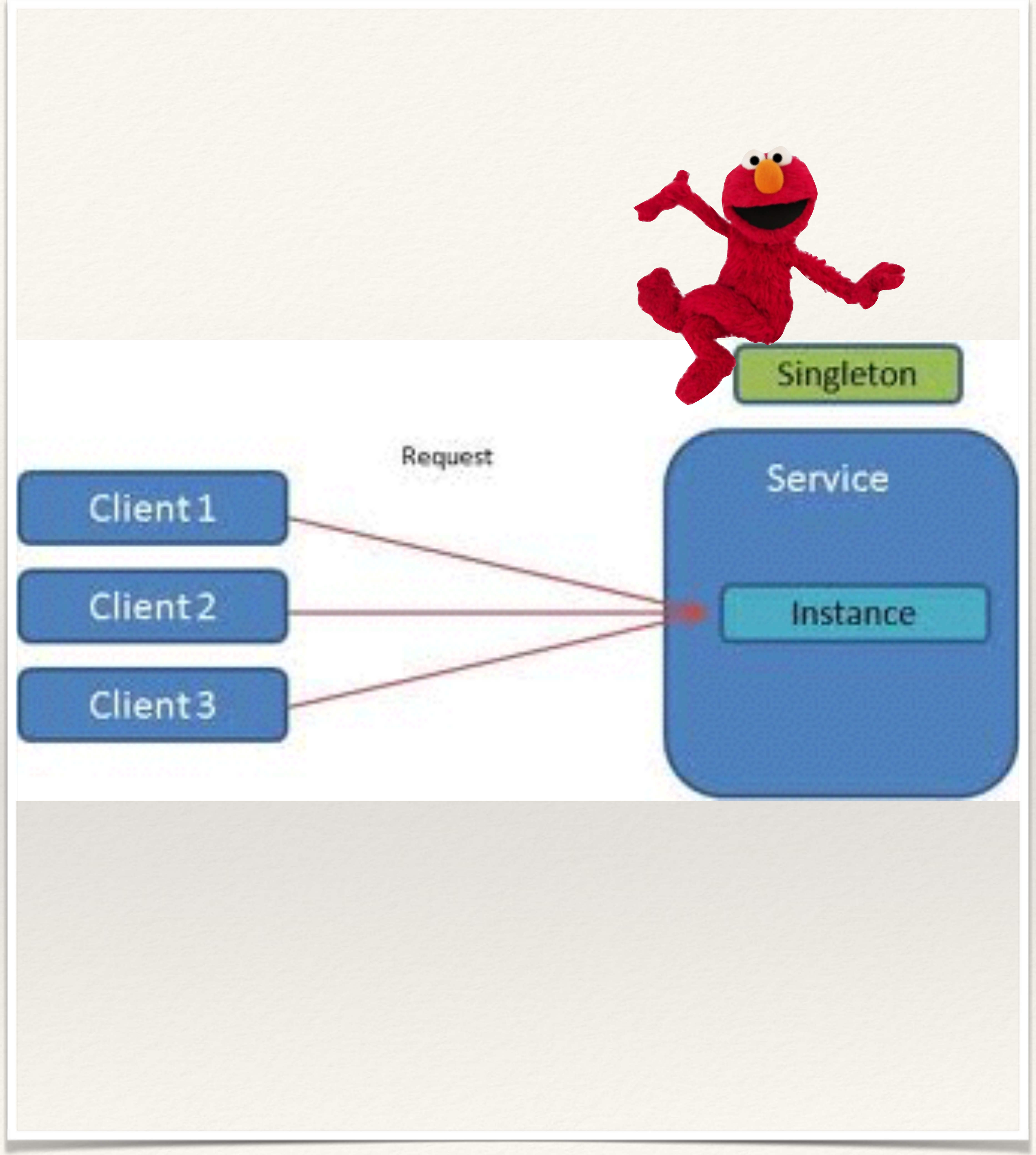
The Singleton pattern

Singleton

Of a singleton object, there is only one instance in the system.

There is a naming convention associated with the singleton pattern: every singleton class starts with **The**

In ELMO, we have the classes **The**Gatherer and **The**DataAccess.

