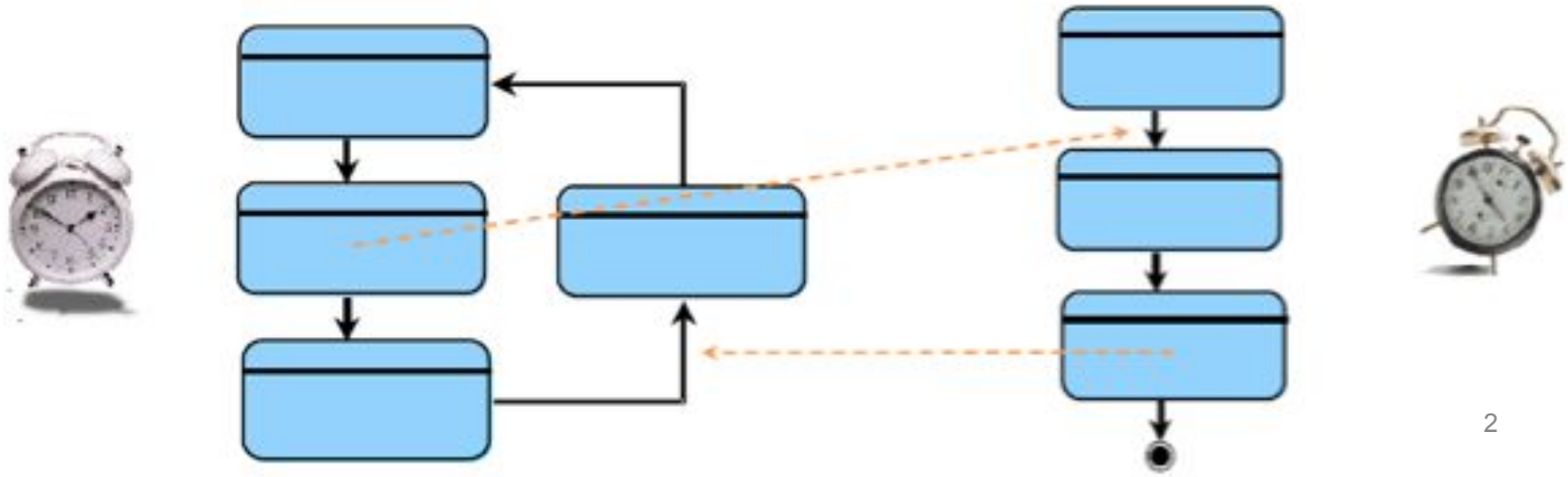
xtUML: Shlaer-Mellor Runtime with Cryptographic Extensions

Cortland Starrett
cortland.starrett@onefact.net

Outline

- How a S-M Runtime Works
- Operating Runtime Model
- Security Aims
- Leveraging S-M Runtime
- Examples
- Future





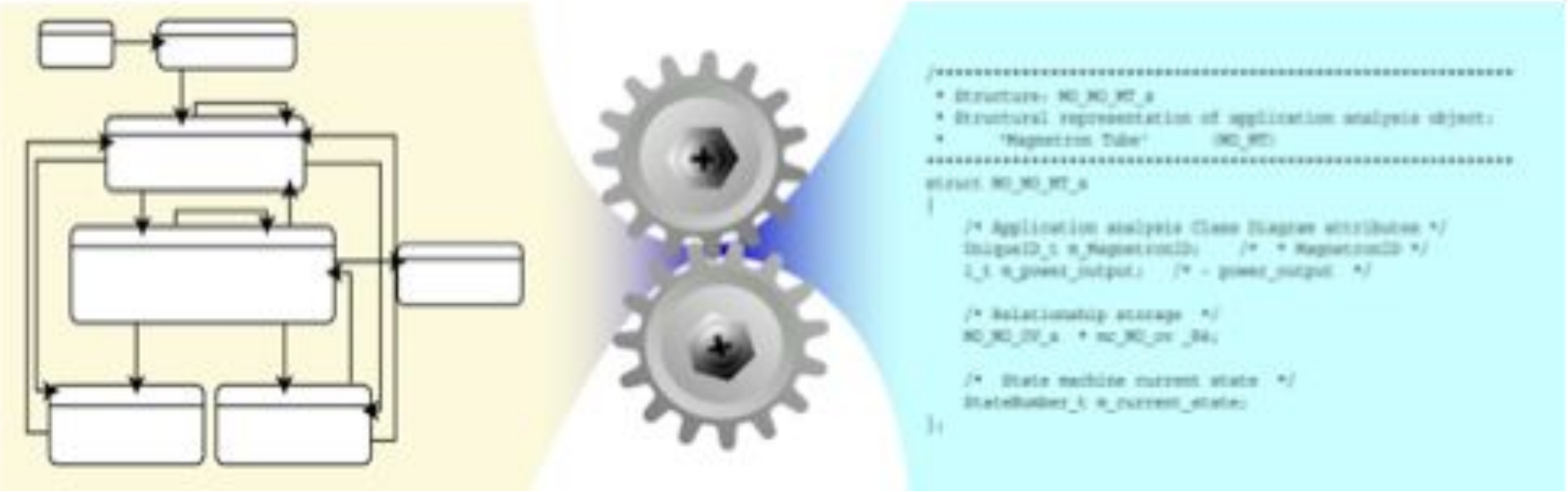
Aims of this Presentation

- elucidating fundamentals of Shlaer-Mellor runtime
- introducing security extensions to the runtime model
  - obscuring state sequencing
  - obscuring message sequencing
  - obscuring/leveling resource (energy) utilization
  - *not* speaking to existing function obfuscation techniques (*indistinguishability obfuscation*)

A working model of a S-M runtime is to be understood and explored with respect to managing observable energy consumption.

Elements of a Shlaer-Mellor Software Architecture

- direct translation of model elements (mapping)
- collections
- runtime core



Direct Translation of Model Elements (Mapping)

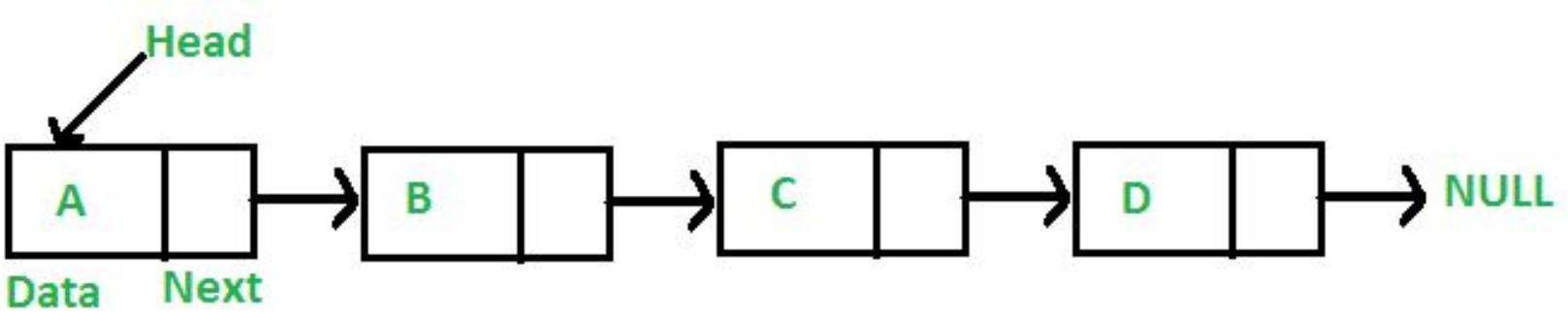
- classes, state machines, action language
  - class -> `class, struct, table`
  - state machine -> nested `switch/case` statement, 2 dimensional lookup table
  - action language -> each statement maps to lower level construct
    - relationship creation, deletion and navigation, etc.

```
struct Tracking_TrackPoint {

  /* application analysis class attributes */
  Escher_TimeStamp_t session_startTime;  /* * session_startTime (R1)(R3)(R2) */
  i_t time;  /* * time */
  r_t longitude;  /* - longitude */
```

Collections

- instances, relationship extents, transient sets
  - instances
  - relationship lists
  - transient sets



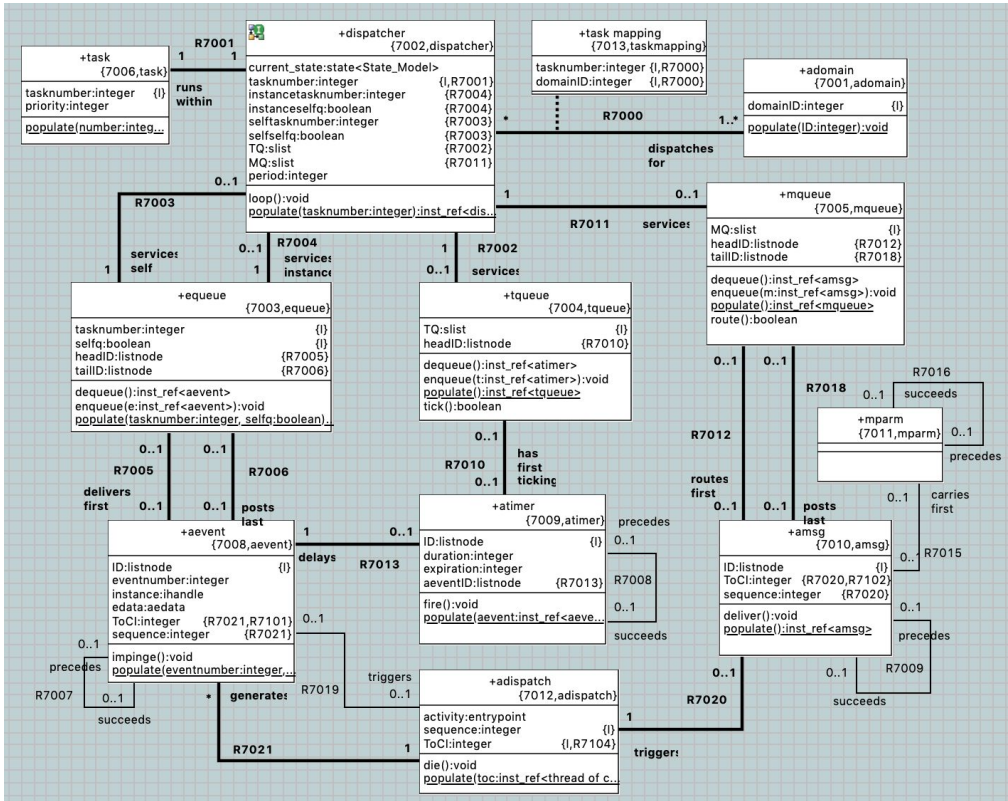
Runtime Core

- event, timer and message dispatching
  - event queueing (self and instance)
  - timer management
  - message passing
  - activity dispatch



