Declare, verify and execute microservices-based process flows with Baker

Scale By the Bay 2017, San Francisco

Nikola Kasev | ING Bank


KENAMJU
Rudolph's
BAKERY

Symptoms of a Failing Application Architecture

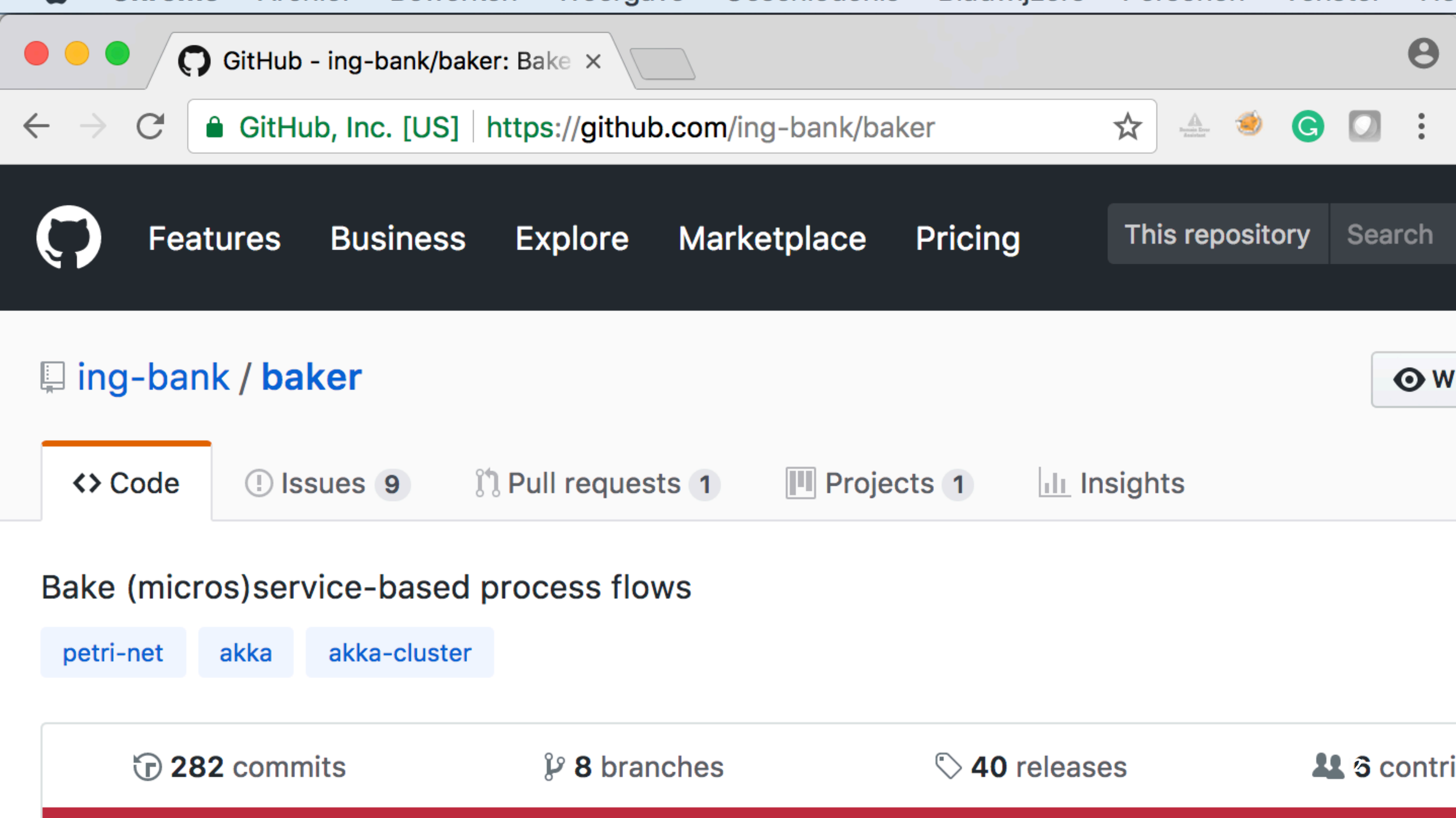

Afraid to change the application code


Functionality breaks unexpectedly


Slow time to market

How to turn this around?


GUAYABA
LICHI


GitHub - ing-bank/baker: Bake
GitHub, Inc. [US]
https://github.com/ing-bank/baker
Features
Business
Explore
Marketplace
Pricing
This repository
Search
ing-bank / baker
Code
Issues 9
Pull requests 1
Projects 1
Insights
Bake (micros)service-based process flows
petri-net
akka
akka-cluster
282 commits
8 branches
40 releases