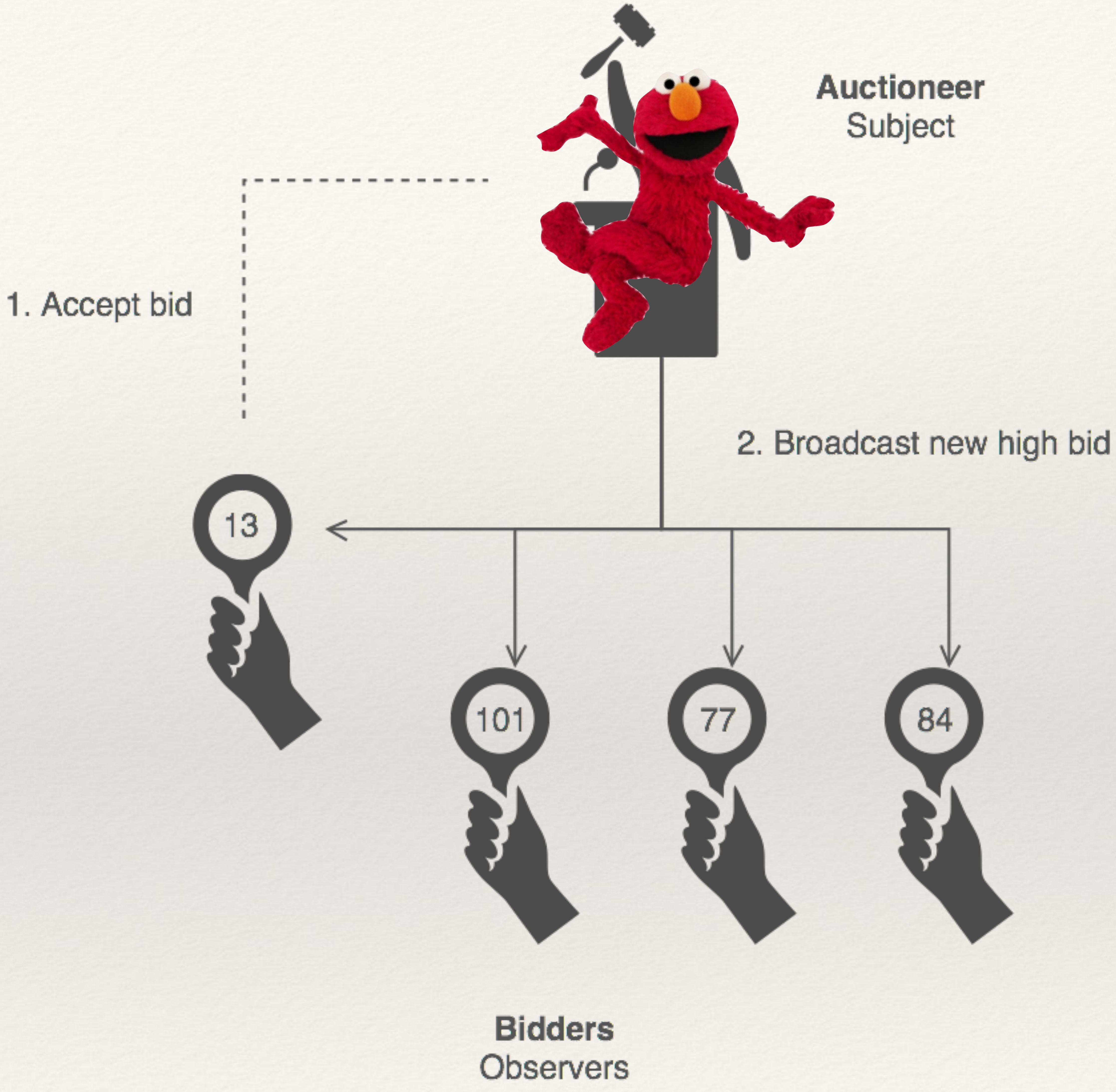


The Observer / Observable pattern

Observer / Observable

The observer pattern is a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods. It is mainly used to implement distributed event handling systems.

In ELMO, the Subject is TheGatherer, which inherits from **Observer** and its dependents are threads that implement the **Monitor** and **Observable** interfaces.