Runtime Core

- This is the focus of this presentation.
- security extensions to be made to the runtime core

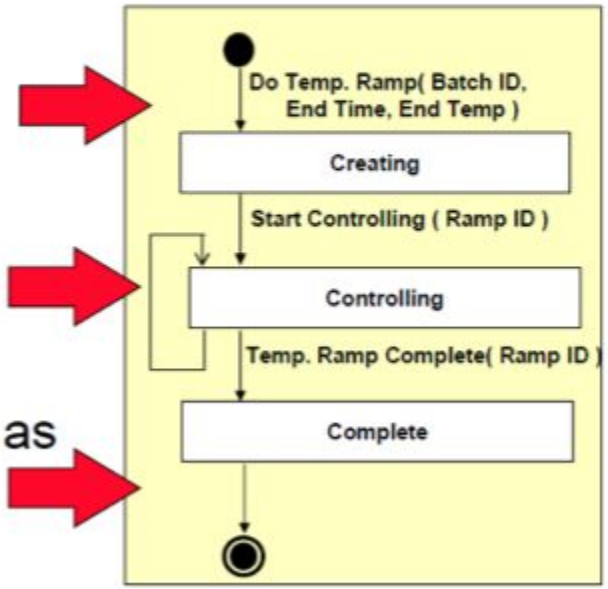


Simulation

- models themselves
- rules
- runtime

The model executes in response to signals from:

- the outside
- timers
- other instances as they execute

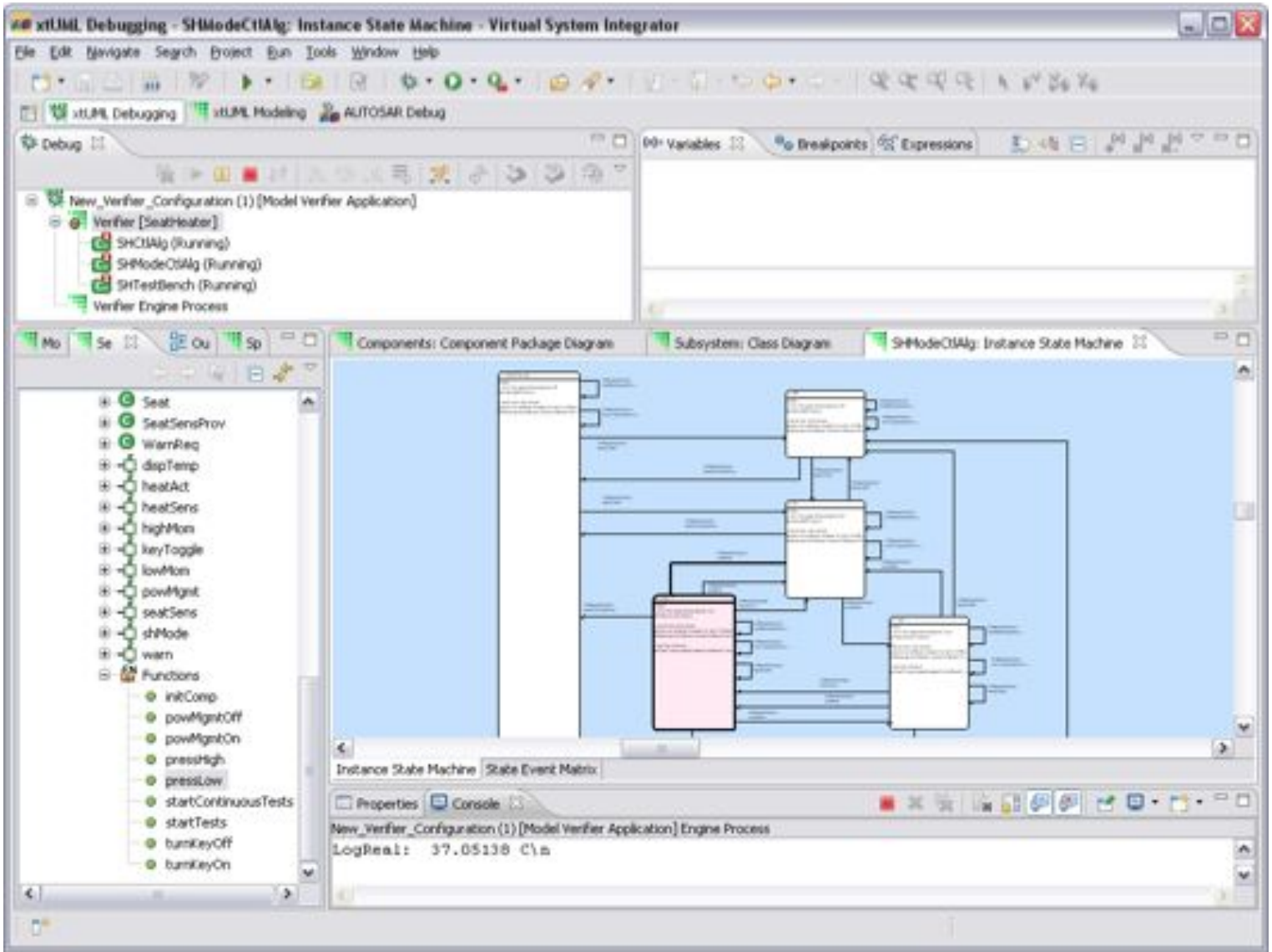


Simulation: Executing the Models

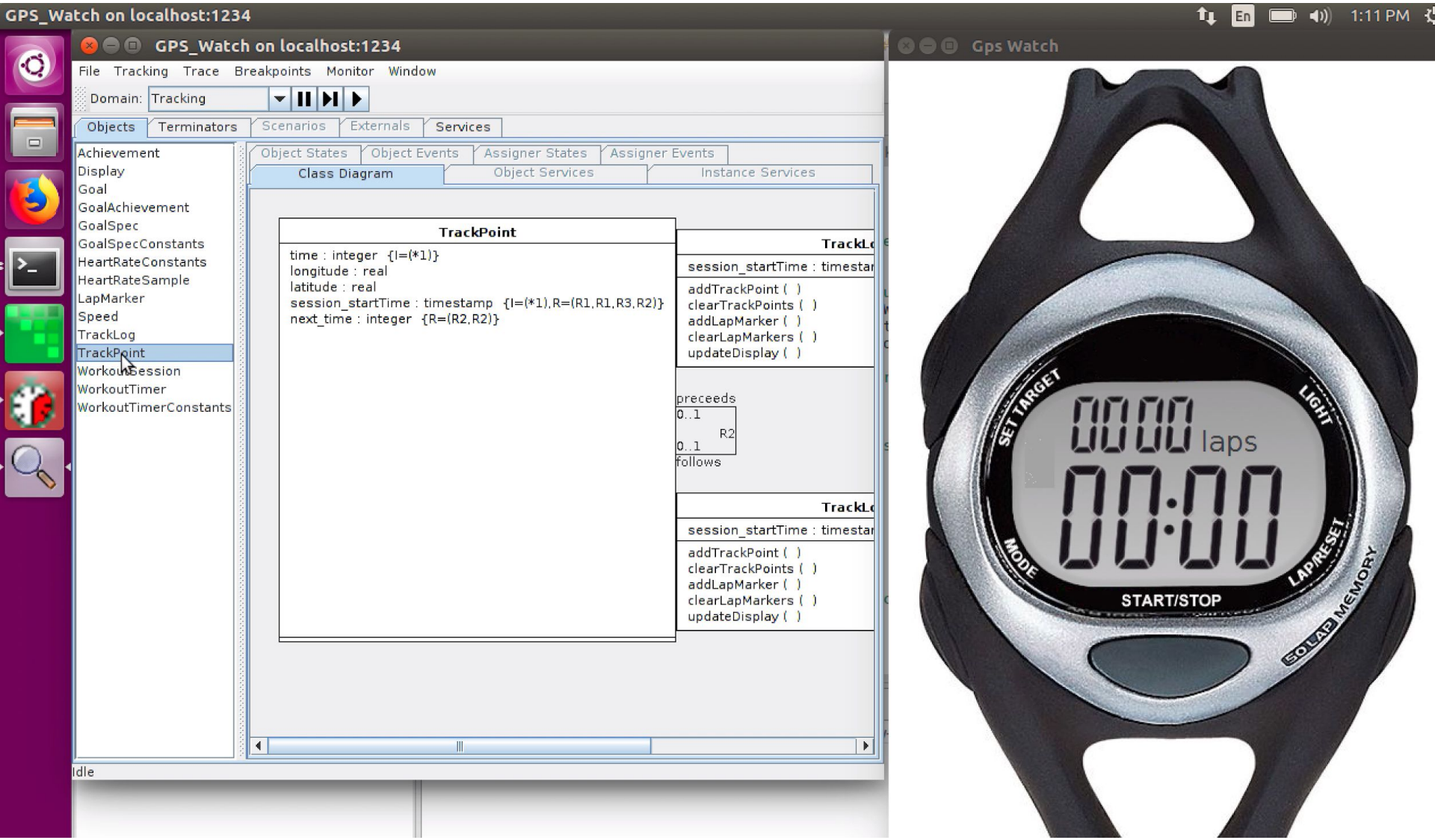
- Verifier
- iUML executor (similar to Verifier)
- Inspector
- penny simulation