Simplify

Domain Specific Language for orchestration flows

Declarative

Easy to change

Reuse

Recipes

Interactions

Ingredients

Events

Communicate

Visualize your code

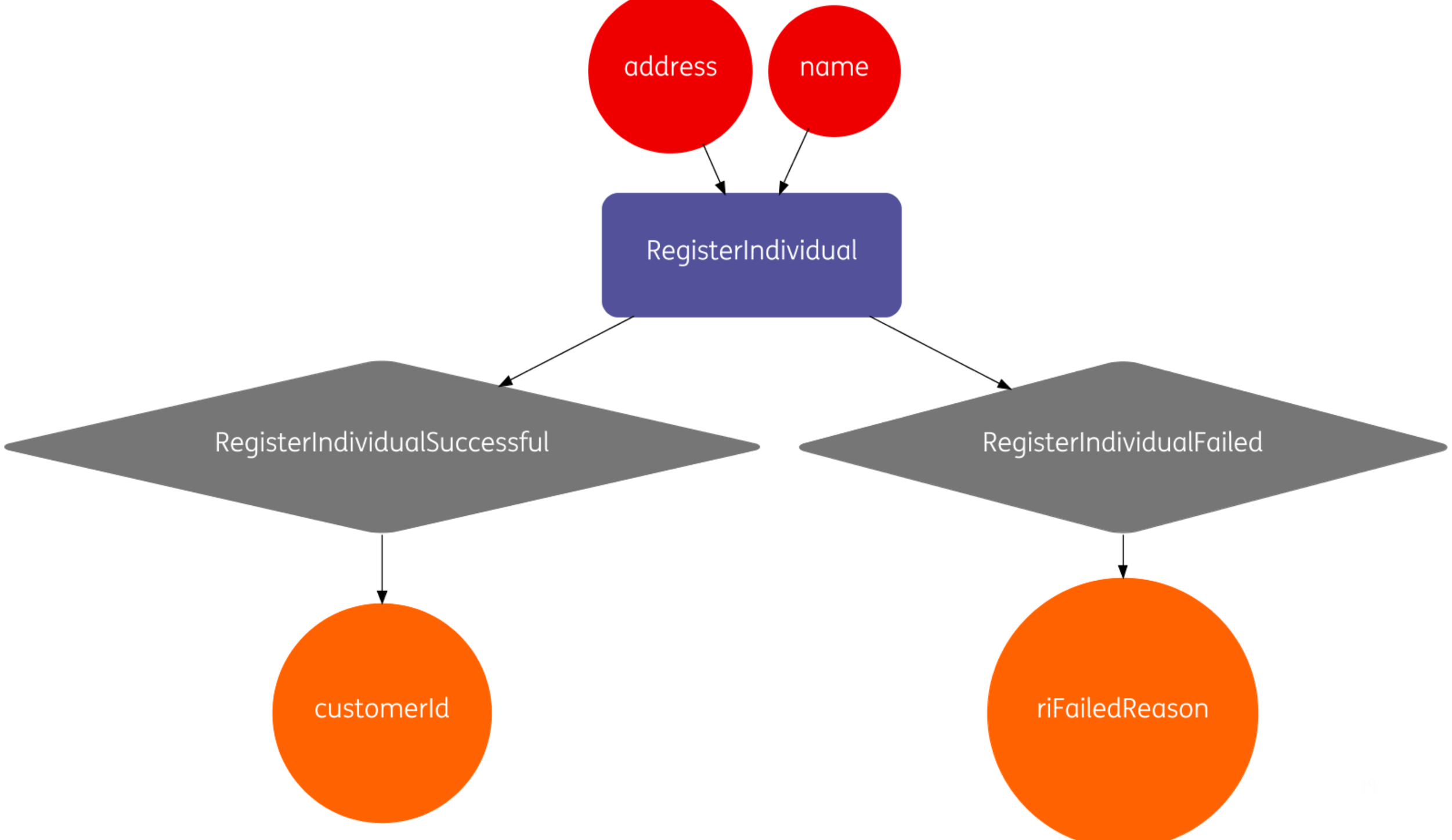
Non-IT understand as well

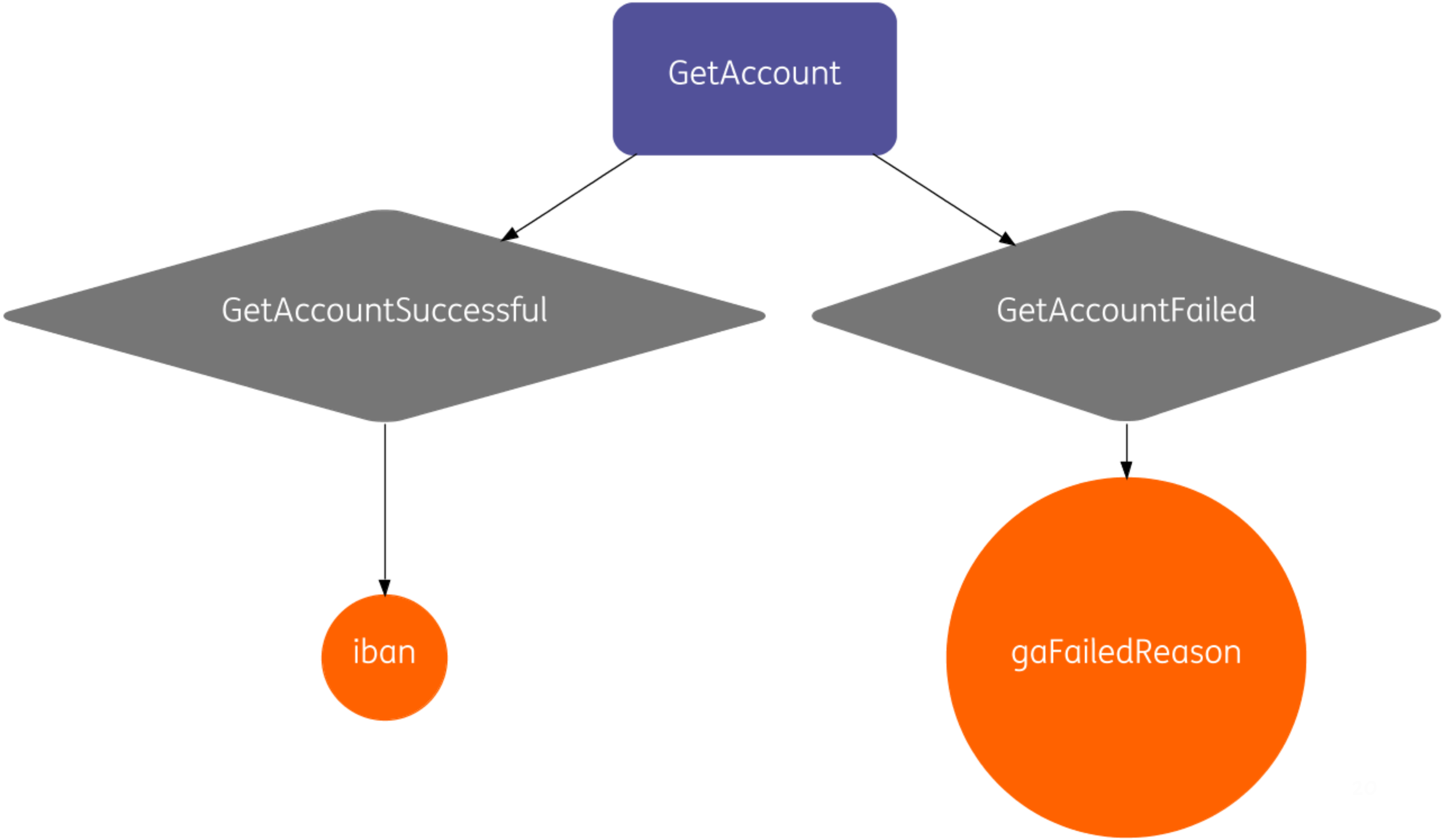
Reason About Comfortably

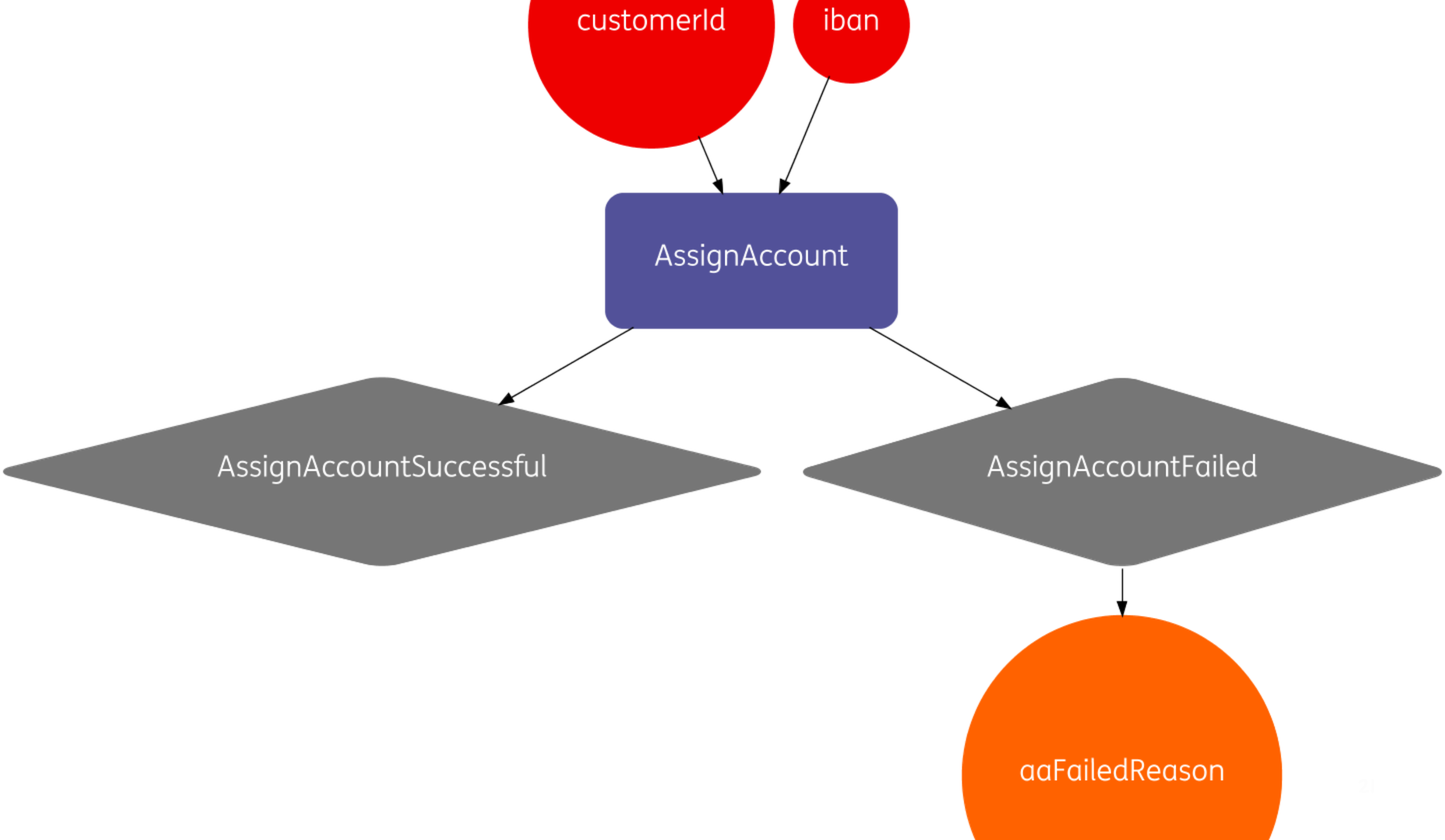
Design-time

```
public interface RegisterIndividual extends Interaction {
    @FiresEvent(oneOf = {RegisterIndividualSuccessful.class,
            RegisterIndividualFailed.class})
    RegisterIndividualOutcome apply(
            @ProcessId String processId,
            @RequiresIngredient("name") String name,
            @RequiresIngredient("address") String address
    );
}
```

