Interesting things sometimes happen unexpectedly. In this case by combining several unrelated posts :

- rapid prototyping
- webworkers and flow based programming
- compiled procedures

In reverse order : the last post described a short-cut to produce webworkers on the fly. The middle post describes an alternative architecture where a network is constructed out of webworkers and messagechannels, and the first post covers a possible take on rapid prototyping.

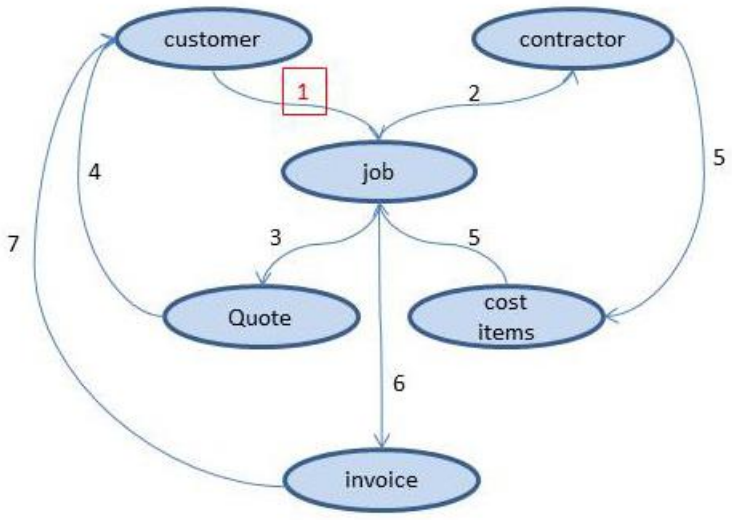
Of special interest here is the last step in rapid prototyping : defining business processes (step3).

The subject of this post is that a) business processes can be graphed as well. Moreover, b) these business process graphs can be mapped as a network of workers, which c) can be of the on-the-fly internal type. That makes for a crazy architecture, but with some advantages (see at the end).

To see how this works out, take a new example of a fictional (electrical or plumbing) services business. This business employs contractors to do jobs. Jobs come in from different sources, one of them being customers who ring up or fill out an online form with their problems or service requests.

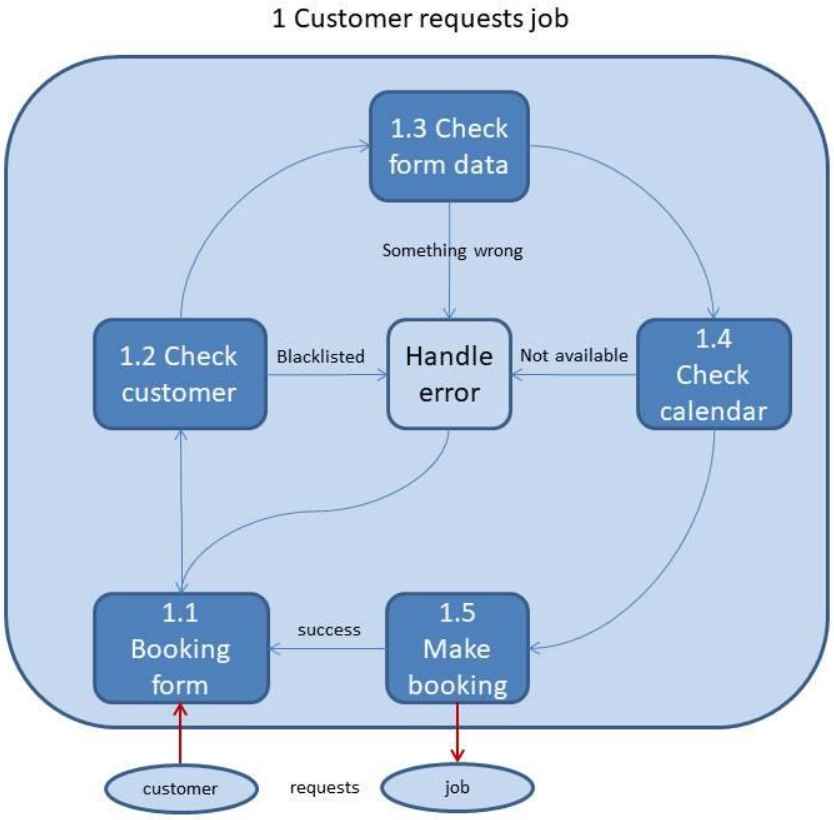
Graph-sentences and concept map (see rapid prototyping – step1) :

1. customers request jobs
2. jobs are assigned to contractors
3. jobs may be quoted
4. customer accepts/rejects quoted job
5. contractor spends material / hours on job
6. job gets invoiced
7. invoice gets paid by customer



Zooming in on business process -1 : “customer requests job” (rapid prototyping – step3)