TheGatherer 1

```
package com.ing.sdp.elmo
import java.util.Observable
/**
 * Class TheGatherer implements...
 * <BR>
 * Created on: di, 27, jan 2015 14:13:56 +0100
 */
class TheGatherer implements Observer

  properties static
  instance     = TheGatherer

  properties static public
  logger_      = Logger.getLogger(TheGatherer.class.getName())
  idealDelta   = 99999
  mingDelta    = 99999
  mobileDelta  = 99999
  profDelta    = 99999

  statusButton = String '<a href="#" class="medium success button">All Nominal</a><br/>'

  -- data from contention monitor
  contentionData  = ConcurrentHashMap()

  -- data for velocitymonitor
  velocityData    = TreeMap()

  -- data for filetransfers
  confirm_result  = ArrayList()
  in_result       = ArrayList()
  out_result      = ArrayList()

  -- data from throughput monitor
  throughputArray = ArrayList()

  -- data for Job abend monitor
  sdpJobAbendData = ArrayList()
```

instance field, to implement singleton

Memory maps for display data

TheGatherer 2

```
method getInstance() returns TheGatherer static protect
   if TheGatherer.instance <> null then
     do
       logger_.info( "TheGatherer: returning singleton instance")
       return TheGatherer.instance
     end
   else
     do
       TheGatherer.instance = TheGatherer()
 return TheGatherer.instance
     end -- do

 /**
  * private constructor enforces singleton
  */
 method TheGatherer() private signals ClassNotFoundException
   logger_.info( "TheGatherer: start")
   t1 = IdealTransactionStatusMonitor(10000)
   t1.addObserver(this)
   Thread(t1).start()
   logger_.info( "TheGatherer: started thread IdealTransactionStatusMonitor")

   t2 = ProfileTransactionStatusMonitor(10000)
   t2.addObserver(this)
   Thread(t2).start()
   logger_.info( "TheGatherer: started thread ProfileTransactionStatusMonitor")

   t3 = ThroughputMonitor(10000)
   t3.addObserver(this)
   Thread(t3).start()
   logger_.info( "TheGatherer: started
```

Singleton (can be fancier but this fits the bill)

Start monitors in threads to observe

TheGatherer 3

```
method update(o=Observable,obj=Object) protect
  cl = o.getClass().getName()
  select
    when cl = 'com.ing.sdp.elmo.IdealTransactionStatusMonitor' then idealDelta = Rexx obj
    when cl = 'com.ing.sdp.elmo.ProfileTransactionStatusMonitor' then profDelta = Rexx obj
    when cl = 'com.ing.sdp.elmo.VelocityMonitor' then do
      v = Velocity obj
      velocityData.put(Rexx(v.getNow().toString()),  v)
    end
    when cl = 'com.ing.sdp.elmo.ThroughputMonitor' then do
      throughputArray = ArrayList obj
    end
    when cl = 'com.ing.sdp.elmo.ContentionMonitor' then do
      contentionData  = ConcurrentHashMap obj
    end
    when cl = 'com.ing.sdp.elmo.SDPJobMonitor' then do
      sdpJobAbendData = ArrayList obj
    end
    when cl = 'com.ing.sdp.elmo.FileTransferMonitor' then do
      select
        when obj.getClass.getName = 'com.ing.sdp.elmo.FileConfArrayList' then confirm_result = ArrayList obj
        when obj.getClass.getName = 'com.ing.sdp.elmo.FilesinArrayList' then in_result = ArrayList obj
        when obj.getClass.getName = 'com.ing.sdp.elmo.FilesoutArrayList' then out_result = ArrayList obj
        otherwise
          say 'filemonitor sent an unknown update object'
      end
    end
    when cl = 'com.ing.sdp.elmo.IbPostIDThroughputMonitor' then do
      t = ThroughPut obj
      sectprocessed     = t.processed
      sectnotprocessed  = t.notprocessed
      sectdiff30m       = t.sectdiff30m
      sectdiff1h        = t.sectdiff1h
      sectdiff2h        = t.sectdiff2h
      sectdiff2h2       = t.sectdiff2h2
    end
    otherwise
      say 'could not find which observable to update'
  end
```