Executing with… Verifier

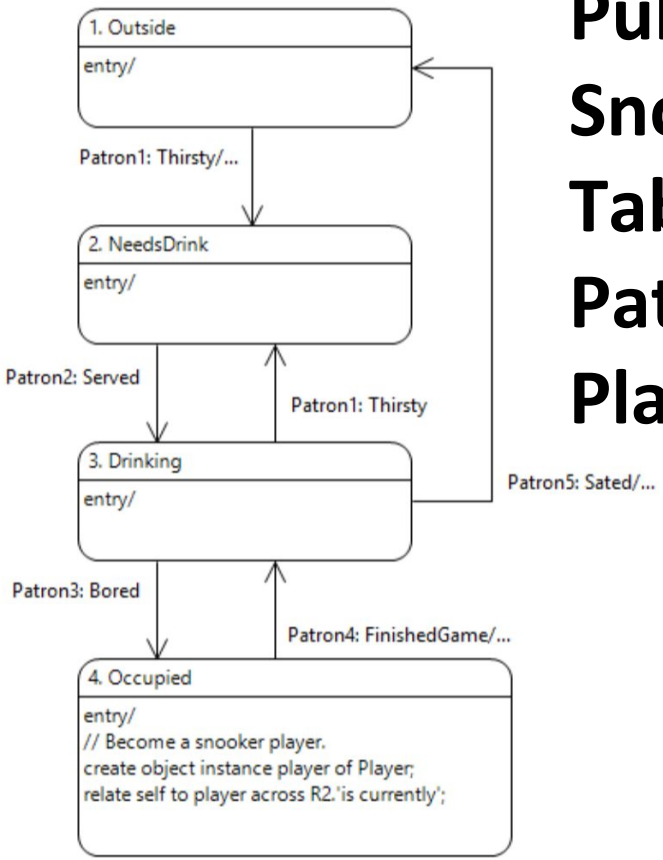


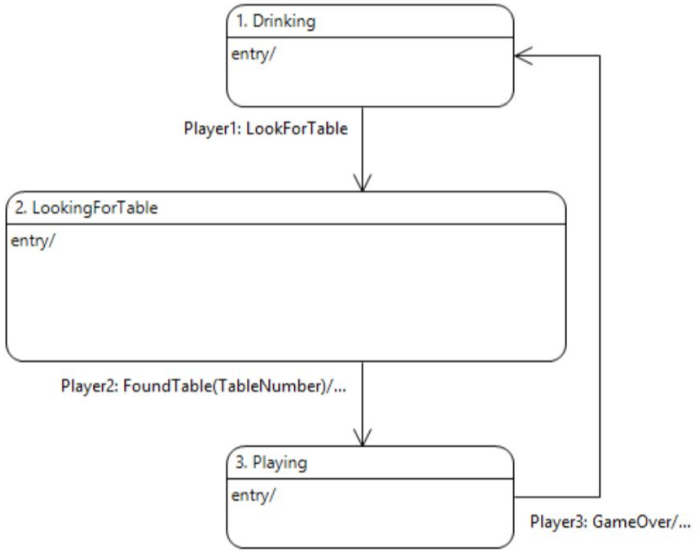
Executing with… Inspector

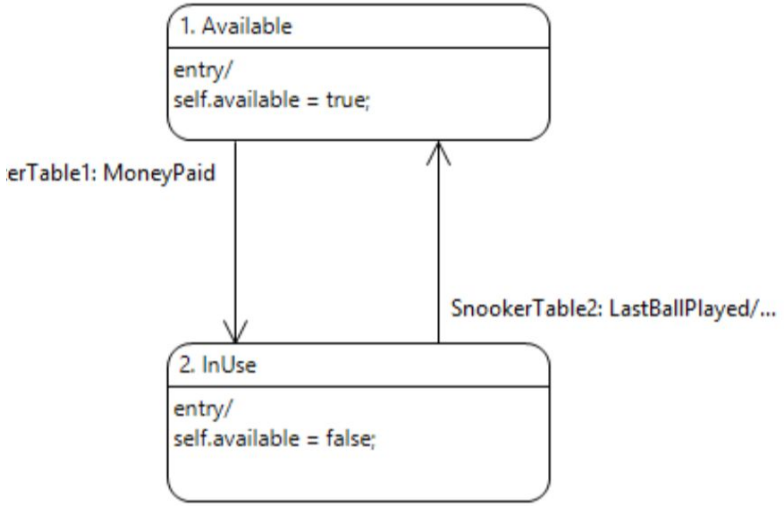


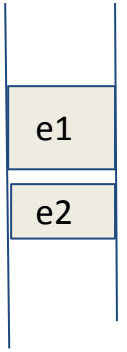
Executing with… Penny Simulation

Pub
Snooker
Table
Patron
Player









Deeper into the S-M Runtime

- how it works…

…just like the simulation.

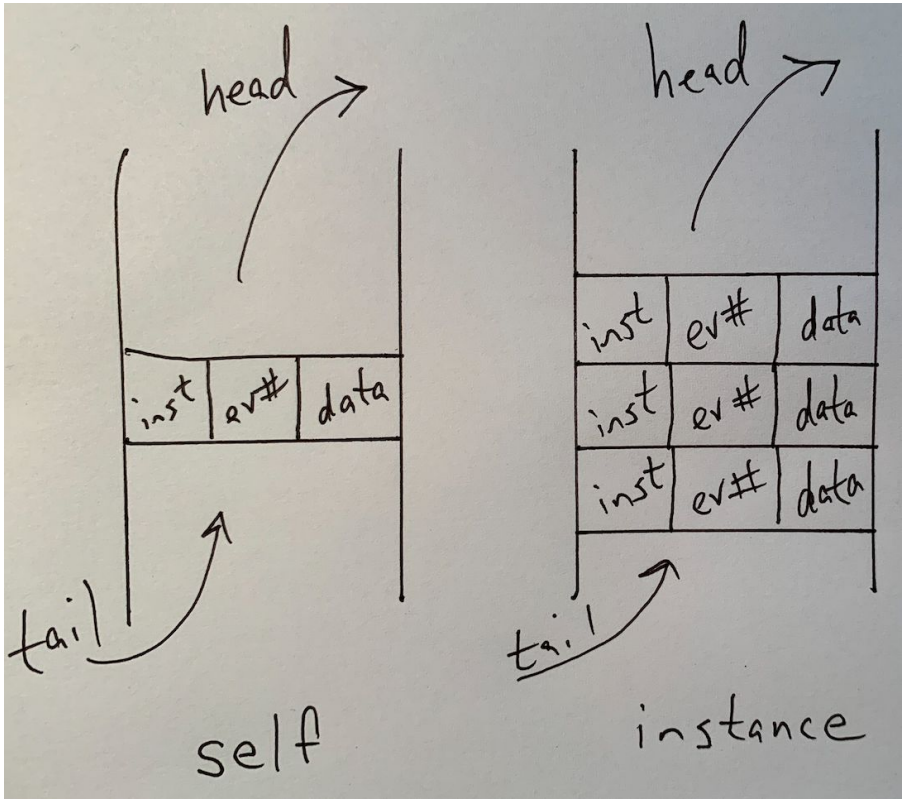
Classes and Instances, State Models and State Machines

- Instances of classes are managed in lists.
- Instances of active classes maintain a *current_state* into the state model defined for the class.

