BPE

Business Processing for Enterprise Done Right

BPE: Business Processing
for Enterprise
Done Right

FIRST EDITION

Book Design and Illustrations by Maxim Sokhatsky
Author Maxim Sokhatsky

Editors: Stanislav Spivakov
Viktor Sovietov

Publisher imprint:
Toliman LLC
251 Harvard st. suite 11, Brookline, MA 02446
1.617.274.0635

Printed in Ukraine

Order a copy with worldwide delivery:
https://balovstvo.me/bpe

ISBN: 978-1-62540-052-9

Contents

To all office workers and rest sentient beings.

1 TPS: Financial Processing

The TPS is an transactional database application that provides distributed, fault tolerant, network-split resistant, performant, transactional intermediary processing. It also guarantees data locality for storing customer objects which allows TPS to be easily scalable and maintainable. TPS is supported following backends:

1 **sql** transactional processing;
2 **riak_kv** application;
3 **mnesia** application;

TPS is based on KVS and CR. Rules for TPS are to be defined using UPL language.

1.1 Data Locality Cache Ring

All customers of bank are being grouped or sharded using custom hash function that is known to be linear and consistent. This function allows TPS to store all customer information for a given master of its key on a single node. Thus all TPS operations on transactions, cards, account of a given customer are passed to a hash function with a same customer ID key. In TPS all properties of a top customer object with the same key are stored in the same VNODE.

1.2 Riak

All interface operations and application data are stored and being read from Riak TPS Ring, which is in fact Riak Core application. Each VNODE should be treated as Bank department or isolated Bank part which could be detached to other place or Data Center. Each Riak Core VNODE TPS application also has access to its SQL warehouse, the primary source of transactions.

1.3 Transactions Rate Calculation

Here we define the formula of expected Transactions per month, that is the source for all system configuration. Here is example:

The operational data is fixed. E.g. we have 30M customers. Consider each customer performs 10 transactions in a month, thus we have 300M transaction. Each transaction has 2K in its size. So wee need 600G space of a cluster in a month. After each month we could outsource this data or even reduce it by cutting the tails of transactions list. Also these 600G can be divided by the number of nodes with accuracy to a coefficient of replication for TPS Ring. For SQL warehouse we double its size for SQL warm stand by failover. Also the number of VNODEs in TPS Ring is exactly the number of failover SQL instances.

1.4 Financial Warehouse Operations

CHARGE	unconditional INSERT and head UPDATE inside TRANSACTION, Cache Write-Back
WITHDRAW	conditional INSERT, based on last known amount of latest customer transaction, and chain root UPDATE inside TRANSACTION, Cache Write-Back

All SQL Operations also perform write backs to TPS Cache, which is backed by Riak.

1.5 Cache Operations

— Retrieval Transactions for an Account
— Display Accounts for a Customer
— Display list of Cards for a Customer
— Retrieval of details of an TPS object

1.6 Maintenance Operations

— Cutting Tails in Transactions list
— Perform Recalculation of UPL rule for a Time Range
— Adding/Removing Nodes

1.7 Database Schema

— customer: basic bank client profile
— account: IBAN identified bank account
— card: EMV issued card
— transaction: tracking record for beneficiary and subsidiary
— cashback: dfferent programs for credit transactions
— program: UPL encoded deposit or credit program

2 BPE: Business Processes

What is BPE? BPE is an production grade business workflow engine that is enought for managing automated procesess. It can substitute WWF, BizTalk, Activity or Oracle BPM for those who understand the basic features of workflow systems. BPE is an subset of BPMN 2.0 standart, the evelution and unification of most previous worflow standard such as XPDL, BPML, OpenWFE, WWF and jBPM.

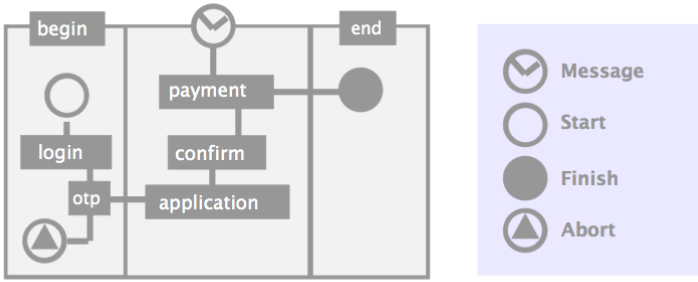

Figure 1: Process Sample

This project was written essentially for biggest ukrainian commercial bank and is based on previous research work already done by athor for CLR platform. You may read a dedicated book about Workflow Process theory written by BPE author earlier while this book is dedicated essentially to BPE application and its companion libraries. The same way as web frameworks are the core of web applications, the workflow engine is a core of business applications.

In its core BPE is microkernel that accept native Erlang record-based process definition in Erlang language along with Event-Condition-Action erlang functions, similar you may find in OTP principles, but much

simplified, intended for business analytics and system integration engineers. It acts like FSM machine, runned under Erlang supervision and is totally persisted, which means no data loss can be happened. It has very clean and minimalistic API for keeping this product robust and stable, th approach you already can find in Synrc stack in common and in N2O framework in particular.