When a Monitor sends an update, it is in the form of an Observable

TheDataAccess 1

```
package com.ing.sdp.elmo
import java.sql.
import java.util.

/**
 * Class TheDataAccess is a singleton that takes care of all queries to the payments production environment.
 */

class TheDataAccess uses RexxDate

  properties private static
  jdbcCon   = Connection        -- to dpg1
  instance  = TheDataAccess null

  method TheDataAccess() private protect

  method getInstance() returns TheDataAccess static protect signals ClassNotfoundException
    if instance <> null then return instance
    instance = TheDataAccess()

    -- get encrypted credentials
    c      = Credentials('elmo.properties')
    userid = c.getUserid()
    pswd   = c.getPassword()

    Class.forName("com.ibm.db2.jcc.DB2Driver")
    url=‘jdbc:db2://xxxx.xx.xxxx.intranet:XXX/NLXXX_XXX1’

    do
      -- make the connection
      jdbcCon  = Connection DriverManager.getConnection(url, userid, pswd)
    catch e = SQLException
      printException(e)
    end -- do

    return instance
```

TheDataAccess 2

```
method getcurrenttimestamp() returns java.sql.Timestamp
  timer = TimeIt()
   ts = java.sql.Timestamp null
   do
     sqlstmt = “ SELECT                " -
               " CURRENT TIMESTAMP     " -
               " FROM SYSIBM.SYSDUMMY1 " -
               " WITH UR               "
     stmt = Statement this.jdbcCon.createStatement()
     rs = ResultSet stmt.executeQuery(sqlstmt)

     -- get the data rows
     loop while rs.next()
       ts = rs.getTimestamp(1)
     end -- loop while rs
     rs.close()
     stmt.close()
     timer.sayDiff('method getcurrenttimestamp took:')
     return ts
   catch e = SQLException
     printException(e)
     return ts
   end
```

This is the one query I can show you

Monitor (Superclass of all monitors, provides database (TheDataAccess) connection)

```
package com.ing.sdp.elmo
import java.util.Observable
/**
 * Class Monitor implements...
 * <BR>
 * Created on: za, 14, mrt 2015 15:11:35 +0100
 */
class Monitor extends Observable

  properties public
  logger_ = Logger.getLogger(Monitor.class.getName())
  sleeptime

  properties static
  da      = TheDataAccess null

  /**
   * Default constructor
   */
  method Monitor()
    this.da  = TheDataAccess.getInstance()
```

a Monitor instance

```
package com.ing.sdp.elmo

class ThroughputMonitor implements Runnable extends Monitor

  method ThroughputMonitor(s) signals ClassNotFoundException
    this.sleeptime = s

  method run()
    do
      Thread.currentThread().sleep(this.sleeptime) -- sleep for sleeptime seconds
      loop forever
        setChanged()
        notifyObservers(this.da.online())
        Thread.currentThread().sleep(this.sleeptime) -- sleep for sleeptime seconds
      end
    catch InterruptedException
      parse source s
      say "thread interrupted:"  s
    end
```

A Monitor sleeps, does a database call and notifies its observers

Some more TheDataAccess, then …

```
method online() returns ArrayList protect
   timer = TimeIt()
   logger_.info( "TheDataAccess: start method online")
   a=ArrayList()
   do
     crstmt = "DECLARE GLOBAL TEMPORARY TABLE PAYMENTTYPES (      " -
              "NAME VARCHAR(40)                                   " -
              ") on commit preserve rows                              " -
              ";"

     stmt = Statement this.jdbcCon.createStatement()
     stmt.execute(crstmt)
     stmt.close()
     this.jdbcCon.commit()

     instmt = "INSERT INTO session.PAYMENTTYPES                       " -
              "VALUES ('IDEAL WEB')                                   " -
              ";                                                      "
```

(…)

```
  logger_.info( 'TheDataAccess: method online returned' a.size() 'lines to ThroughputMonitor.')
  timer.sayDiff('method online took:')
  return a
catch e = SQLException
  printException(e)
  say "online failed"
  return a
end
```