next state = current_state + event

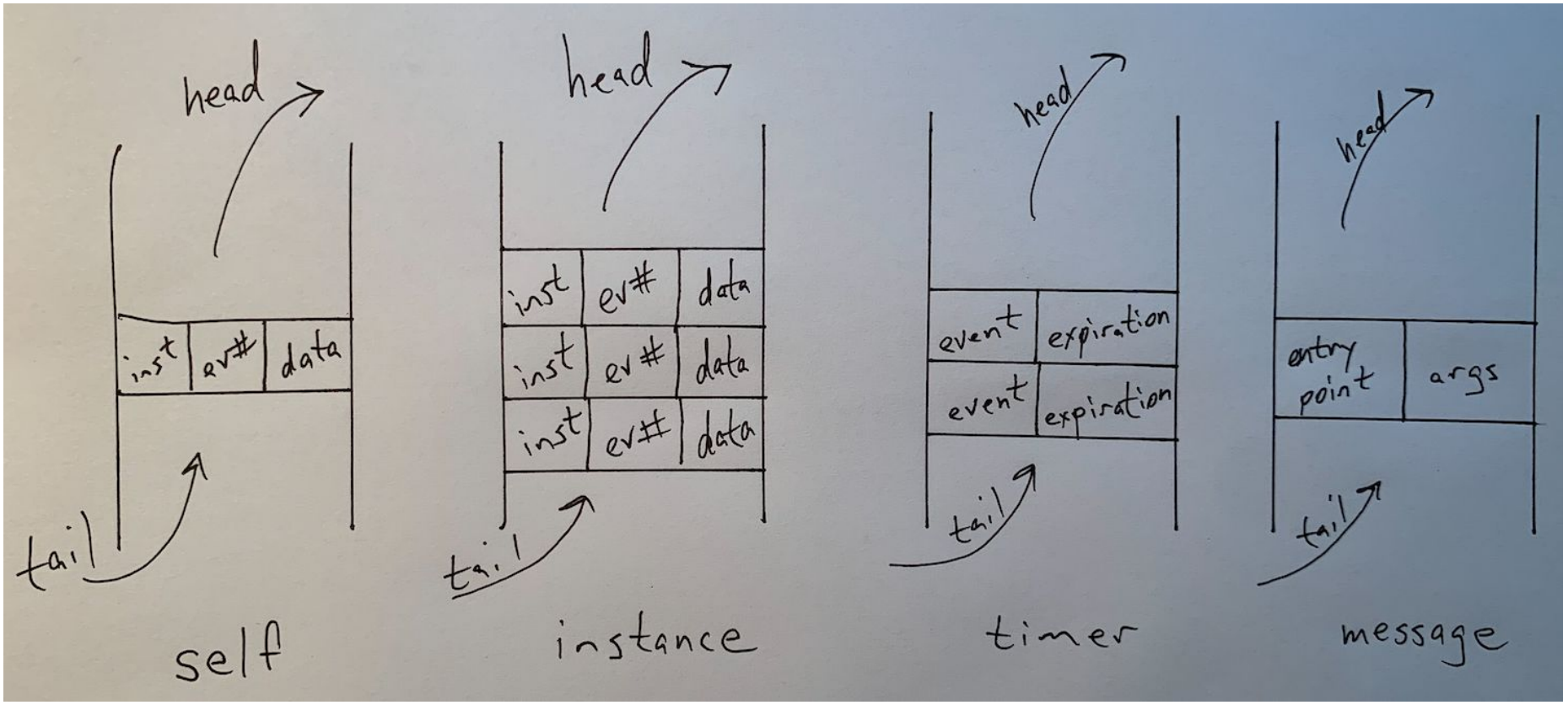
Queue: Event Queue

- head, tail, next, instance, event number



4 Queues: Events, Timers, Msgs

- head, tail, next, instance, event number



Strength: Behavior

Sequencing Observability

- strength of the S-M event dispatcher WRT behavior sequencing
  - examples of instrumentation for tracing
  - IBM fiber channel control unit
  - BAS "can't happen" war story

Weakness: Observability

- exposure of the S-M event dispatcher WRT side channel cryptanalysis
- growth of shared resource (cloud) computing

Strength: Centrality!

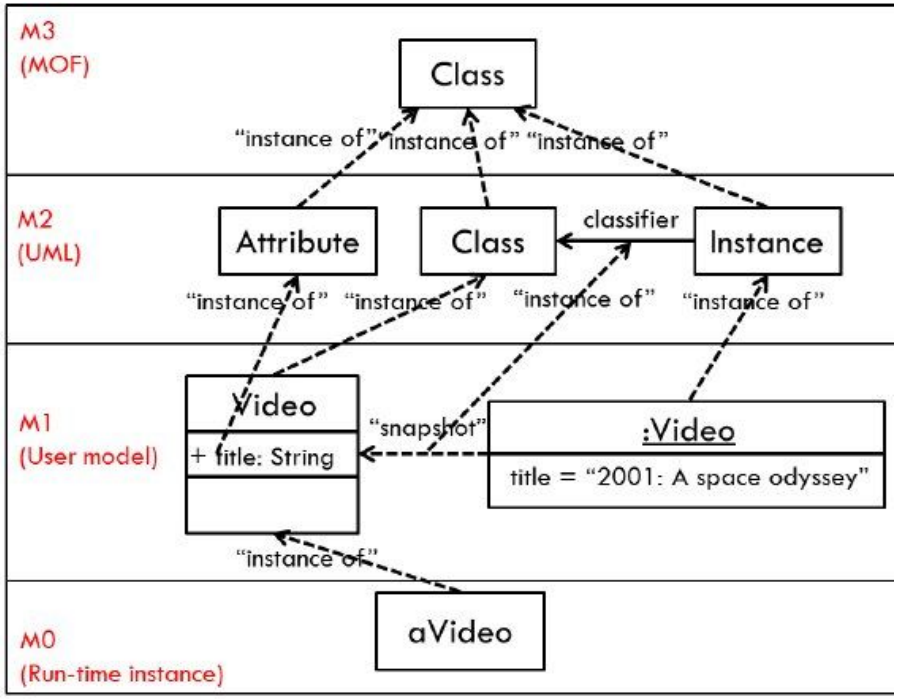
- The runtime provides a central point to measure energy consumption and take action to counteract information leakage.

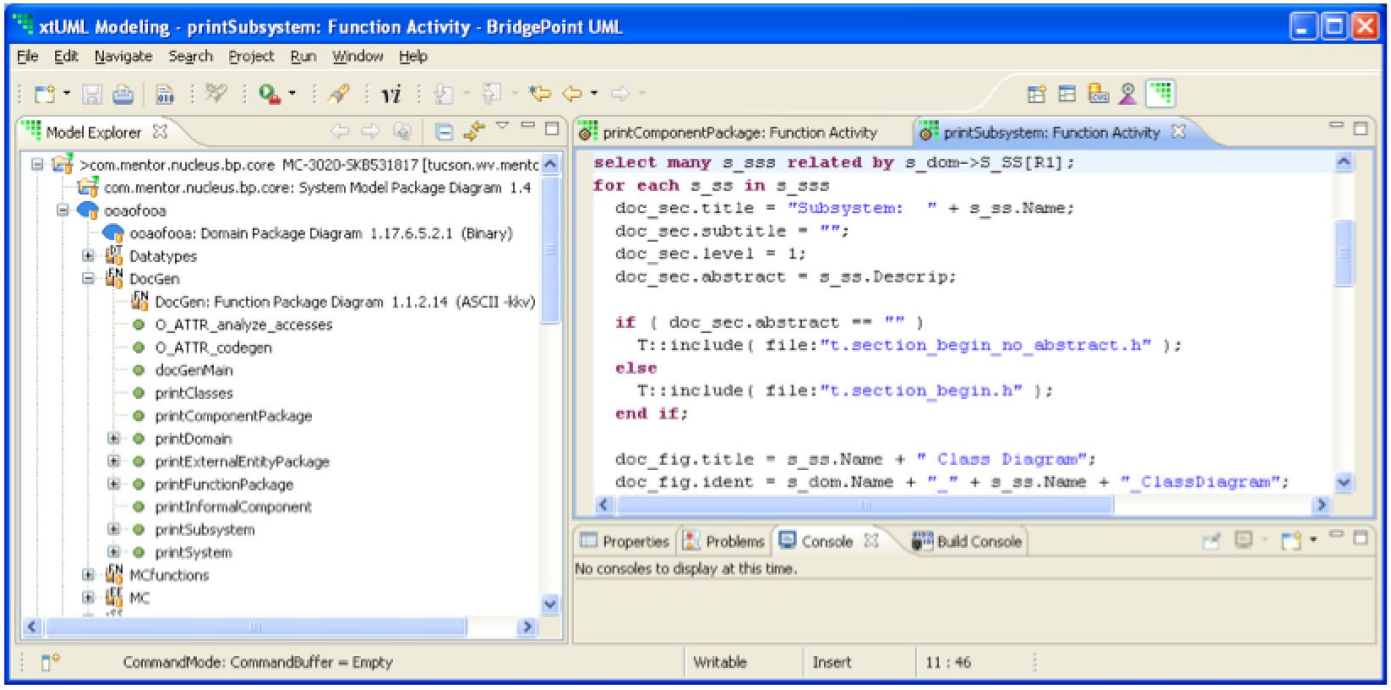
Distinctives of a "model-based model compiler"

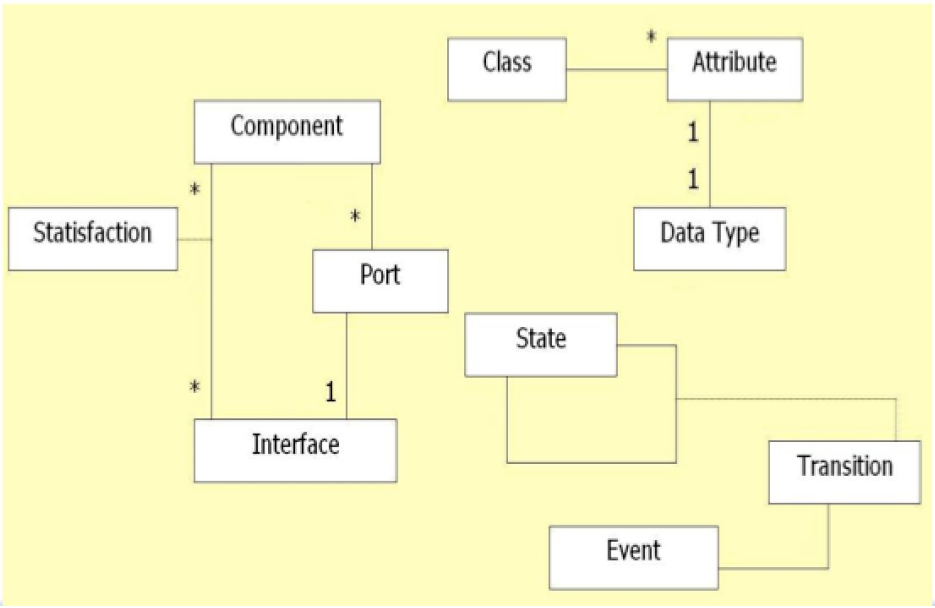
- OOA MM instance loading
- thus OOA of OOA is key subsystem
- templated serialization of text ('model2text')
- otherwise model compiler is a normal S-M model