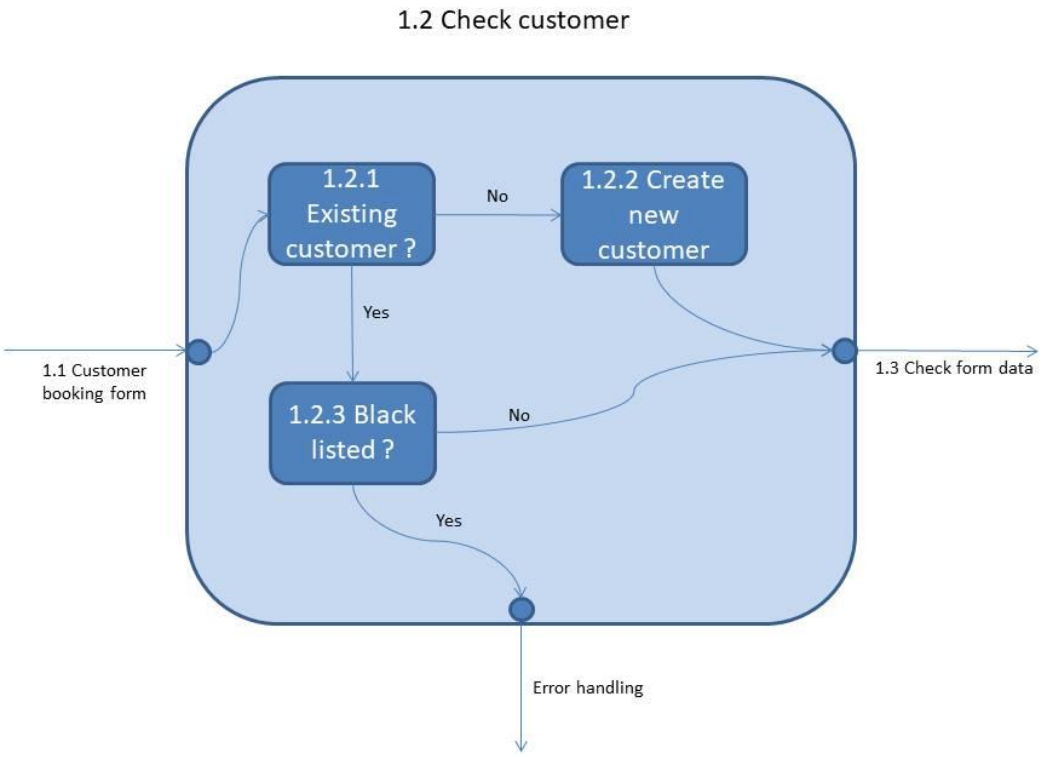
1. a customer has a problem or service request and starts an online process to book a visit
2. The system checks if the customer is an existing customer
   2a. If he or she exists, verifies that the customer has not been blacklisted
   2b. otherwise creates a new customer
3. The system checks the validity of the rest of the booking data, and
4. checks the availability of contractor capacity at the desired date.



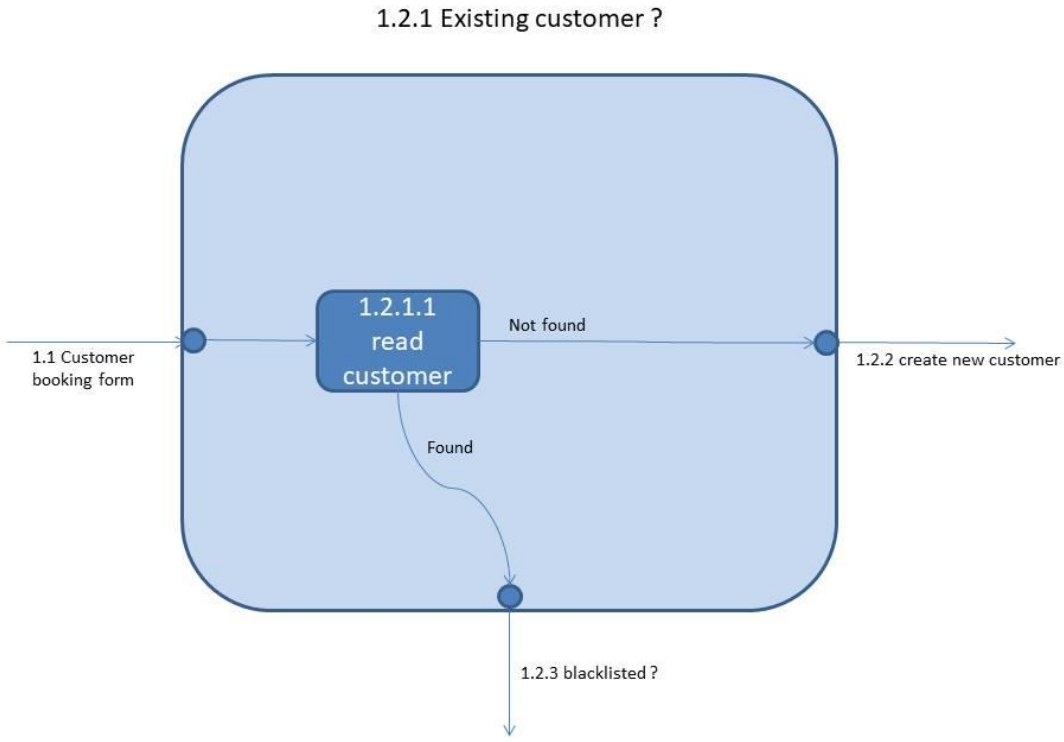
This lists sub-processes 1.1 to 1.5

Drilling down into these sub-processes gives more detail :

Zooming in on sub-process 1.2 (level 2)



And zooming in on sub-sub-process 1.2.1 (level 3)



Deconstructing a business process this way gives a list of elementary components in a network structure.