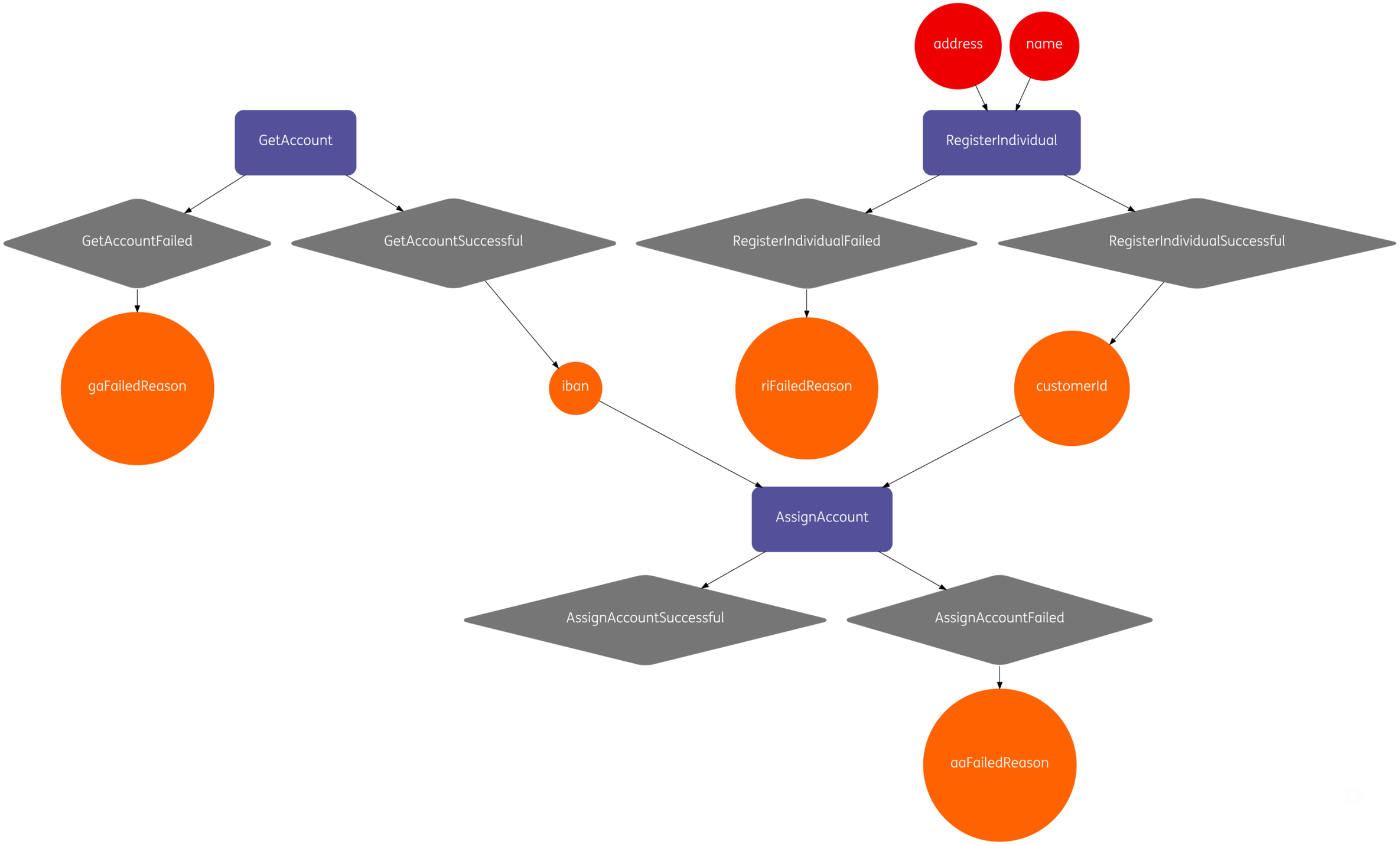
```
public interface RegisterIndividual extends Interaction {
    @FiresEvent(oneOf = {RegisterIndividualSuccessful.class,
            RegisterIndividualFailed.class})
    RegisterIndividualOutcome apply(
            @ProcessId String processId,
            @RequiresIngredient("name") String name,
            @RequiresIngredient("address") String address
    );
}
```


address
name
RegisterIndividual
RegisterIndividualSuccessful
RegisterIndividualFailed
customerId
riFailedReason