BPE context holds KVS documents which in fact typed Erlang records and are defined by KVS schema. Each process has its own persistent execution log and is fully fault-resistant with migration capabilities. The BPE protocol is well defined and is a part of N2O protocols stack.

The author of BPE has implemented business workflow engine for CLR virtual machine using C# language. However Erlang implementation more idiomatic and canonical due to semantic corresponding of process calculus and the core of underlying virtual machine. Send async messages across processes means exactly what it says up to Erlang pids. For sending documents to business process you can use process's name or its Pid:

```
1> bpe:amend(Process#process.id,#deposit{}).
```

Thanks to this isomorphic corresponding between Erlang process and Calculus process, code size of core BPE server was reduced to 400 lines of code. This is definitely most clean functional implementation of workflow engine avalible in electronical banking systems.

2.1 Pi-calculus and Petri nets

The nice thing about all palette of different implementation of workflow models is that all of them reduced to one of two kinds of encoding: one is algebraic one and the other is geometric.

The geometric one is **Petri nets**. Carl Petri introduced it in 1962 during discrete analysis of asynchronous computer systems. Any its graphical representation could be defined with Petri nets formalism. Petri modeling in one of its forms is a good complementation to process algebra useful as computational model.

The algebraic one is **Pi-calculus** developed by Robin Milner who gained Turing award for 1) Meta Language ML, 2) Calculus for Communication Systems CCS (1980), the general theory of concurrency and 3) theoretical base for proof assistants, Logic for Computable Functions LCF. The model of process calculus is a theoretical background of virtual environment of Erlang infrastructure, so BPE implementation fully relies on Pi-calculus (1999), the successor of CCS notion. Thus providing effective computational model for implementation of workflow process management.

2.2 Finite State Machines

One of the common known types of encoding process calculus is well developed **FSM framework** (60-s). This language is widely used almost in any programming language presented as core feature or as library. The process defines with an extension to Turing machine with states, input, outputs and functions.

2.3 SADT

The next language (framework) that used in (80-s, 90-s) to describe similar to process calculus definitions with graphical Petri nets and model definitions was **SADT** introduced by Marca and MacGowan 1988, 1991.

2.4 Reactive Systems

One of the wide range of semantics is Reactive Systems based on message passing and event routing, but also it could be known as Functional Reactive Programming FRP which is rather a set of combinators over streams. Both interpretations are used in languages and frameworks, depending on involvement of stream in core definition (2010-s).

2.5 Typing Pi-calculus

In typed theory Pi-calculus defines also the typing system (could be System F, e.g.) for input and outputs of processes or function signatures specified in process definition. In BPE the role of types was taken by document types, which is simple Erlang records, so in BPE workflow processing is type-safe on compilation stage with respect to document types.

```
1> #deposit{} = bpe:doc(#deposit{}, pid(0,185,0)).
```

2.6 Scenarios

Workflows are complimentary to business rules and could be specified separately. BPE definitions provides front API to the end-user application. Workflow Engine – is an Erlang/OTP application which handles process definitions, process instances, and provides very clean API for Workplaces.

Before using Process Engine you need to define the set of business process workflows of your enterprise. This could be done via Erlang terms or some DSL that lately converted to Erlang terms. Internally BPE uses Erlang terms workflow definition:

```
bpe:start(#process{name="Order1",
          flows=[#sequenceFlow{source='Start',target='Mid'},
                 #sequenceFlow{source='Mid',target='Finish'}],
          tasks=[#userTask{name='Start'},
                 #userTask{name='Mid'},
                 #userTask{name='Finish'}],
          beginEvent='Start',endEvent='Finish'},[]).
```

Slightly bigger example:

```
deposit_app() -> #process { name = 'Create Deposit Account',

    flows = [
        #sequenceFlow{source='Init',      target='Payment'},
        #sequenceFlow{source='Payment',   target='Signatory'},
        #sequenceFlow{source='Payment',   target='Process'},
        #sequenceFlow{source='Process',   target='Final'},
        #sequenceFlow{source='Signatory', target='Process'},
        #sequenceFlow{source='Signatory', target='Finish'}
    ],

    tasks = [
        #userTask    { name='Init',      module = deposit },
        #userTask    { name='Signatory', module = deposit},
        #serviceTask { name='Payment',   module = deposit},
        #serviceTask { name='Process',   module = deposit},
        #endEvent    { name='Finish'}
    ],

    beginEvent = 'Init',
    endEvent = 'Final',
    events = [
         #messageEvent{name="PaymentReceived"}
    ]
}.
```

2.7 Actions

Business rules should be specified by developers. RETE is used for rules specifications which can be triggered during workflow.

