VisualAivika 9

TO MODEL IN FUNCTIONAL STYLE

BY DAVID E. SOROKIN <DAVSOR@MAIL.RU>

HTTPS://VISUALAIVIKA.RU

VisualAivika 9
To Model in Functional Style

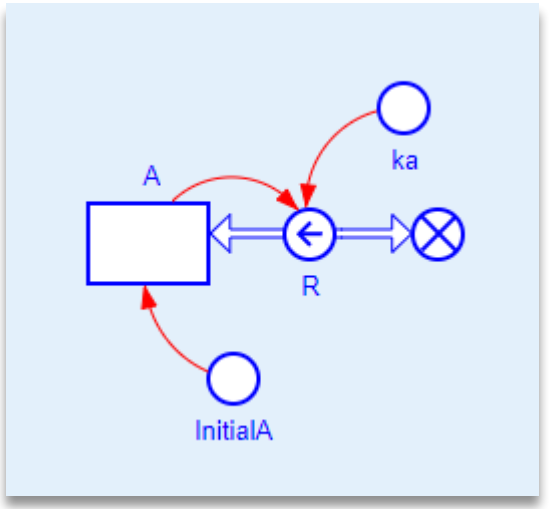
- **System Dynamics**
  - **ODE System** (State Machine, Runge-Kutta Method)
  - Example
    - A = integ(R, InitialA);
      InitialA = 1000;
      R = ka * A;
      ka = 0.07;
- **Discrete Event Simulation**
  - **Queueing System** (Transacts, Queues, Resources, Block Computations)
  - Example
    - P3 = Block.preempt(Prof) >>> P4;
      P4 = Block.advance(exponential(200)) >>> P5;
      P5 = Block.return(Prof) >>> Block.terminate;
      Prof = Facility.create();

VisualAivika 9
System Dynamics

Example

```
A = integ(R, InitialA);
InitialA = 1000;
R = ka * A;
ka = 0.07;
```

Stock and Flow Map



Equations and the diagram are one whole

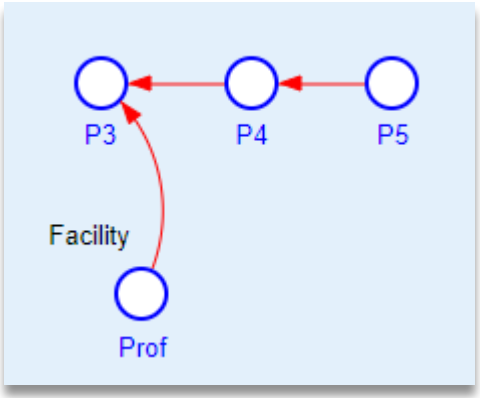
VisualAivika resolves the system of equations and builds on-the-fly the computational graph, by which it generates the C# code for numerical integrating with help of state machine

VisualAivika 9
Queueing System

Example

```
P3 = Block.preempt(Prof) >>> P4;
P4 = Block.advance(exponential(200)) >>> P5;
P5 = Block.return(Prof) >>> Block.terminate;
Prof = Facility.create();
```

Transact Processing



Blocks P3, P4 and P5 are computations that are combined with help of composition

VisualAivika builds the computational graph and generates the C# code to run these computations within the state machine

VisualAivika 9
To Model in Functional Style

Both the ODE system and the queueing system are transformed to the same state machine

This state machine is dispatched by the event queue, which allows combining the both paradigms within the same simulation model

VisualAivika 9 To Model in Functional Style

THE NUMERICAL INTEGRATION OF THE SYSTEM OF EQUATIONS IS NATURALLY EXPRESSED AS A STATE MACHINE

BUT WHAT INDEED ARE BLOCK COMPUTATIONS?

VisualAivika 9
To Model in Functional Style

The block computation is the Kleisli arrow

The arrow is built on the continuation monad

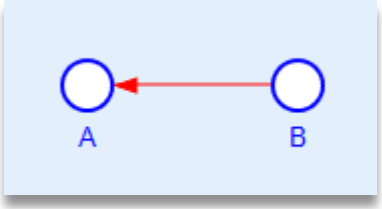
The continuations are bound up with the event queue

But **you can use all this** without deep understating of functional programming!