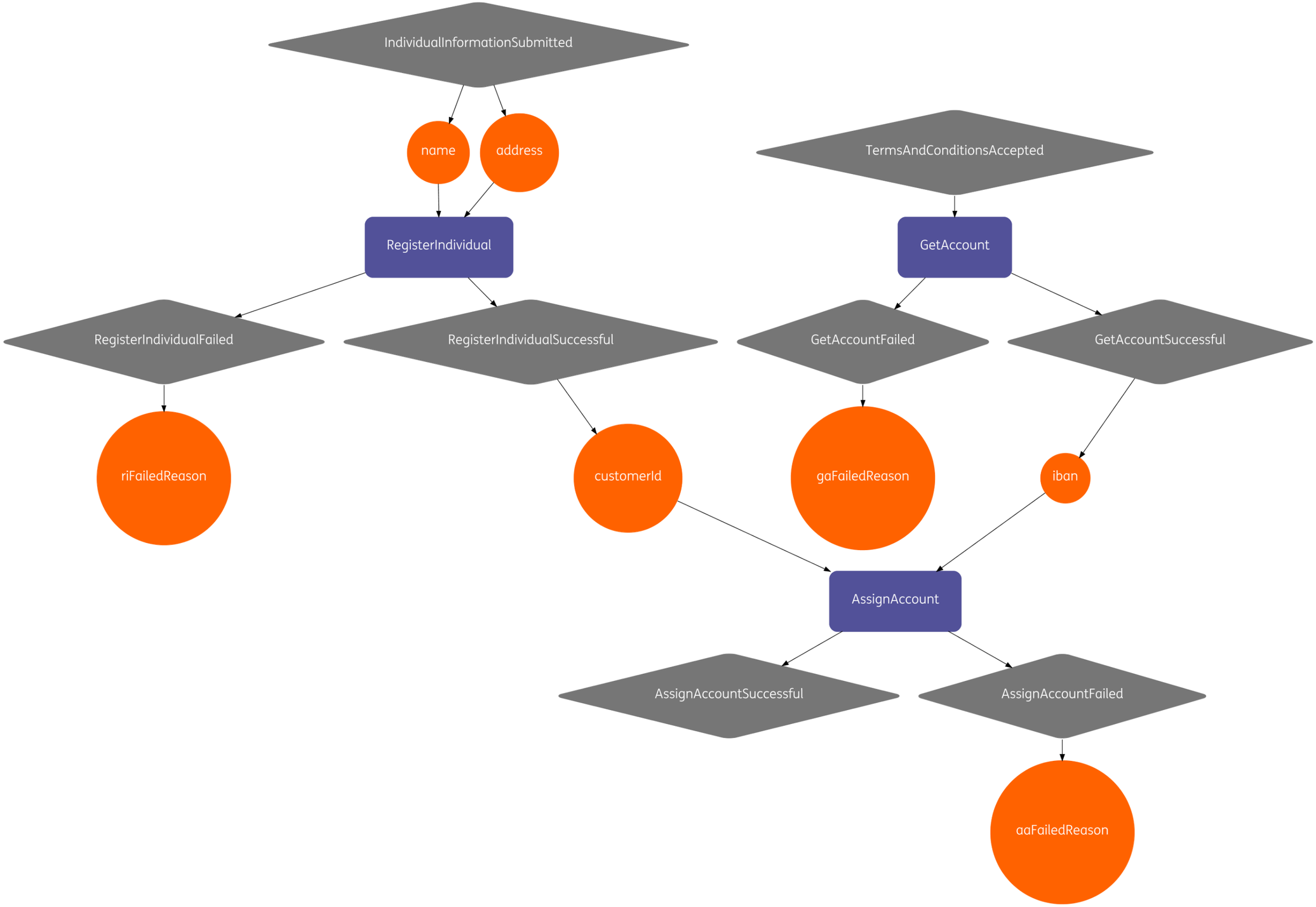

GetAccount
GetAccountSuccessful
GetAccountFailed
iban
gaFailedReason


customerId
iban
AssignAccount
AssignAccountSuccessful
AssignAccountFailed
aaFailedReason

```
public Recipe get(){
    return new Recipe("Demo").
            withInteractions(
                    of(AssignAccount.class),
                    of(GetAccount.class),
                    of(RegisterIndividual.class));
}
```


address
name
GetAccount
RegisterIndividual
GetAccountFailed
GetAccountSuccessful
RegisterIndividualFailed
RegisterIndividualSuccessful
gaFailedReason
iban
riFailedReason
customerId
AssignAccount
AssignAccountSuccessful
AssignAccountFailed
aaFailedReason

```
return new Recipe("Demo").
        withInteractions(
                of(AssignAccount.class),
                of(GetAccount.class).
                        withRequiredEvent(TermsAndConditionsAccepted.class),
                of(RegisterIndividual.class)).
        withSensoryEvents(
                TermsAndConditionsAccepted.class,
                IndividualInformationSubmitted.class);
}
```

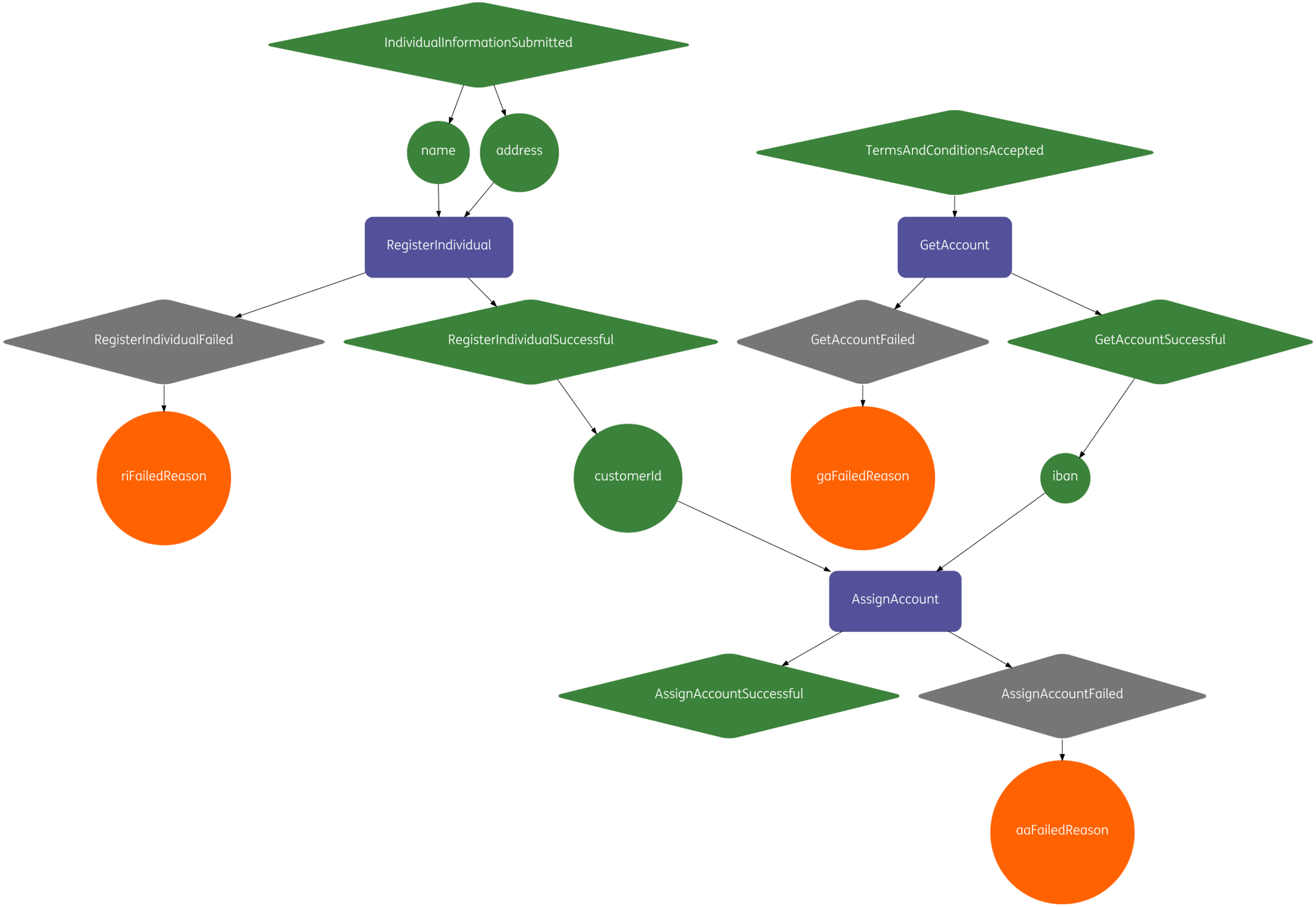

IndividualInformationSubmitted
name
address
RegisterIndividual
RegisterIndividualFailed
RegisterIndividualSuccessful
riFailedReason
customerId
TermsAndConditionsAccepted
GetAccount
GetAccountFailed
GetAccountSuccessful
gaFailedReason
iban
AssignAccount
AssignAccountSuccessful
AssignAccountFailed
aaFailedReason

Run-time

```
//for each process instance, bake the recipe
baker.bake(processId);
//notify Baker when events occur
baker.processEvent(processId, new SensoryEvents.IndividualInformationSubmitted(name, address));
baker.processEvent(processId, new SensoryEvents.TermsAndConditionsAccepted());

//retrieve ingredients stored in the accumulated state
assert(baker.getIngredients(processId).get("customerId").equals(customerId));
assert(baker.getIngredients(processId).get("iban").equals(iban));

//retrieve all events that have occurred
Set<String> occurredEvents = new HashSet<>(
        baker.getEvents(processId).getEventNameList()
);
```


IndividualInformationSubmitted
name
address
RegisterIndividual
RegisterIndividualFailed
RegisterIndividualSuccessful
riFailedReason
customerId
TermsAndConditionsAccepted
GetAccount
GetAccountFailed
GetAccountSuccessful
gaFailedReason
iban
AssignAccount
AssignAccountSuccessful
AssignAccountFailed
aaFailedReason

Under the Hood

Why Scala?

- Best fit for developing DSLs[1] on the JVM
- Compile-time recipe validation
- Type safety

[1] https://martinfowler.com/books/dsl.html

Why Akka?