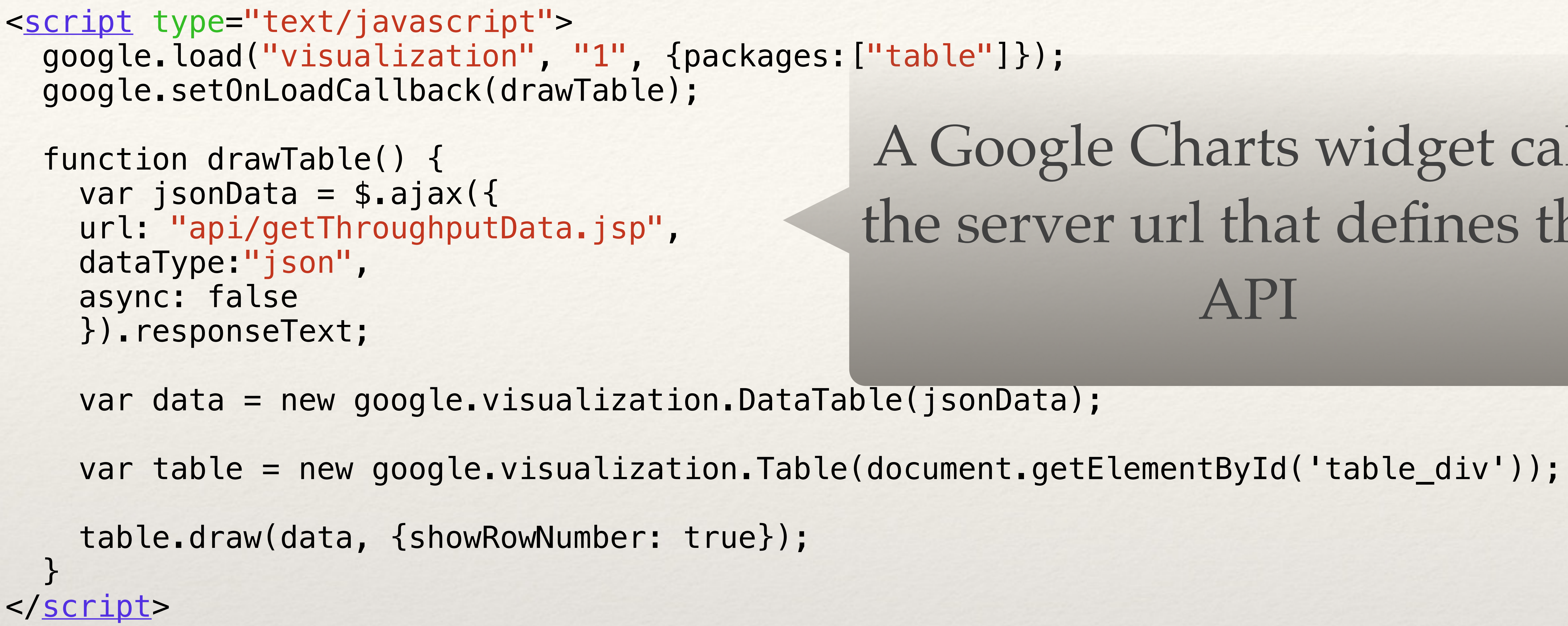
well, I need to show a little bit more here from TheDataccess

It returns an ArrayList, which is wrapped into the Observable, which updates the memory maps in TheGatherer

Backend to Frontend

- So you saw the backend that starts the monitoring processes that get to the payment data streams
- These fill the memory maps the front ends look at (there are as many Web Browsers open as you want, these do not add overhead)
- The route here is browser page (.jsp), JSON API call, Viewer, TheGatherer, and back again to display the widget

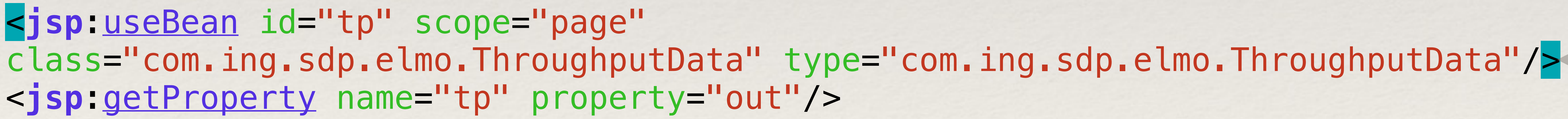
Browser Page



```
<script type="text/javascript">
  google.load("visualization", "1", {packages:["table"]});
  google.setOnLoadCallback(drawTable);

  function drawTable() {
    var jsonData = $.ajax({
    url: "api/getThroughputData.jsp",
    dataType:"json",
    async: false
    }).responseText;

    var data = new google.visualization.DataTable(jsonData);

    var table = new google.visualization.Table(document.getElementById('table_div'));

    table.draw(data, {showRowNumber: true});
  }
</script>
```

