Business Process Management ISO 19510:2015

ERP/1: Processes

Zen Crypted

Process Orchestration Prerequisites

Overview

Process Management is needed if these criteria meet at scale:

- Formal Definition of the Process
- Process Governance (Versioning, Archiving)
- Manual Edition
- Human Code Review of the Process
- Process Audit (Public Key Infrastructure)
- Process Automation
- Process Tracing
- Governmental and Defense Contracts

History of Processing Languages

— EMAIL: FSM
— Event-Condition-Action Reactive Rule Engines
— Expert Systems: RETE Engine, Prolog
— Workflow Standards of the past: XPDL, BPML, OpenWFE, WWF and jBPM
— Workflow Standard After 2008: BPMN
— Trading: TpML, Fix, business contract routers, cross-exchanges, arbitrage
— Business Contacts Virtual Machines: EVM, Script VM, aebytecode
— Business Contract Languages: Sophia, Solidity, Plutus
— MLTT Frameworks: Dhall
— Interaction Networks Evaluators: Formality, Moonad
— Stream Processing: Oz, Erlang, np/ling, Futhark

Isomorphic Process Models

Overview

Process Management has a series of manifestation, most notable among them:

- BPMN
- Finite State Machines (gen_server, gen_fsm, gen_statem, etc.)
- SADT
- np/ling
- Event Condition Action
- Petri Nets
- Pi-Calculus
- Linear Types (Lock Calculus)
- Tensor Calculus (Symmetric Monoidal Categories)

Two Computational Models

Lambda Calculus (Alonzo Church)

Non-typed core of **Cartesian Closed Categories** inner language. Arguments of functions are **Scalars**.
Has theoretical flaw in its core evaluator due to single core nature of thinking.

Pi Calculus (Yves Lafont)

Non-typed core of **Symmetric Monoidal Categories** inner language. Arguments of functions are **Tensors** with forward only time axis access. Notable Applications: Parallel Concurrent (Modal) Models like Erlang, GPU parallelization (Maya Victor).

BPE/BPMN Informal Model (Handwaving, CPO)

```
def Process (P: U) (⊙ ⊕: U → U) : U
 := inductive { start : ⊤ → P
              | run, stop : P → ⊥
              | get : ⊙ P → P
              | step : ⊕ P → P
              }
```

BPE/BPMN Formal Model (MLTT)

```
storage : U -> U = list

process : U := Σ (protocol state: U) (current: prod protocol state)
  (action: id (prod protocol state)), storage (prod protocol state)

spawn (protocol state: U) (init: prod protocol state) (action: id (prod protocol state))
  : process := (protocol, state, init, action, nil)

protocol (p: process) : U := p.1
state (p: process): U := p.2.1
signature (p: process) : U := prod p.1 p.2.1
current (p: process) : signature p := p.2.2.1
action (p: process) : id (signature p) := p.2.2.2.1
trace (p: process) : storage (signature p) := p.2.2.2.2

receive (p: process) : protocol p = axiom
send (p: process) (message: protocol p) : unit = axiom
```

BPE/BPMN Practical Model (Erlang)

```
     Documents : #document | {_,_}

        Errors : #error
           OKs : #history

Timers, Events : #event
         Tasks : #task
        Module : { action : Σ P → P }
```

Zen Crypted

BPE/BPMN Practical Model (Erlang)

```
boundaryEvent(name: "*", timeout: {0,{0,30,0}})

timer(event: :ping)

def action({request,'Stage'} = sigma, BPE.process() = process) do
  1/0
  {:reply, process}
end

:kvs.add(edge, history)

:kvs.add(error, errors)
```

Zen Crypted

BPE/BPMN Practical Model (System F/MLTT)

```
axiom start : Proc -> IO Sup
axiom stop : String -> IO Sup
axiom next : ProcId -> IO ProcRes
axiom amend : Π (k: U), ProcId -> k -> IO ProcRes
axiom discard : Π (k: U), ProcId -> k -> IO ProcRes
axiom modify : Π (k: U), ProcId -> k -> Atom -> IO ProcRes
axiom event : ProcId -> String -> IO ProcRes
axiom head : ProcId -> IO Hist
axiom hist : ProcId -> IO (List Hist)
```

KVS/NVMe Practical Model (System F/MLTT)

```
axiom get : Π (f k v: U), f -> k -> IO (Maybe v)
axiom put : Π (r: U), r -> IO StoreResult
axiom delete : Π (f k: U), f -> k -> StoreResult
axiom index : Π (f p v r: U), f -> Atom -> v -> List r
axiom next : Reader -> IO Reader
axiom prev : Reader -> IO Reader
axiom take : Reader -> IO Reader
axiom drop : Reader -> IO Reader
axiom save : Reader -> IO Reader
axiom append : Π (f r: U), f -> r -> IO StoreResult
axiom remove : Π (f r: U), f -> r -> IO StoreResult
```