- Event Sourcing[2] (events can be replayed)
- Persistent actors (with Cassandra)
- Distributed actors across machines (with cluster sharding)

[2] http://martinfowler.com/eaaDev/EventSourcing.html

Best Practices

Short-lived vs. long-running flows

State is taken care of:

- Cassandra for persistent storage
- Ingredients encrypted by default
- State recovered automatically

Run Baker inside of your API

"Smart endpoints and dumb pipes"

When failure occurs:

- Baker retries technical failures with exponential backoff
- Works well with **idempotent** services
- Deal with functional failure in your recipe

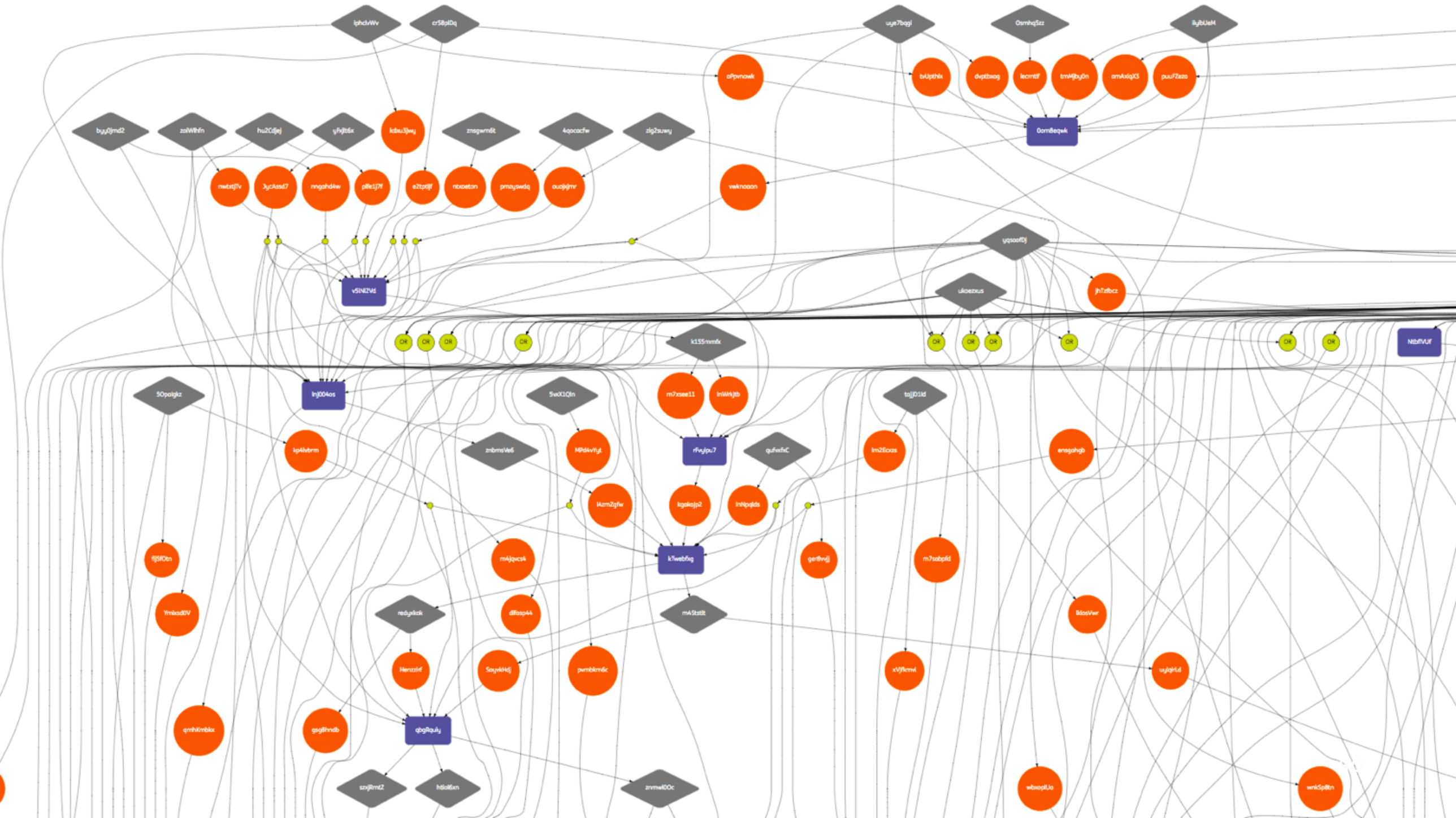
Baker Capability Matrix:

- Investigate not one, not two, but **all business processes** in your company
- Where do you see re-use?
- Map using MoSCoW[4] to give importance (M = 10, S = 5, C = 2, W = 1)

[4] https://en.wikipedia.org/wiki/MoSCoW_method

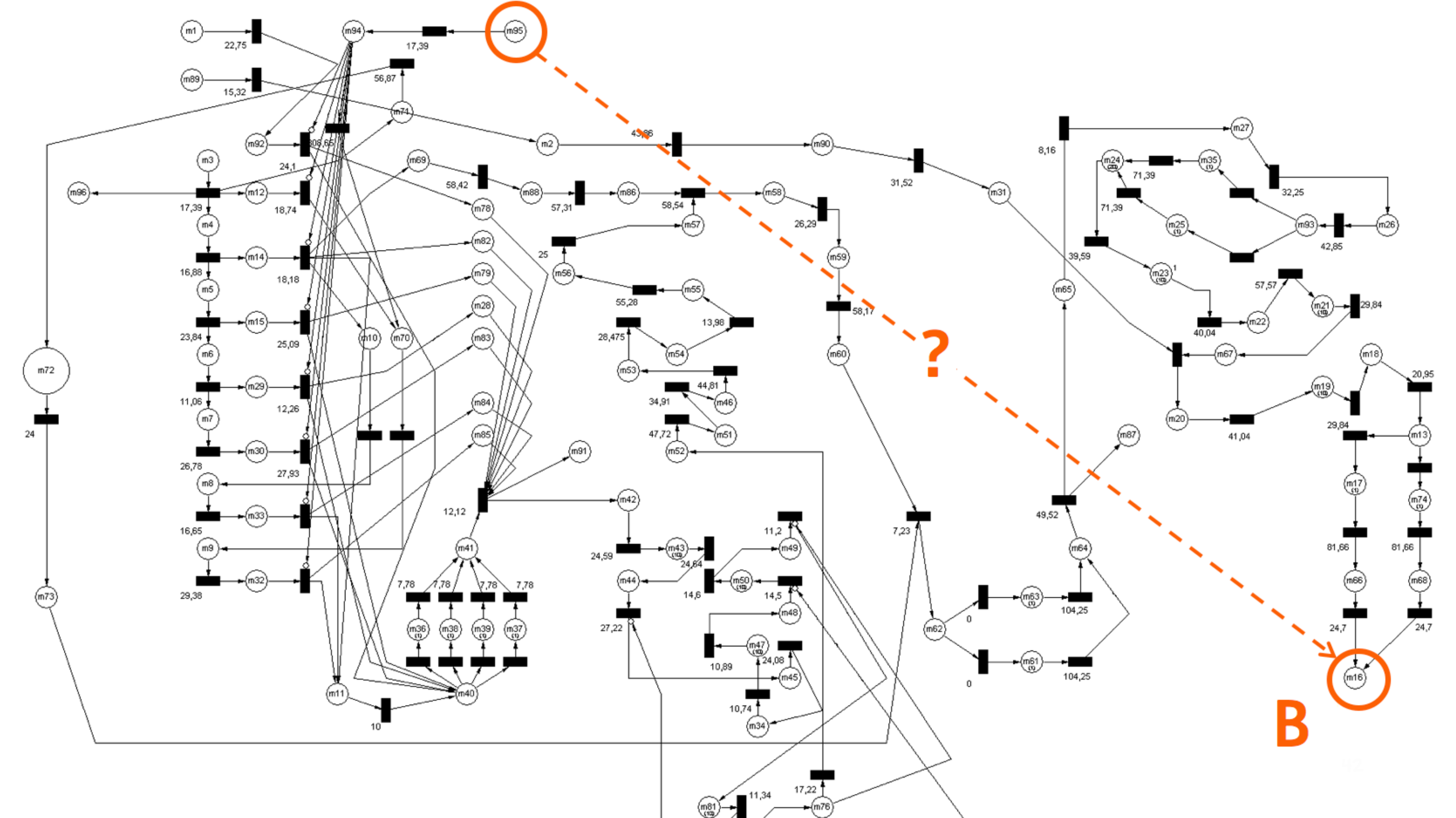
Checking Account	Savings Account	Customer Onboarding
Verify Identity	Verify Identity	Verify Identity
Register Individual	Register Individual	Register Individual
Open *Checking* Account	Open *Savings* Account	n/a
Issue Debit Card	n/a	n/a
Send Message	Send Message	Send Message
Register Product Owner	Register Product Owner	n/a