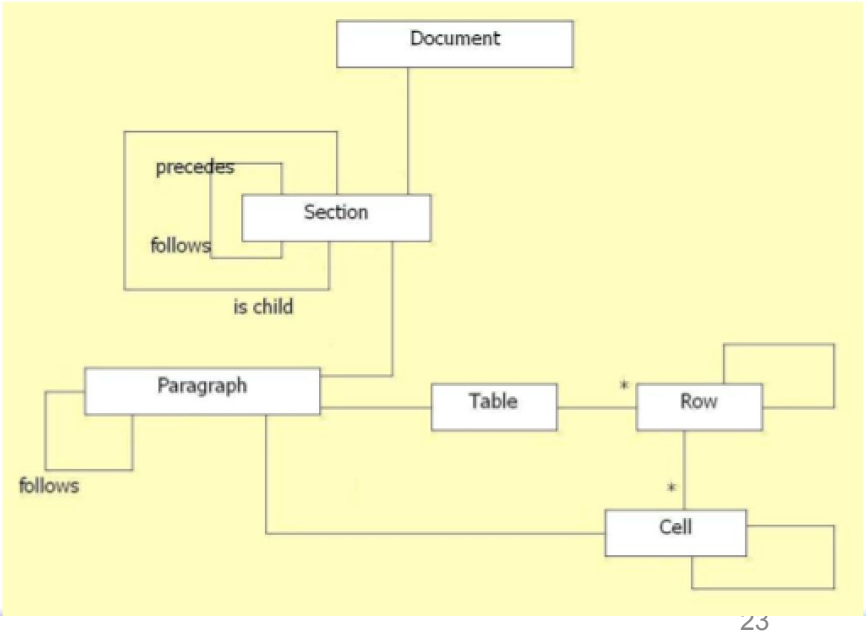
Which (Meta) Models Involved?

- application model
- OOA (meta) model
- model compiler model
- target (meta) model



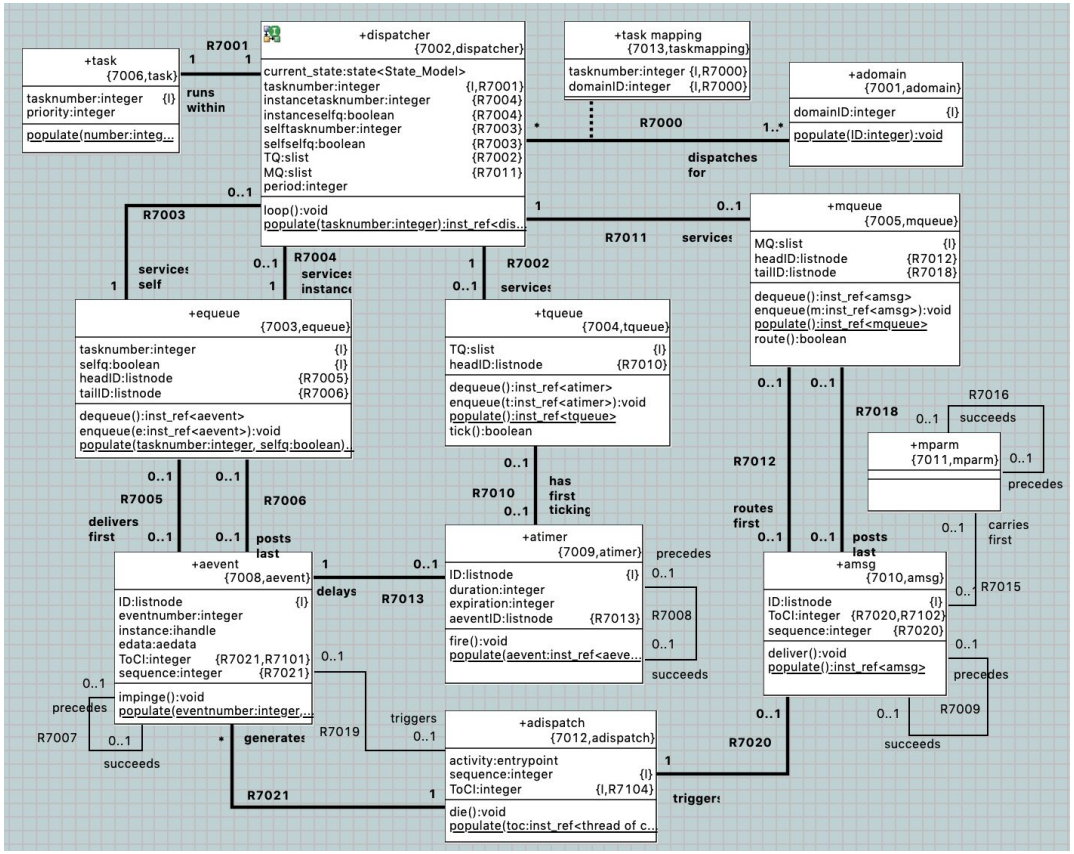

xtUML Modeling - printSubsystem: Function Activity - BridgePoint UML
File Edit Navigate Search Project Run Window Help
Model Explorer
>com.mentor.nucleus.bp.core MC-3020-SKB531817 [tucson.wv.mentc
com.mentor.nucleus.bp.core: System Model Package Diagram 1.4
ooaofooa
ooaofooa: Domain Package Diagram 1.17.6.5.2.1 (Binary)
Datatypes
DocGen
DocGen: Function Package Diagram 1.1.2.14 (ASCII -kkv)
O_ATTR_analyze_accesses
O_ATTR_codegen
docGenMain
printClasses
printComponentPackage
printDomain
printExternalEntityPackage
printFunctionPackage
printInformalComponent
printSubsystem
printSystem
MCfunctions
MC
printComponentPackage: Function Activity
printSubsystem: Function Activity
select many s_sss related by s_dom->S_SS[R1];
for each s_ss in s_sss
doc_sec.title = "Subsystem: " + s_ss.Name;
doc_sec.subtitle = "";
doc_sec.level = 1;
doc_sec.abstract = s_ss.Descrip;
if (doc_sec.abstract == "")
T::include(file:"t.section_begin_no_abstract.h");
else
T::include(file:"t.section_begin.h");
end if;
doc_fig.title = s_ss.Name + " Class Diagram";
doc_fig.ident = s_dom.Name + "_" + s_ss.Name + "_ClassDiagram";
Properties
Problems
Console
Build Console
No consoles to display at this time.
CommandMode: CommandBuffer = Empty
Writable
Insert
11 : 46

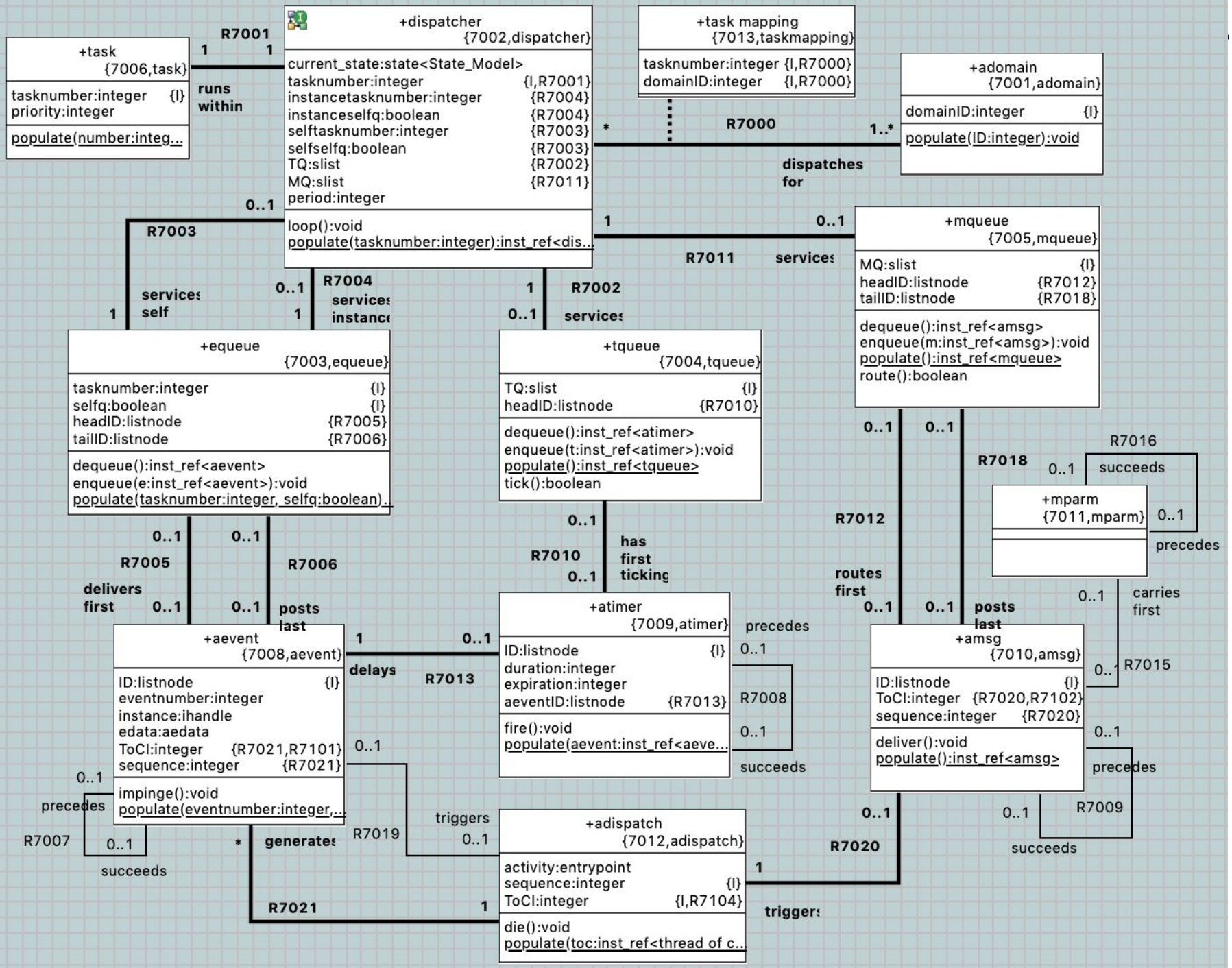

Class
Attribute
Component
Statisfaction
Port
Data Type
State
Interface
Transition
Event
*
1