A Google Charts widget calls the server url that defines that API

API Definition



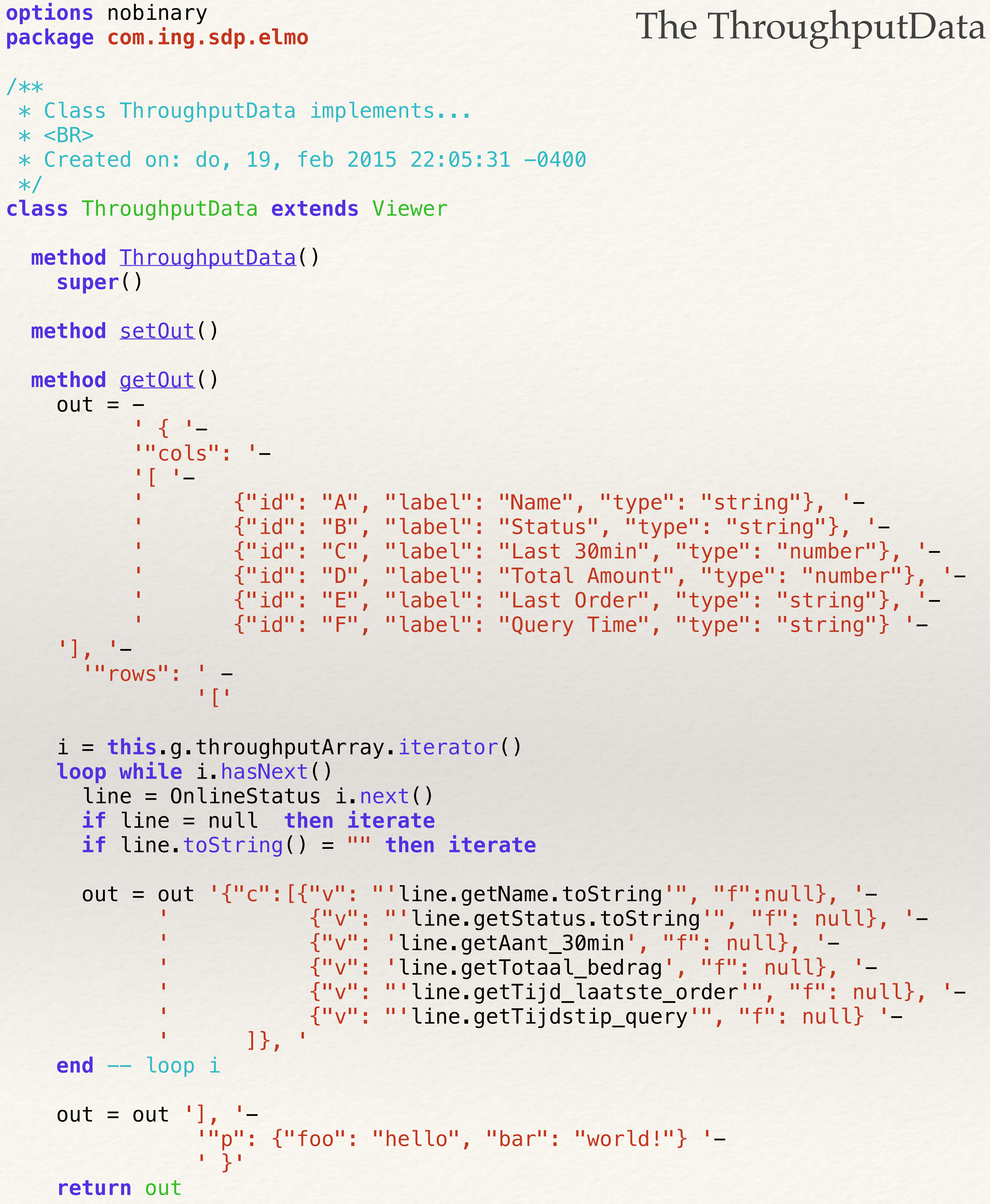
```
<jsp:useBean id="tp" scope="page"
class="com.ing.sdp.elmo.ThroughputData" type="com.ing.sdp.elmo.ThroughputData"/>
<jsp:getProperty name="tp" property="out"/>
```