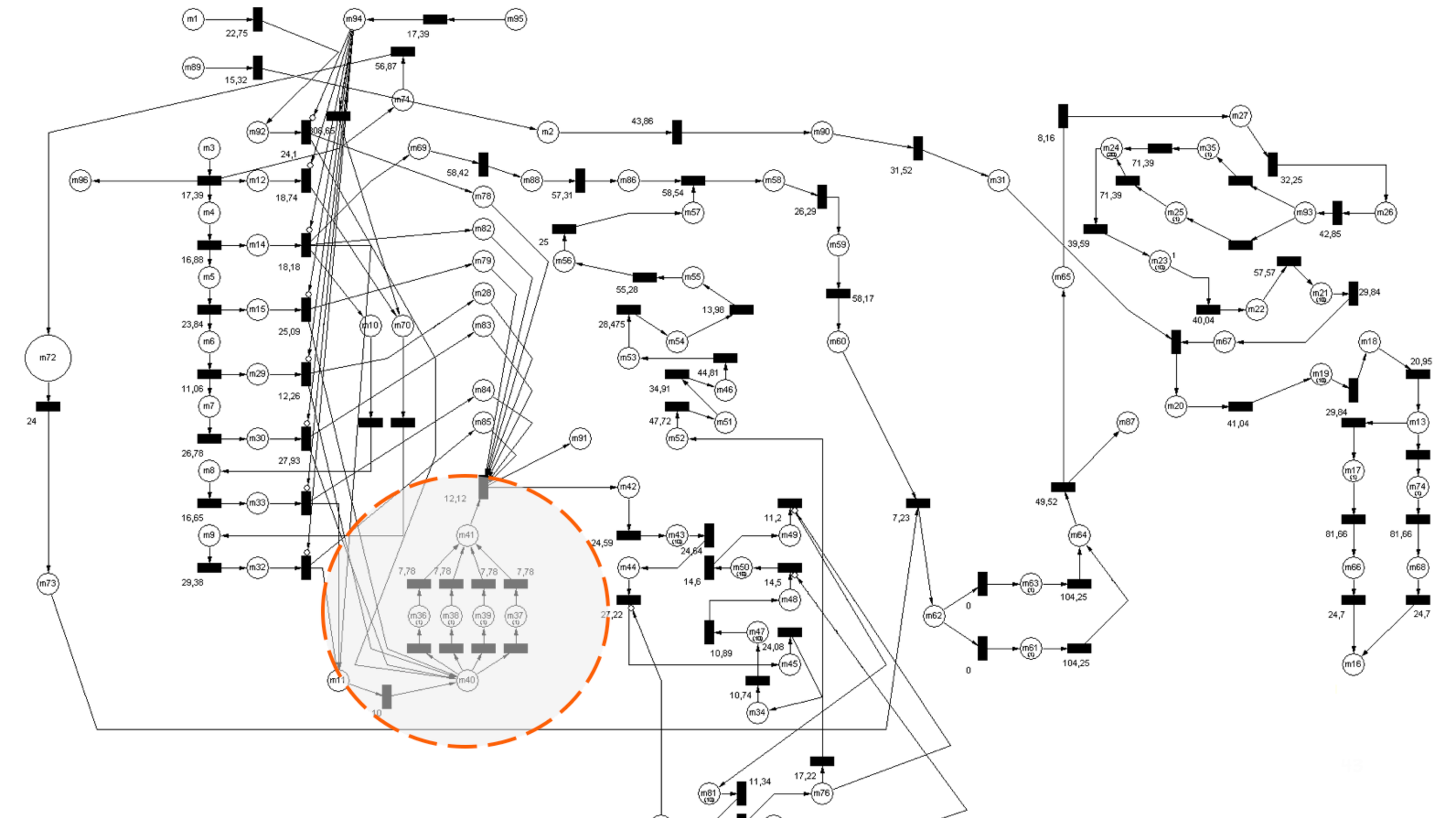

https://github.com/ing-bank/baker

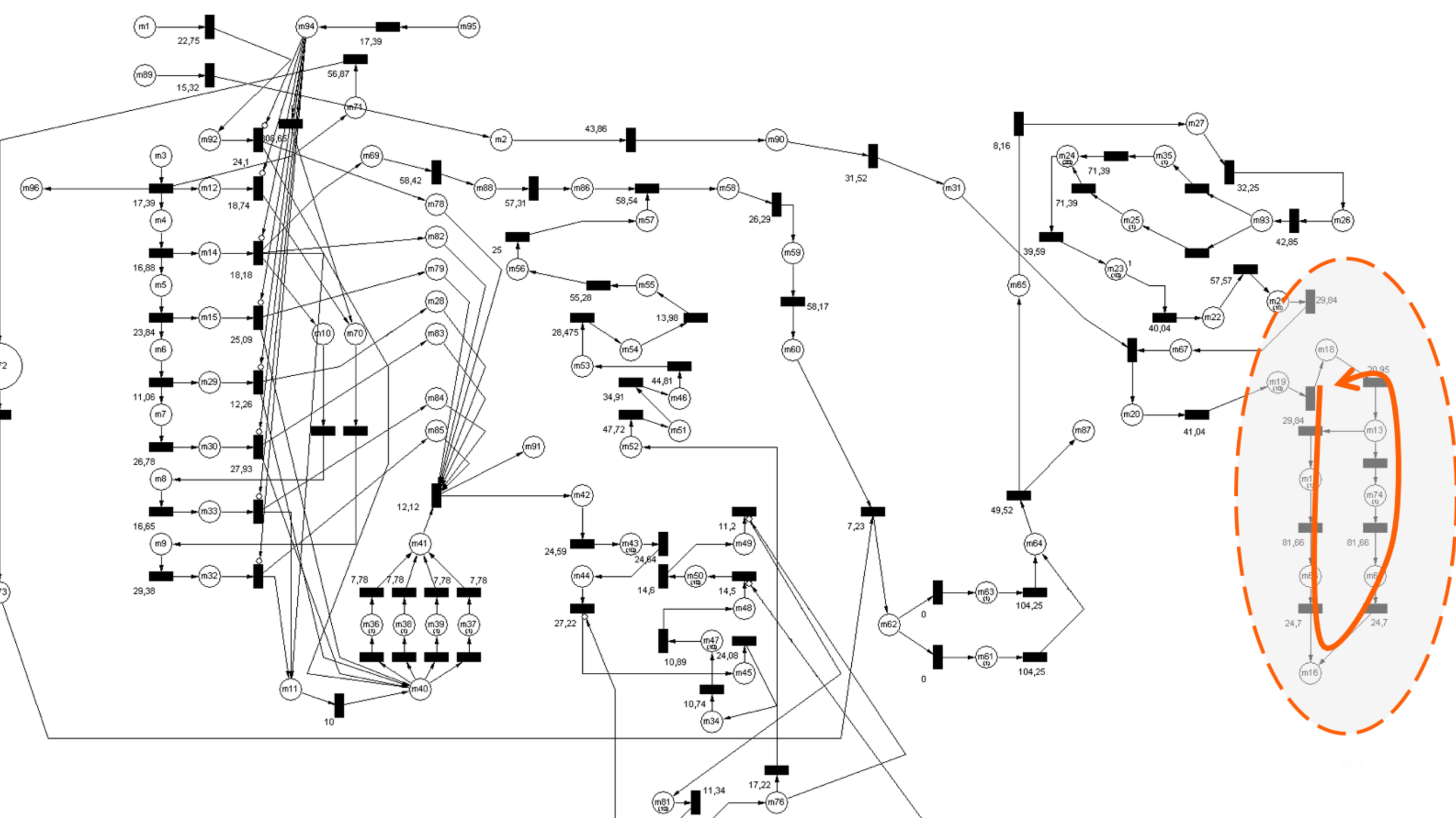

ipheVWv
cr58plDq
uye7bqgj
0smhq5zz
ilybUeM
oPpvnowk
tvUpthlx
dvptbxog
lecmtlf
tmMjbyQn
amAxlqX3
puuFZezo
0orn8eqwk
bygOjmd2
zoiWfnfn
hu2Cdjej
yFvjlt6x
kdxu5jwy
znsgwm6t
4qocacfw
zig2suwy
nwtxtJ7v
JycAasd7
nngahd4w
plfe1j7f
e2tptJjf
ntxoeton
pmayswdq
ouojkjmr
v4knoaon
v5lNlZVd
yqsaofOj
ukoezxus
jh7zfbcz
OR