And this component network can be mapped 1:1 to simple webworkers, who communicate with each other through messageChannels.

Of course only the lowest level of nodes have to be implemented as workers. In this example worker-1.2.1.1 (but not its parents 1.2.1 or 1.2).

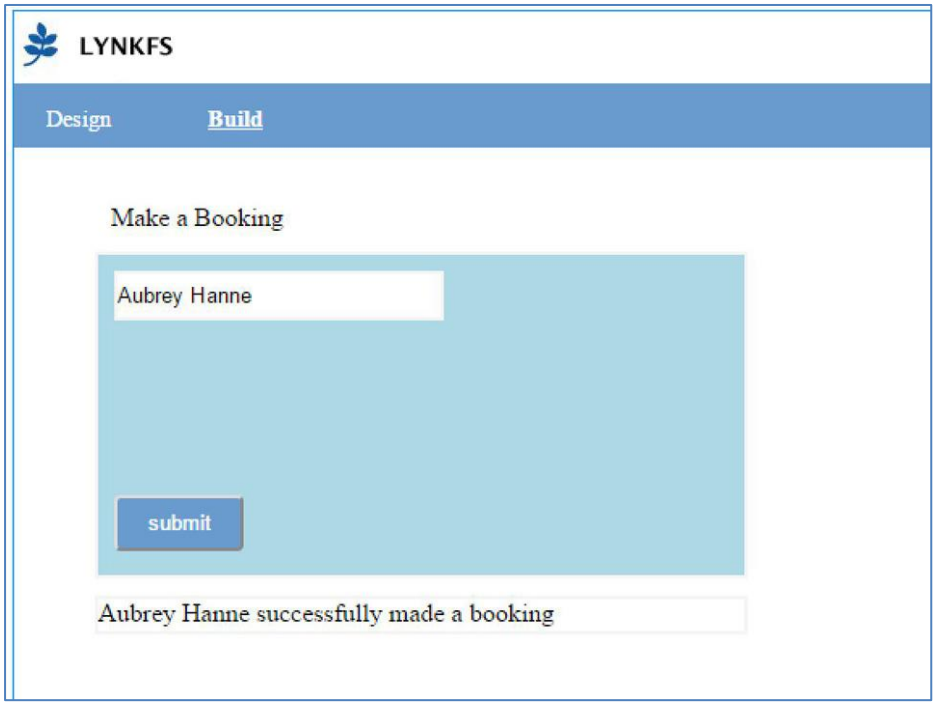
As business processes are deconstructed to very elementary levels, the worker-code for these workers is pretty simple too :

//worker-1.2.1.1 pseudo-code :
Read customer file with key of customer-id / name
If found, activate and push the value of field 'blacklisted' to worker-1.2.3
If not found activate worker-1.2.2

A somewhat crazy way of doing things.
The advantages however are that

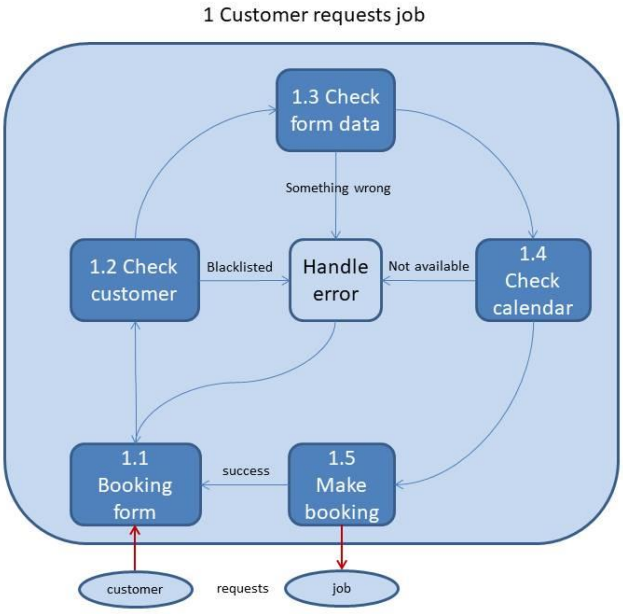
- it gives a straight pathway from design to build
- rapid prototyping converges into rapid application development
- the webworkers, and how they are wired up (messageChannels), are external to the main program, so it is easy to make changes without having to recompile

See this in action : www.lynkfs.com/Experiments/rp-rad-ww/index.html



The above screenshot of a pretend booking form looks deceptively simple. It is just a form with a name-field and a button.

However behind the scenes the webworker network as per below is activated.



For demo purposes it is wired up in such a way that 3 out of 4 times the network ends up in sub-process 1.5 without errors, and the message bar displays ‘succesfully made a booking’. Otherwise processing randomly stops in 1.2, 1.3 or 1.4, simulating the specific error in that sub-process.