The file contents of api/getThroughputData.jsp

Viewer base class provides TheGatherer singleton link to all Viewers

```
package com.ing.sdp.elmo

/**
 * Class Viewer implements the common superclass for all viewers
 * <BR>
 * Created on: vr, 13, mrt 2015 16:36:45 +0100
 */
class Viewer

  properties public
  g = TheGatherer

  /**
   * Default constructor
   */
  method Viewer() signals ClassNotfoundException
    this.g = TheGatherer.getInstance()
    return
```

The ThroughputData class is a Viewer



```
options nobinary
package com.ing.sdp.elmo

/**
 * Class ThroughputData implements...
 * <BR>
 * Created on: do, 19, feb 2015 22:05:31 -0400
 */
class ThroughputData extends Viewer

  method ThroughputData()
    super()

  method setOut()

  method getOut()
    out = -
          ' { '-
          '"cols": '-
          '[ '-
          '       {"id": "A", "label": "Name", "type": "string"}, '-
          '       {"id": "B", "label": "Status", "type": "string"}, '-
          '       {"id": "C", "label": "Last 30min", "type": "number"}, '-
          '       {"id": "D", "label": "Total Amount", "type": "number"}, '-
          '       {"id": "E", "label": "Last Order", "type": "string"}, '-
          '       {"id": "F", "label": "Query Time", "type": "string"} '-
    '], '-
      '"rows": ' -
               '['

    i = this.g.throughputArray.iterator()
    loop while i.hasNext()
      line = OnlineStatus i.next()
      if line = null  then iterate
      if line.toString() = "" then iterate

      out = out '{"c":[{"v": "'line.getName.toString'", "f":null}, '-
            '           {"v": "'line.getStatus.toString'", "f": null}, '-
            '           {"v": 'line.getAant_30min', "f": null}, '-
            '           {"v": 'line.getTotaal_bedrag', "f": null}, '-
            '           {"v": "'line.getTijd_laatste_order'", "f": null}, '-
            '           {"v": "'line.getTijdstip_query'", "f": null} '-
            '      ]}, '
    end -- loop i

    out = out '], '-
               '"p": {"foo": "hello", "bar": "world!"} '-
               ' }'
    return out
```

This does “JSON by hand”. It is no party but you have to get it right only once.

It picks the live chart data out of the throughputArray structure of TheGatherer

Useful resources

- Google charts API demo page at: https://developers.google.com/chart/interactive/docs/gallery
- Browser development tools - debuggers. Safari, Chrome, Firefox - all have their strong points and I really needed them all at one point to get the all the JSON of the different live chart type widgets going.
- Of course Internet Explorer was the most troublesome, did not want to update live data at all without some really obscure tweaks (thanks, Joris and Leo!) - so if you really want IE, you need IE chops.

Useful resources

- Git repository for team cooperation is invaluable. We cooperated very geographically dispersed (Amsterdam, Rotterdam, Arnhem, Aruba) and with very few merge conflicts
- NetREXX: we developed on Windows, Linux, z/OS, with Notepad, UltraEdit, Emacs, Eclipse, VI, ISPF/PDF: Don’t worry, be happy! So use the tools that you like most.
- (None of the others in this 5-person team ever used NetREXX or Git; all are fans now; ELMO was no full-time project, everyone had other - primary - responsibilities).

What happened to ELMO in 2016 (after me leaving)

- ELMO is alive and well, and lives on a production server where he is well looked after. He enjoys his connection to DB2 z/OS production, and in turn looks after the large payment and booking systems, which themselves are happier now also.
- ELMO won a software innovation price ("**ING Team Craftsmanship Award**") and my former co-workers earned a trip to Silicon Valley!
- Google charts was later built-out of ELMO and was replaced by an open source live charts library due to privacy concerns; only some Javascript calls and their JSON needed change.

Thanks for your attention.
Questions?

René Vincent Jansen, rvjansen@xs4all.nl