k155nvmfx
NtbfVUf
lnJ004os
SOpoigkz
Sw0X1Qln
m7xsee11
inWrkJtb
tajjD1kl
kp4lvbrm
znbmsVe6
MPd4vYLf
rFvyjpu7
qufxefsC
lm2Ecazs
ensgahgb
i4zmZgfw
kgakajp2
lnNpqlds
fIJ5fOtn
Ymlxsd0V
m4Jqvcs4
k1webfxg
ger8vvjj
m7sabpfd
redyxkak
dlfasp44
m43tstlt
8dasVwr
Henzzlrf
Sayvk4dj
pwnbkm6c
xVjflrmvl
uylqirLd
qmhKmblx
gsg8hndb
qbgRquky
szxjRmtZ
h8ol6xn
znvmwlOOc
wbxopfUo
wnkSp8tn

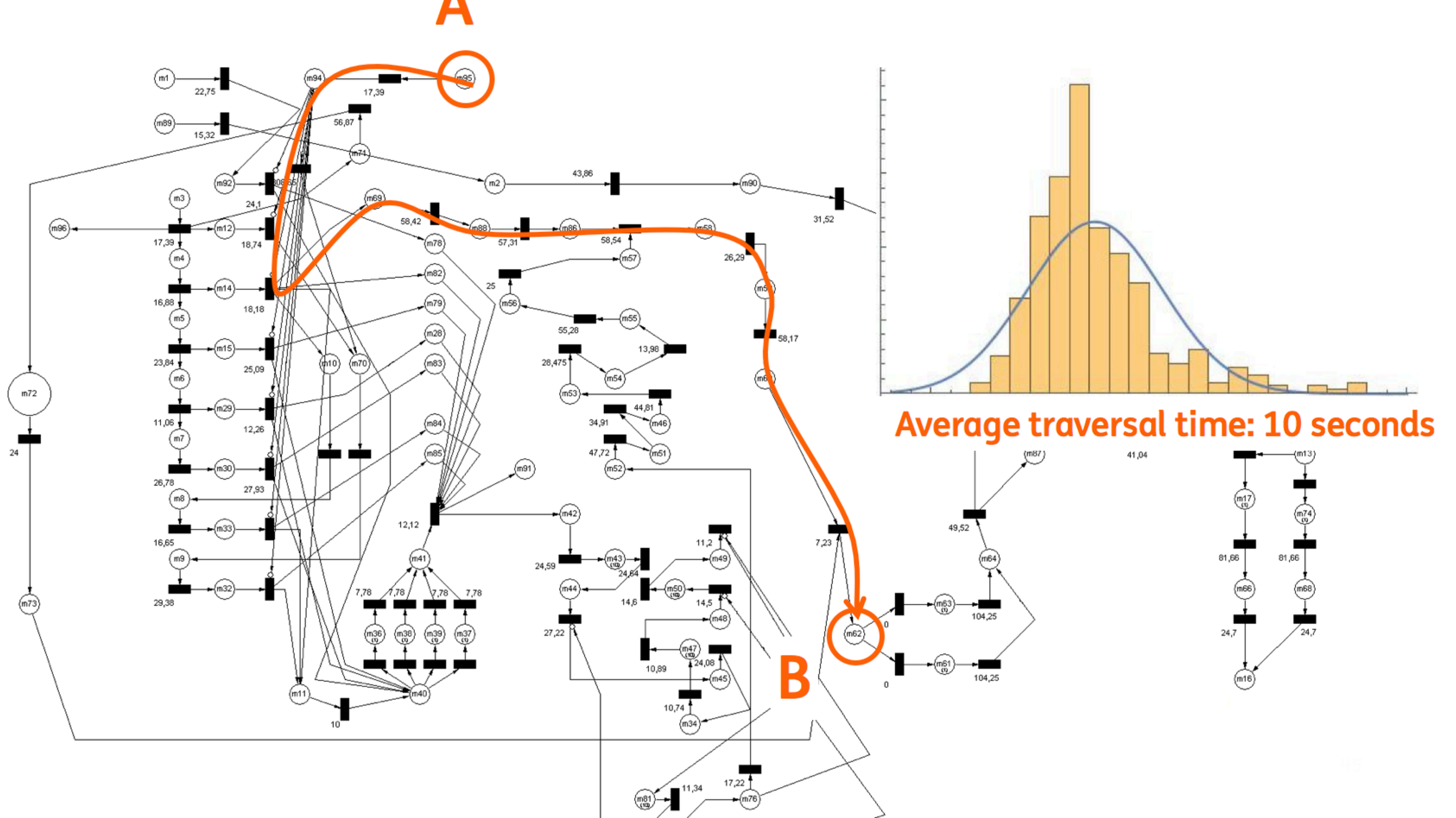
Why Petri net?[3]