VisualAivika 9

Block composition is just an arrow composition (in terms of Haskell)

Block Composition



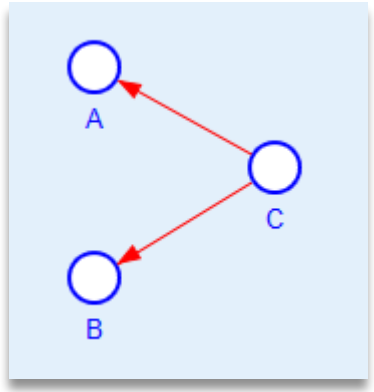
C = A >>> B;

Here we create a new block C, where every transact is initially processed by block A and then by block B

VisualAivika 9

When multiplexing blocks, we just use the shared instance of the same block computation

Multiplexing Block Computations



A = Block.identity >>> C;
B = Block.identity >>> C;

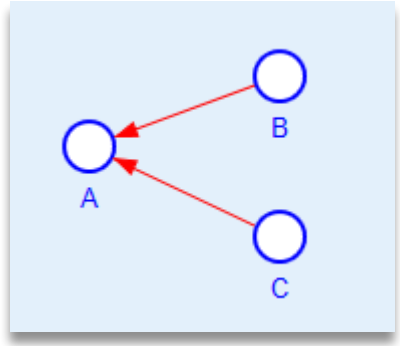
If the transact is processed by A, then it is processed by C

Again, if the transact is processed by B, then it is processed by C too

VisualAivika 9

To select the next block, we can test the condition for every transact

Selecting Next Blocks



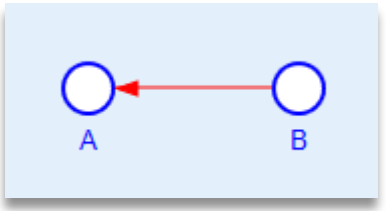
A = Block.**select**(if transact.Count > 7
then B else C);

Here if some Count attribute value of the transact is greater than 7, then the transact goes to block B; otherwise, the transact is processed by block C

VisualAivika 9

The delay is a block computation that holds every transact for the specified time interval

Delaying Transacts



A = Block.**advance**(random(1, 10)) >>> B;

Here we hold every transact for
the random delay between 1 and 10.
Then the transact is processed by block B

VisualAivika 9
To Model in Functional Style

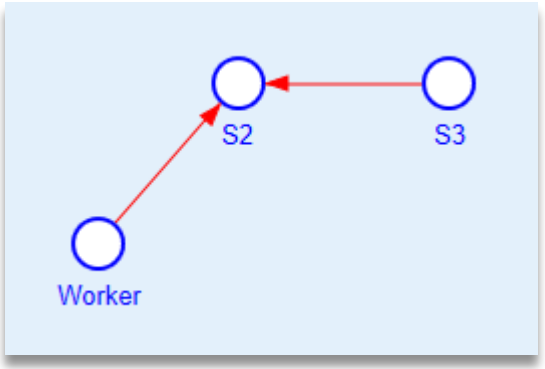
BUT THEN WHAT ABOUT RESOURCES AND QUEUES?

VisualAivika 9

The resources and queues are entities, which can be used within block computations

Here the role of such entities is secondary unlike block computations

Handling Resources and Queues



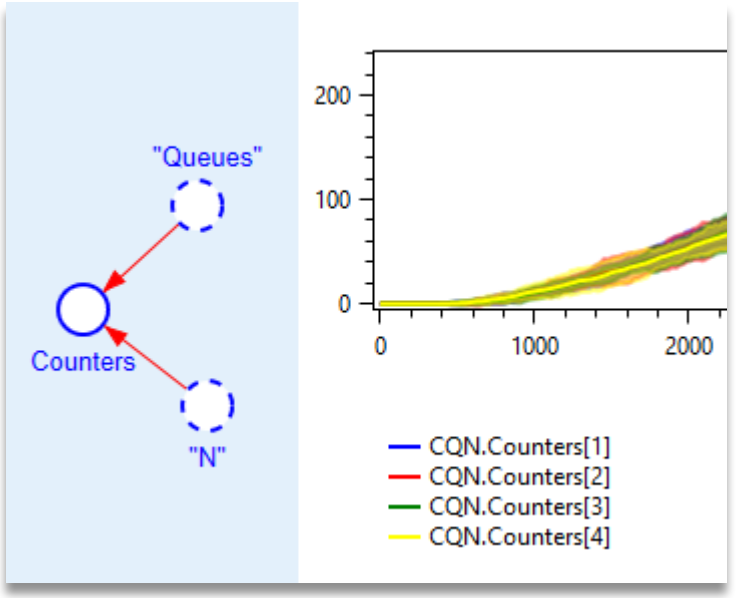

```
Worker = Facility.create();

S2 = Block.seize(Worker) >>>
        Block.advance(random(5, 11)) >>>
        Block.release(Worker) >>> S3;
```