BPE/BPMN Objects (Schema)

#process
#monitor
#continue
#lock

#task
#beginTask
#endTask
#userTask
#serviceTask
#receiveTask
#sendTask
#gateway

#sequenceFlow

#event
#boundaryEvent
#timeoutEvent
#messageEvent

#history

BPE/BPMN Morphisms (Functions)

API

- LOAD (Load Process State)
- START (Start Process under Supervision)
- PROC (Process Cache)
- NEXT (Stage Completion with Optional Target State Specify)
- COMPLETE (Complete the Stage of the Process)
- AMEND (Modification of Process State with Process Tick)
- DISCARD (Modification of Process State with Process Tick)
- APPEND (Modification of Process State)
- REMOVE (Modification of Process State)
- EVENT (Trigger Event)
- HIST (Get Process Trace)
- TASK (Get Task Info)
- DOC (Search Documents in Context)

Zen Crypted

BPE/BPMN Corporate Integration Cases

Worldwide

- Biggest Eastern European Bank ~30M subscribers (Deposit Application)
- Ministry of Internal Affairs (Including National Guard Forces)
- National Weapon Registry of MIA Ukraine
- NYNJA Communicator

Zen Crypted

BPE/BPMN Bank Account Example

Definition 1. Bank Account

```
beginEvent(id: "Created"),
userTask(id: "Init"),
userTask(id: "Upload"),
userTask(id: "Signatory"),
serviceTask(id: "Payment"),
serviceTask(id: "Process"),
endEvent(id: "Final")

sequenceFlow(id: "->Init",               source: "Created",   target: "Init"),
sequenceFlow(id: "->Upload",             source: "Init",      target: "Upload"),
sequenceFlow(id: "->Payment",            source: "Upload",    target: "Payment"),
sequenceFlow(id: "Payment->Signatory",   source: "Payment",   target: "Signatory"),
sequenceFlow(id: "Payment->Process",     source: "Payment",   target: "Process"),
sequenceFlow(id: "Process-loop",         source: "Process",   target: "Process"),
sequenceFlow(id: "Process->Final",       source: "Process",   target: "Final"),
sequenceFlow(id: "Signatory->Process",   source: "Signatory", target: "Process"),
sequenceFlow(id: "Signatory->Final",     source: "Signatory", target: "Final")

messageEvent(id: "PaymentReceived"),
boundaryEvent(id: :*, timeout: timeout(spec: {0, {10, 0, 10}}))
```

Zen Crypted

BPE/BPMN Support Ticket Example

Definition 2. Support Ticket

```
beginEvent(id: "Create"),
userTask(id: "Assign"),
userTask(id: "Work"),
serviceTask(id: "Escalate"),
userTask(id: "Resolve"),
endEvent(id: "Closed")

sequenceFlow(id: "->Assign", source: "Create", target: "Assign"),
sequenceFlow(id: "Assign->Work", source: "Assign", target: "Work"),
sequenceFlow(id: "Work->Resolve", source: "Work", target: "Resolve"),
sequenceFlow(id: "Resolve->Closed", source: "Resolve", target: "Closed"),
sequenceFlow(id: "Escalate->Work", source: "Escalate", target: "Work")

messageEvent(id: "TicketUpdate"),
boundaryEvent(id: :*, timeout: timeout(spec: {0, {24, 0, 0}}))
```

BPE/BPMN Purchase Order Example

Definition 3. Purchase Order

```
beginEvent(id: "CartCreated"),
userTask(id: "CartActive"),
serviceTask(id: "AbandonedRecovery"),
userTask(id: "Checkout"),
serviceTask(id: "Payment"),
serviceTask(id: "Fulfilment"),
serviceTask(id: "PostPurchase"),
endEvent(id: "Delivered")

sequenceFlow(id: "->CartActive", source: "CartCreated", target: "CartActive"),
sequenceFlow(id: "CartActive->Checkout", source: "CartActive", target: "Checkout"),
sequenceFlow(id: "AbandonedRecovery->CartActive", source: "AbandonedRecovery", target: "CartActive"),
sequenceFlow(id: "Checkout->Payment", source: "Checkout", target: "Payment"),
sequenceFlow(id: "Payment->Fulfilment", source: "Payment", target: "Fulfilment"),
sequenceFlow(id: "Fulfilment->PostPurchase", source: "Fulfilment", target: "PostPurchase"),
sequenceFlow(id: "PostPurchase->Delivered", source: "PostPurchase", target: "Delivered")

messageEvent(id: "PaymentReceived"),
messageEvent(id: "ShipmentUpdate"),
boundaryEvent(id: :*, timeout: timeout(spec: {0, {1, 0, 0}}))
```

Groupoid Infinity

Thank you!

Zen Crypted