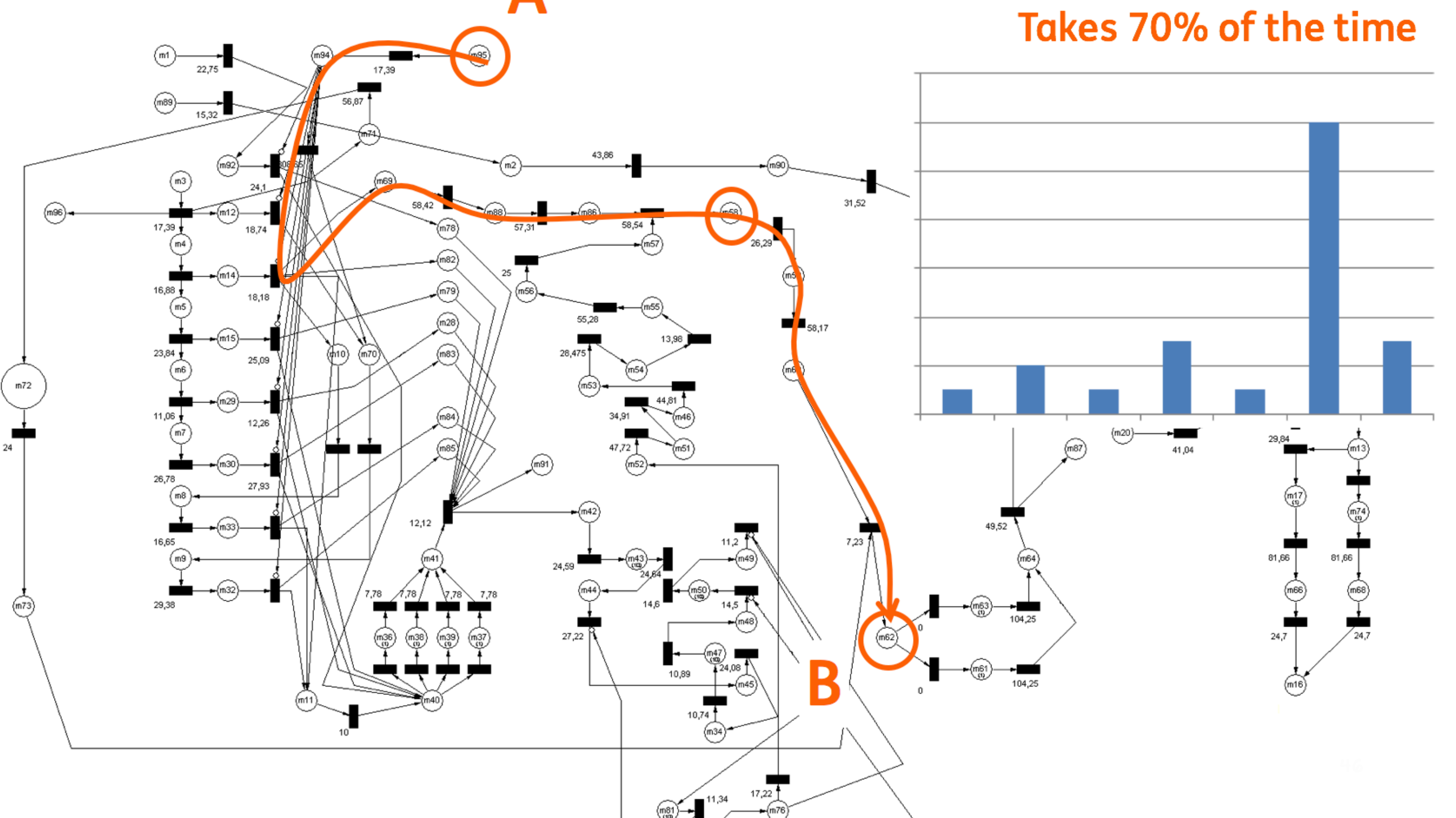
[3] https://en.wikipedia.org/wiki/Petri_net


m1
22,75
m94
17,39
m95
m89
15,32
56,87
m74
m92
m2
43,86
m90
31,52
m31
m3
24,1
m69
58,42
m88
57,31
m86
58,54
m58
26,29
m59
m96
17,39
m12
18,74
m78
m4
m14
18,18
m82
m57
25
m56
55,28
m55
m79
16,88
m5
m15
25,09
m28
13,98
28,475
m54
m10
m70
m83
23,84
m6
m29
12,26
m53
44,81
34,91
m46
m72
11,06
m7
m84
47,72
m51
m85
m52
24
m30
27,93
m91
26,78
m8
m42
12,12
m33
16,65
m9
m41
24,59
m43
24,64
11,2
m49
m73
m32
29,38
m44
m50
14,6
14,5
m48
7,78
7,78
7,78
7,78
27,22
m36
m38
m39
m37
m47
10,89
24,08
m45
m11
m40
10,74
m34
10
m81
11,34
17,22
m76
m62
0
m63
104,25
0
m61
104,25
7,23
m87
49,52
m64
8,16
m27
m24
71,39
m35
32,25
71,39
m25
m93
m26
42,85
39,59
m23
1
57,57
m21
29,84
m65
40,04
m22
58,17
m67
m18
20,95
m19
m20
41,04
29,84
m13
m17
m74
81,66
81,66
m66
m68
24,7
24,7
m16
?
B


m1
22,75
m94
17,39
m95
m89
15,32
56,87
m74
m92
306,65
m2
43,86
m90
31,52
m31
m3
24,1
m69
58,42
m88
57,31
m86
58,54
m58
26,29
m59
58,17
m60
m96
17,39
m12
18,74
m78
m57
25
m56
55,28
m55
13,98
28,475
m54
m53
44,81
34,91
m46
47,72
m51
m52
m4
16,88
m14
18,18
m82
m5
23,84
m15
25,09
m79
m10
m70
m28
m6
11,06
m29
12,26
m83
m7
26,78
m30
27,93
m84
m85
m8
16,65
m33
m91
m9
29,38
m32
m72
24
m73
12,12
m41
7,78
7,78
7,78
7,78
m36
m38
m39
m37
m11
m40
10
m42
24,59
m43
24,64
m44
27,22
14,6
m50
14,5
11,2
m49
m48
10,89
m47
24,08
m45
10,74
m34
m81
11,34
17,22
m76
7,23
m62
0
m63
104,25
0
m61
104,25
49,52
m64
m87
8,16
m27
32,25
m24
71,39
m35
71,39
m25
m93
42,85
m26
39,59
m23
1
57,57
m21
29,84
40,04
m22
m65
m67
m20
41,04
m19
m18
20,95
29,84
m13
m17
m74
81,66
81,66
m66
m68
24,7
24,7
m16


m1
22,75
m94
17,39
m95
m89
15,32
56,87
m74
m92
306,65
m2
43,86
m90
31,52
m31
8,16
m27
m24
71,39
m35
32,25
71,39
m25
m93
m26
42,85
39,59
m23
1
m65
57,57
m2
29,84
40,04
m22
m67
m18
20,95
m19
m20
41,04
29,84
m13
m1
m74
81,66
81,66
24,7
24,7
m16
m3
24,1
m69
58,42
m88
57,31
m86
58,54
m58
26,29
m59
58,17
m60
m96
17,39
m12
18,74
m78
m4
16,88
m14
18,18
m82
m5
23,84
m15
25,09
m79
m10
m70
m28
m6
11,06
m29
12,26
m83
m7
26,78
m30
27,93
m84
m8
16,65
m33
m85
m9
29,38
m32
m91
12,12
m42
24,59
m43
24,84
m41
m44
27,22
14,6
m50
14,5
11,2
m49
m48
m47
10,89
24,08
m45
10,74
m34
7,78
7,78
7,78
7,78
m36
m38
m39
m37
m11
10
m40
m55
55,28
m56
25
m57
28,475
m54
13,98
m53
44,81
34,91
m46
47,72
m51
m52
7,23
m62
0
m63
104,25
0
m61
104,25
49,52
m64
m87
m81
11,34
17,22
m76


A
B
Average traversal time: 10 seconds


A
Takes 70% of the time
B
m1
22,75
m89
15,32
m94
17,39
m95
56,87
m74
m92
24,1
m2
43,86
m90
31,52
m3
m96
17,39
m12
18,74
m69
58,42
m88
57,31
m86
58,54
m58
26,29
m78
m57
m4
16,88
m14
18,18
m82
m79
25
m56
55,28
m55
m5
58,17
m28
m83
m5
23,84
m15
25,09
m10
m70
13,98
28,475
m54
m53
m6
11,06
m29
12,26
m84
44,81
34,91
m46
m7
47,72
m51
m52
m72
24
m85
m91
m8
26,78
m30
27,93
12,12
m42
m33
16,65
m41
24,59
m43
24,64
11,2
m49
m9
m44
14,6
m50
14,5
m48
m32
29,38
7,78
7,78
7,78
7,78
27,22
m36
m38
m39
m37
m47
10,89
24,08
m45
m73
m11
m40
10
10,74
m34
7,23
m62
0
m63
104,25
49,52
m64
m87
0
m61
104,25
m20
41,04
29,84
m13
m17
m74
81,66
81,66
m66
m68
24,7
24,7
m16
m81
11,34
17,22
m76

I love cooking food and for the rest of the talk I'll be using analogies from there. It's very **similar to our industry**: long hours, hard work, and delivering experiences to our customers.

Have you been woken up at 3 o'clock in the morning on a **Saturday morning** after a night of partying, having to go to the war room and resolve an application incident. I've been there. When I remember the cold of the **airconditioners**, it still **makes me shiver.**

If we are building microservices or a monolith or any type of application in general we are **serving business logic to our clients.** So no matter what, we cannot escape the **architectural discussion.** If we are not careful of how we architect our applications we end up serving a bad meal to our clients.