Here we seize the resource, delay randomly the transact and then release the resource before switching to block S3

VisualAivika 9

The resources and queues gather the statistics,
which can be displayed on the charts

Displaying Resource and Queue Statistics



Counters =
  [Queue.**enqueueCount**(Queues[i]) | i <- 1..N];

Here we display the array of counters
(the enqueue counts) for the array of queues

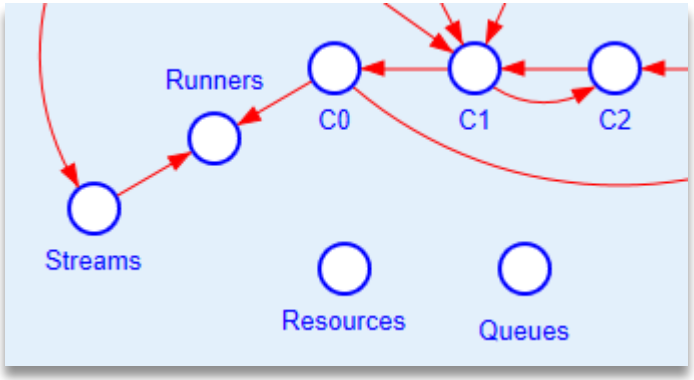
VisualAivika 9
To Model in Functional Style

BUT HOW TO INITIATE THE TRANSACT PROCESSING BY BLOCK COMPUTATIONS?

VisualAivika 9

The transacts are born from the streams of arrival events

Creating Stream Computations



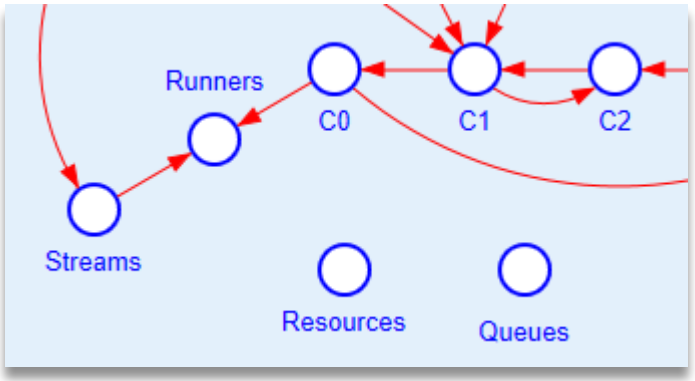

```
Streams =
  [ Stream.take(Stream.randomInt(0, 0),
      Initial_Job_Count_Per_Tandem) | i <- 1..N ]
```

Initially, we have to create a stream of arrival events. Actually, here we create an array of streams by the specified number of events

VisualAivika 9

We take the input stream of arrival events and then begin creating transacts to be processed by the specified block computation

Generating Transacts




```
Runners =
  [ do! Block.runByStream(Streams[i], C0[i]) |
      i <- 1..N ];
```

This construct creates transacts by the specified array of streams and then begins processing such transacts by the specified block computations (array C0)

VisualAivika 9
To Model in Functional Style

WHAT TO DO IF THERE ARE NO READY-TO-USE BLOCK COMPUTATIONS?

VisualAivika 9
To Model in Functional Style

You can **extend your simulation models** with help of the arbitrary .NET programming code (F# and C#) based on IronAivika [https://gitflic.ru/project/dsorokin/iron-aivika]

Example

```
F =
    [<assembly("lib/DelayBlockFunction.dll")>]
    [<class(VisualAivika.Demo.DelayBlockFunction)>]
    extern fun (A: Block): Block;

Y1 = F(Block.advance(10) >>> Block.terminate);
```

VisualAivika 9
To Model in Functional Style

Moreover, you **can export your simulation models** as .NET applications

Such exported applications will not depend on VisualAivika IDE. You will control how and when to build and run these applications

VisualAivika 9
To Model in Functional Style

David E. Sorokin (Yoshkar-Ola, Russia)

davsor@mail.ru

HTTPS://visualaivika.ru