2.8 BPMN 2.0

The workflow definition uses following persistent workflow model which is stored in KVS:

```
-record(task,          { name, id, roles, module }).
-record(userTask,      { name, id, roles, module }).
-record(serviceTask,   { name, id, roles, module }).
-record(messageEvent,  { name, id, payload }).
-record(beginEvent ,   { name, id }).
-record(endEvent,      { name, id }).
-record(sequenceFlow,  { name, id, source, target }).
-record(history,       { ?ITERATOR(feed,true), name, task }).
-record(process,       { ?ITERATOR(feed,true), name,
                         roles=[], tasks=[], events=[],
                         history=[], flows=[], rules,
                         docs=[], task, beginEvent,
                         endEvent }).
```

Full set of BPMN 2.0 fields could be obtained at OMG definition document, page 3-7[1].
Unusual that BPE process implemented as **gen_server** rather that **gen_fsm**:

Listing 1: Boundary Event

```
process_event(Event, Proc) ->
  Targets = bpe_task:targets(element(#event.name,Event),Proc),
  {Status,{Reason,Target},ProcState} =
    bpe_event:handle_event(Event,
      bpe_task:find_flow(Targets),Proc),
  NewState = ProcState#process{task = Target},
  FlowReply = fix_reply({Status,{Reason,Target},NewState}),
  kvs:put(NewState),
  FlowReply.
```

[1] http://www.omg.org/spec/BPMN/2.0

Listing 2: Flow Processing

```
process_flow(Proc) ->
  Curr = Proc#process.task,
  Term = [],
  Task = bpe:task(Curr,Proc),
  Targets = bpe_task:targets(Curr,Proc),
  {Status,{Reason,Target},ProcState}
    = case {Targets,Proc#process.task} of
  {[],Term} -> bpe_task:already_finished(Proc);
  {[],Curr} -> bpe_task:handle_task(Task,Curr,Curr,Proc);
  {[],_}    -> bpe_task:denied_flow(Curr,Proc);
  {List,_}  -> bpe_task:handle_task(Task,Curr,
               bpe_task:find_flow(List),Proc) end,

  kvs:add(#history{ id = kvs:next_id("history",1),
      feed_id = {history,ProcState#process.id},
      name = ProcState#process.name,
      task = {task, Curr} }),

  NewState = ProcState#process{task = Target},
  FlowReply = fix_reply({Status,{Reason,Target},NewState}),
  kvs:put(NewState),
  FlowReply.
```

Listing 3: BPE protocol

```
  {run}
  {until,_}
  {complete}
  {complete,_}
  {amend,_}
  {amend,_,_}
  {event,_}
```

2.9 Erlang Session

```
> bpe:start(spawnproc:def(),[]).
  bpe_proc:Process 18 spawned <0.961.0>
  {ok,18}

> bpe:complete(18).
  ACT Deposit Init
  bpe_proc:Process 18 Task: 'Init' Targets: ['Payment']
  bpe_proc:Process 18 Flow Reply {reply,{complete,'Payment'}}
  {complete,'Payment'}

> bpe:complete(18).
  ACT Deposit Payment
  bpe_proc:Process 18 Task: 'Payment' Targets: ['Signatory','Process']
  bpe_proc:Process 18 Flow Reply {reply,{complete,'Signatory'}}
  {complete,'Signatory'}

> bpe:complete(18).
  bpe_proc:Process 18 Task: 'Signatory' Targets: ['Process','Final']
  bpe_proc:Process 18 Flow Reply {reply,{complete,'Process'}}
  {complete,'Process'}

> bpe:complete(18).
  ACT Deposit Process
  bpe_proc:Process 18 Task: 'Process' Targets: ['Final']
  ACT Create Account {user,18,[],feed,[],[],[],[],
                      true,[],[],[],[],
                      [],[],[],[],[],[],[],
                      [],[],[],[],[],[]}
  bpe_proc:Process 18 Flow Reply {reply,{complete,'Final'}}
  {complete,'Final'}

> bpe:complete(18).
  bpe_proc:Process 18 Task: 'Final' Targets: []
  bpe_proc:Process 18 Flow Reply {stop,{normal,'Final'}}
  'Final'
  bpe_proc:Terminating session Id cache: 18
  Reason: normal
```

Take last two from history

```
> bpe:history(18,2).
  [#history{id = 9,version = undefined,container = feed,
            feed_id = {history,18},
            prev = 8,next = 10,feeds = [],guard = true,etc = undefined,
            name = 'Create Deposit Account',
            task = {task,'Process'},
            time = {{2016,8,16},{20,19,39}}},
   #history{id = 10,version = undefined,container = feed,
            feed_id = {history,18},
            prev = 9,next = undefined,feeds = [],guard = true,
            etc = undefined,name = 'Create Deposit Account',
            task = {task,'Final'},
            time = {{2016,8,16},{20,20,0}}}]
```

Load terminated process and try to spawn

```
> bpe:start(bpe:load(18),[]).
  bpe_proc:Process 18 spawned <0.1008.0>
  {ok,18}

> bpe:complete(18).
  bpe_proc:Process 18 Task: 'Final' Targets: []
  bpe_proc:Process 18 Flow Reply {stop,{normal,'Final'}}
  bpe_proc:Terminating session Id cache: 18
  Reason: normal
  'Final'
```