Document
precedes
Section
follows
is child
Paragraph
Table
Row
Cell
follows
*

Operating Model of S-M Runtime

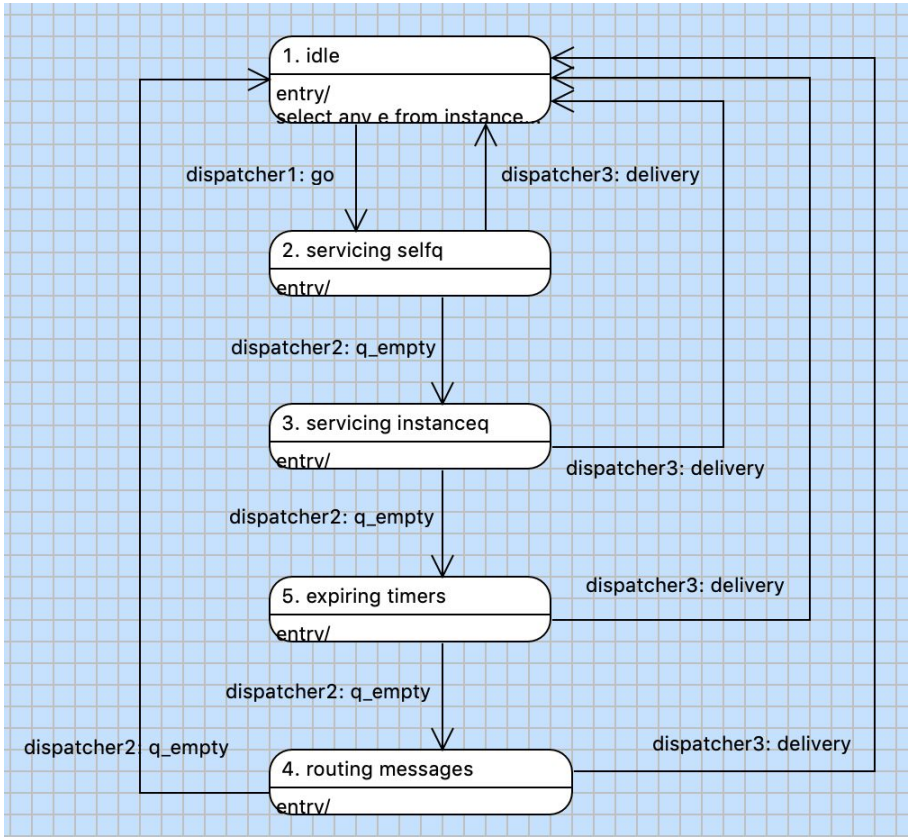
runtime class diagram

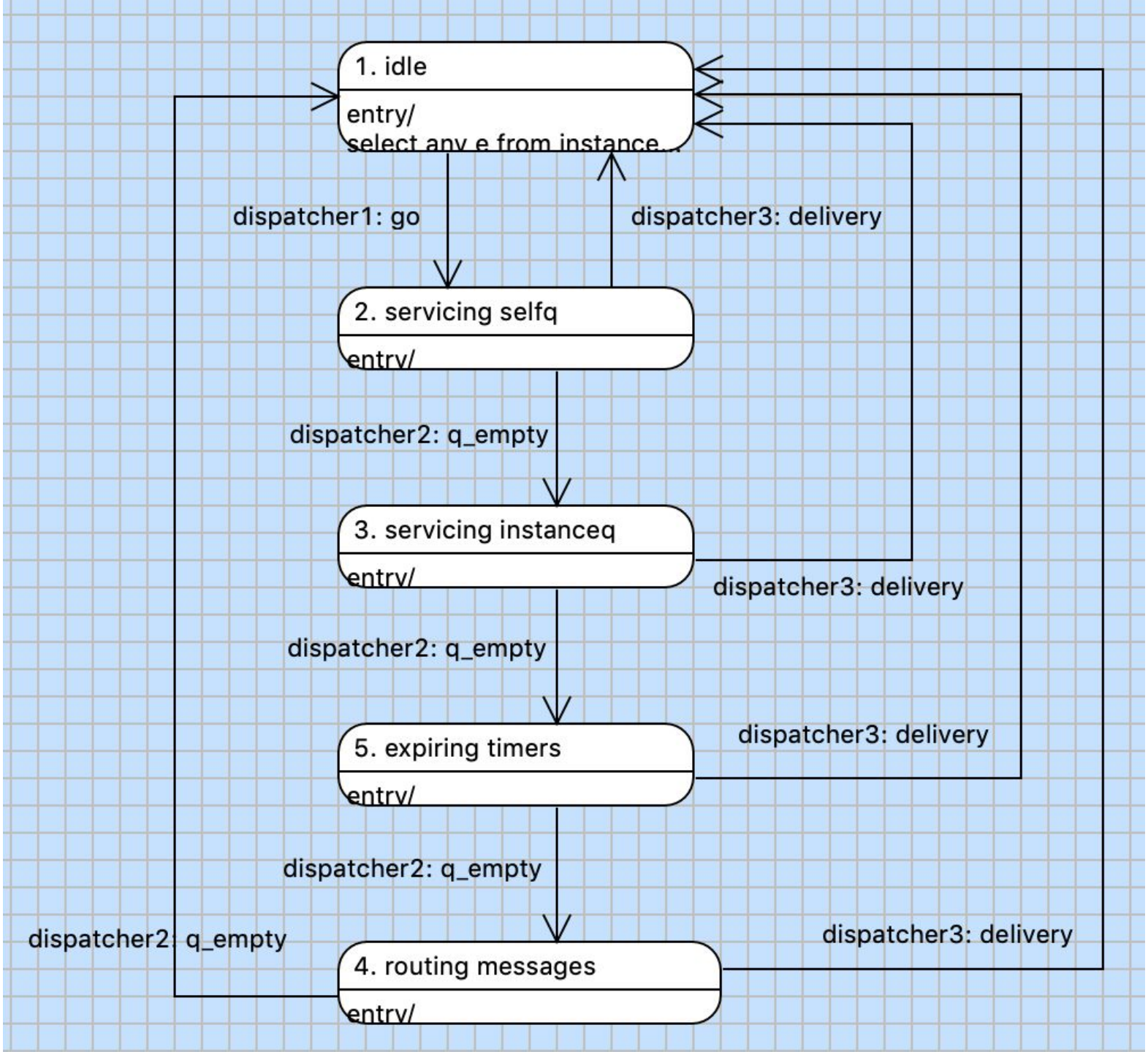



+task
{7006,task}
tasknumber:integer {I}
priority:integer
populate(number:integ...
R7001
runs within
+dispatcher
{7002,dispatcher}
current_state:state<State_Model>
tasknumber:integer {I,R7001}
instancetasknumber:integer {R7004}
instanceselfq:boolean {R7004}
selftasknumber:integer {R7003}
selfselfq:boolean {R7003}
TQ:slist {R7002}
MQ:slist {R7011}
period:integer
loop():void
populate(tasknumber:integer):inst_ref<dis...
+task mapping
{7013,taskmapping}
tasknumber:integer {I,R7000}
domainID:integer {I,R7000}
R7000
+adomain
{7001,adomain}
domainID:integer {I}
populate(ID:integer):void
dispatches for
R7003
services self
R7004
services instance
R7002
services
R7011
services
+mqueue
{7005,mqueue}
MQ:slist {I}
headID:listnode {R7012}
tailID:listnode {R7018}
dequeue():inst_ref<amsg>
enqueue(m:inst_ref<amsg>):void
populate():inst_ref<mqueue>
route():boolean
+equeue
{7003,equeue}
tasknumber:integer {I}
selfq:boolean {I}
headID:listnode {R7005}
tailID:listnode {R7006}
dequeue():inst_ref<aevent>
enqueue(e:inst_ref<aevent>):void
populate(tasknumber:integer, selfq:boolean)...
+tqueue
{7004,tqueue}
TQ:slist {I}
headID:listnode {R7010}
dequeue():inst_ref<atimer>
enqueue(t:inst_ref<atimer>):void
populate():inst_ref<tqueue>
tick():boolean
R7005
delivers first
R7006
posts last
R7010
has first ticking
R7012
routes first
R7018
posts last
R7016
succeeds
+mparm
{7011,mparm}
precedes
carries first
+aevent
{7008,aevent}
ID:listnode {I}
eventnumber:integer
instance:ihandle
edata:aedata
ToCI:integer {R7021,R7101}
sequence:integer {R7021}
impinge():void
populate(eventnumber:integer,...
R7007
precedes
succeeds
delays
R7013
+atimer
{7009,atimer}
ID:listnode {I}
duration:integer
expiration:integer
aeventID:listnode {R7013}
fire():void
populate(aevent:inst_ref<aeve...
precedes
R7008
succeeds
+amsg
{7010,amsg}
ID:listnode {I}
ToCI:integer {R7020,R7102}
sequence:integer {R7020}
deliver():void
populate():inst_ref<amsg>
R7015
precedes
R7009
succeeds
generates
R7021
triggers
R7019
+adispatch
{7012,adispatch}
activity:entrypoint
sequence:integer {I}
ToCI:integer {I,R7104}
die():void
populate(toc:inst_ref<thread of c...
R7020
triggers

Operating Model of S-M Runtime

dispatcher state model




1. idle
entry/
select any e from instance...
dispatcher1: go
dispatcher3: delivery
2. servicing selfq
entry/
dispatcher2: q_empty
3. servicing instanceq
entry/
dispatcher3: delivery
dispatcher2: q_empty
5. expiring timers
entry/
dispatcher3: delivery
dispatcher2: q_empty
dispatcher2: q_empty
4. routing messages
entry/
dispatcher3: delivery

```
select one iq related by self->equeue[R7004.'services instance'];
select one selfq related by self->equeue[R7003.'services self'];
select one tq related by self->tqueue[R7002.'services'];
select one mq related by self->mqueue[R7011.'services'];

while ( true )
  // Fully service self queue before instance queue.
  e = selfq.dequeue();
  if ( empty e )
    e = iq.dequeue();
  end if;
  if ( not_empty e )
    // Impinge event against target state machine.
    e.impinge();
  else
    // Timers expire queueing associated events.
    if ( not tq.tick() )
      // Messages (signals) from outside are processed.
      b = mq.route();
    end if;
  end if;
end while;
```

Operating Model of S-M Runtime

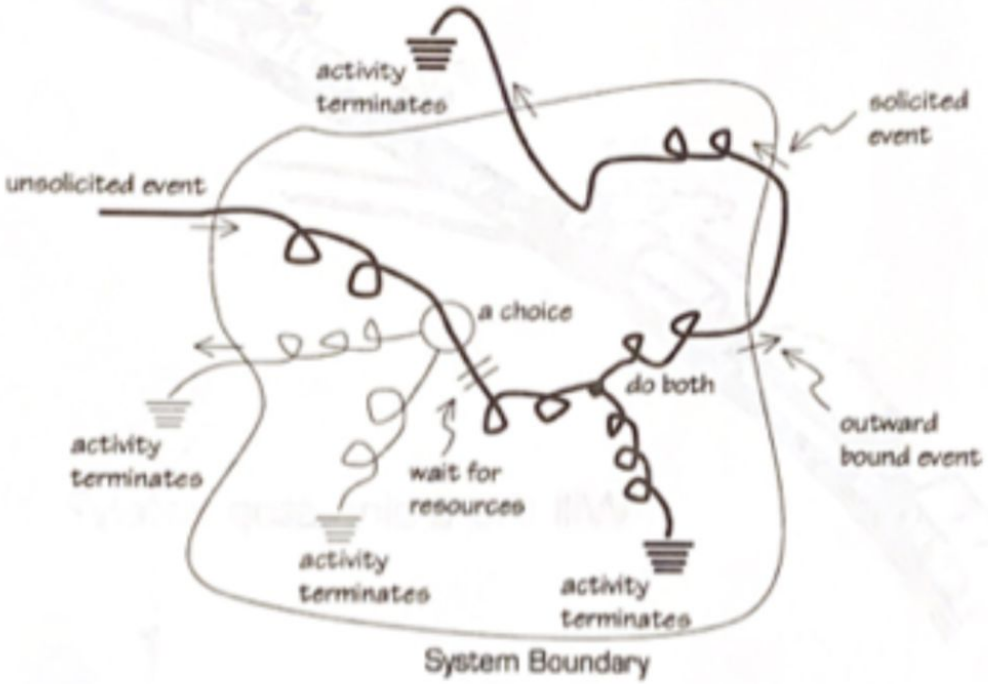
- time rules
- event rules
- implemented in 4 queues:
  - self events
  - instance events
  - timers
  - messages

Extended Model of S-M Runtime

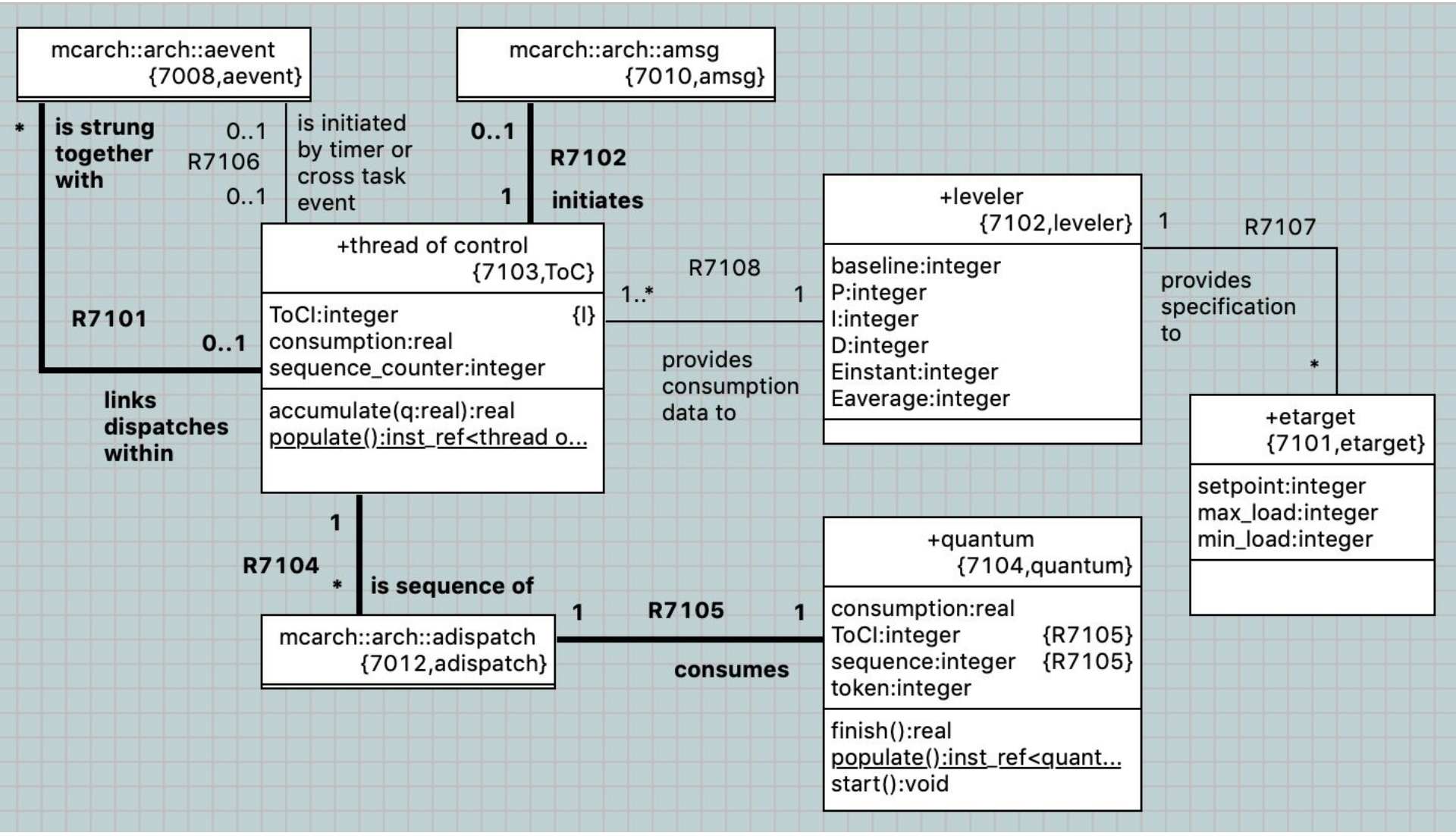
- Thread of Control
- energy quantum
- PID

Thread of Control

A thread of control is the sequence of actions and events that occurs in response to the arrival of a particular unsolicited event when the system is in a particular state.



Extended Model of S-M Runtime

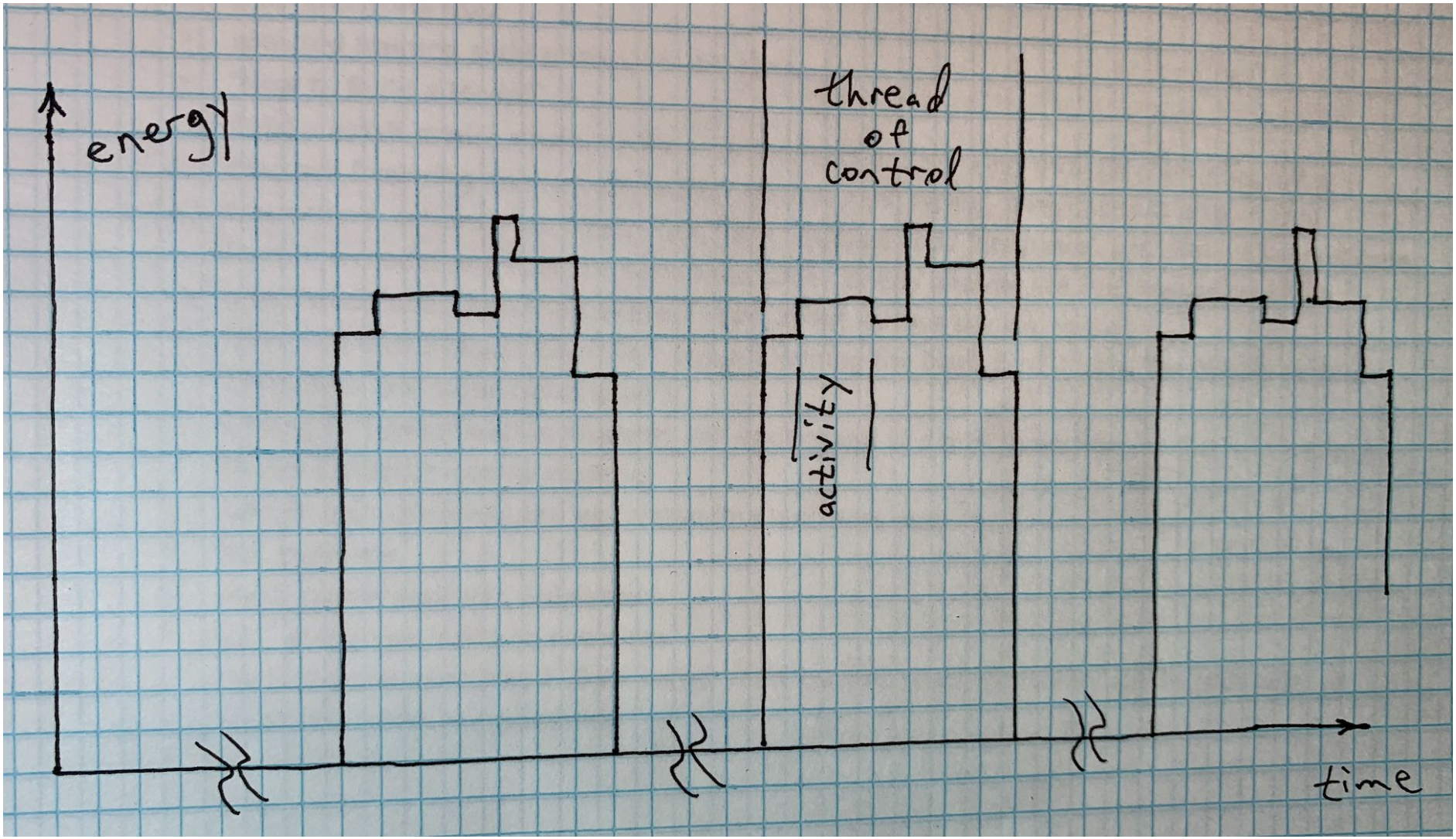


S-M Resource Usage Analyzed

- energy measurement (quantum, ToC)

Each activity dispatch links to a quantum of energy measured from its execution.
The quata are summed into the thread of control.
Patterns at the activity level and/or at the ToC level can be measured, analyzed and counteracted.

Square Wave Time Usage (GPSW)



Object/State Output Represented as Letters/Numbers



Another experiment involves emitting characters correlated with specific states from the threads of control. The pattern will be visible as repeating characters.

After injecting jitter and feedback-balanced threads of control, compare the output stream of characters and analyze (compressibility, etc).

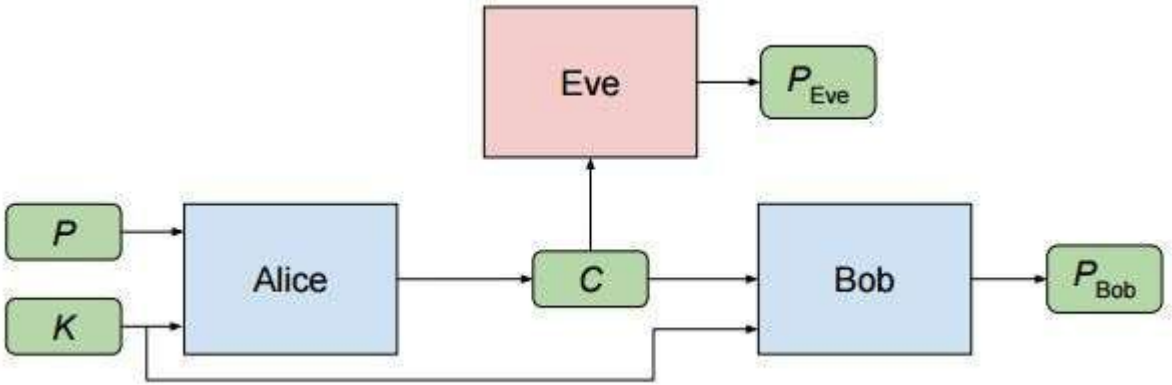
Measurement → Control

- Once we can measure our energy usage, we can balance and shape it.
- self-leveling
- cooperative leveling



Energy Balancing Techniques

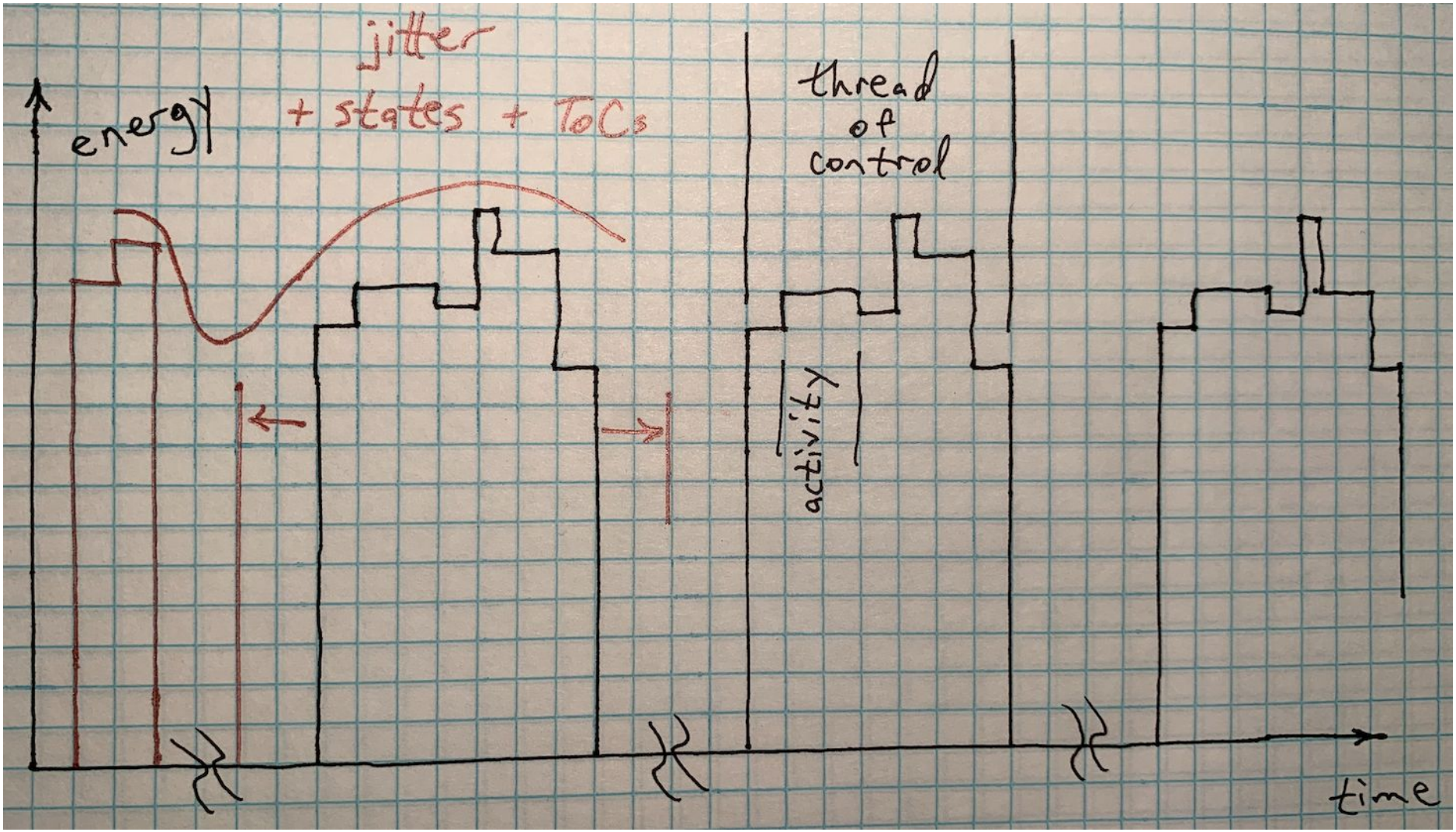
- Runtime energy direct feedback.
- Self injection of benign messages with known threads of control.
- Outside injection of benign messages with known threads of control.



Square Wave Time Usage (GPSW)

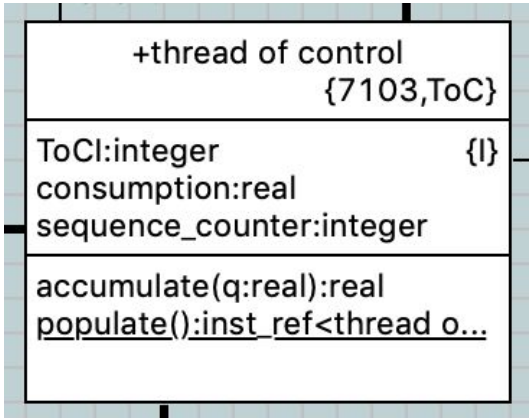
- By injecting states within a thread of control the energy boundaries can be widened.
- By injecting additional threads of control the periodicity can be reduced.
- The quantities of additional energy desired are adjustable.
- Feedback provides opportunity to ramp up/down the injected energy requirements.

Square Wave Time Usage (GPSW)



Steganographic Techniques

- simplistic steganographic translation of multiple models together into the same build
- more purposeful steganographic translation of the same model with variation of the states and activities
- Thread of Control Identifier is key to differentiating legit and benign action dispatches.



Demo

- GPSW unobscured
  - time profile
  - energy profile
  - state, message and energy patterns
- GPSW obscured
  - time profile
  - energy profile
  - state, message and energy patterns

Future

- finish the runtime leveler
- identify protocol for ToCI assignment
- integrate runtime into upcoming model compiler architectures

Review

- elucidating the fundamentals of Shlaer-Mellor runtime
- introducing security extensions to the core of the runtime model
- obscuring state sequencing
- obscuring message sequencing
- obscuring/leveling resource (energy) utilization