Jiří Fišer, Jiří Škvára

J.E. Purkinje University, Ústí nad Labem, Czech Republic

ETOS

discrete event simulation framework

focused on easier team cooperation

this contribution is supported by grant from

company *Severočeské doly a.s. Chomutov* (http://www.sdas.cz)

SimPy

[phase 1] student participation on project (2011)

- several simulations of *e-car* (traffic, refueling, parking&shopping)

[phase 2: *enthusiasm***]**

selecting a usable tool for **discrete event simulation**

SimPy

- **Python** programming language (modern, simple, effective)
- clear design and reasonable complexity
- open source & relatively good documentation

SimPy – problems

[phase 3, *disillusion*] disadvantages and limits of *SimPy*

- SimPy uses coroutines based on **generators** by providing a central dispatcher on top of active coroutines – *trampoline* $\Longrightarrow$ dispatch requests are propagated by **re-yielding**
- **nested coroutines are not directly supported** $\Longrightarrow$ **huge single level coroutines**
- **the separation of developer's roles is almost impossible**
  - *simulation modelers and specifiers (engineers)*
  - *low level Python programmers*
  - *statisticians*

SimPy – problems II

  - typical *SimPy* design patterns (e.g. reneging) are both extremely error-prone and almost non-reusable
  - problematic post-modifications and extensions (code is not scalable)
- behaviour of *Python* **exceptions** in *SimPy* is undefined or esoteric
  - very difficult debugging (some types of failures do not abort simulation)
  - slightly surprising *GeneratorExit* exception (called in generators after leaving of iterator)

Solution

[phase 4, ~~*panic and hysteria*~~] search for solution

- *using a more abstract formalism together with a high-level framework (if possible in* Python*)*
  - ***DEVS*** (*Discrete Event System Specification*) and *DEVSimPy*

    \- relatively complex formalism, different from SimPy model

    \- almost non-existent documentation (*DEVSimPy*)
- *design and implementation of new framework on top of SimPy*

    \+ utilization of SimPy framework and our know-how

    \+ early involvement of student in the project

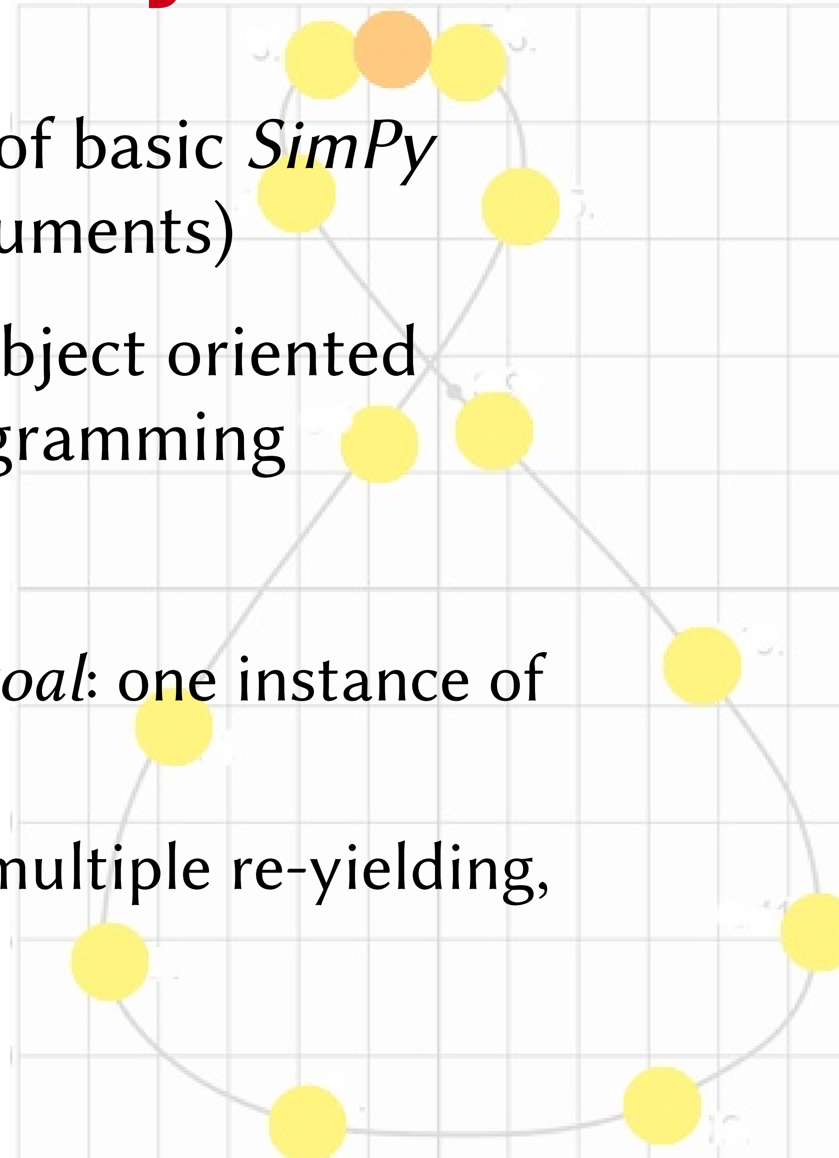
Design objectives

separation of simulation code into (at least)

two levels of abstraction

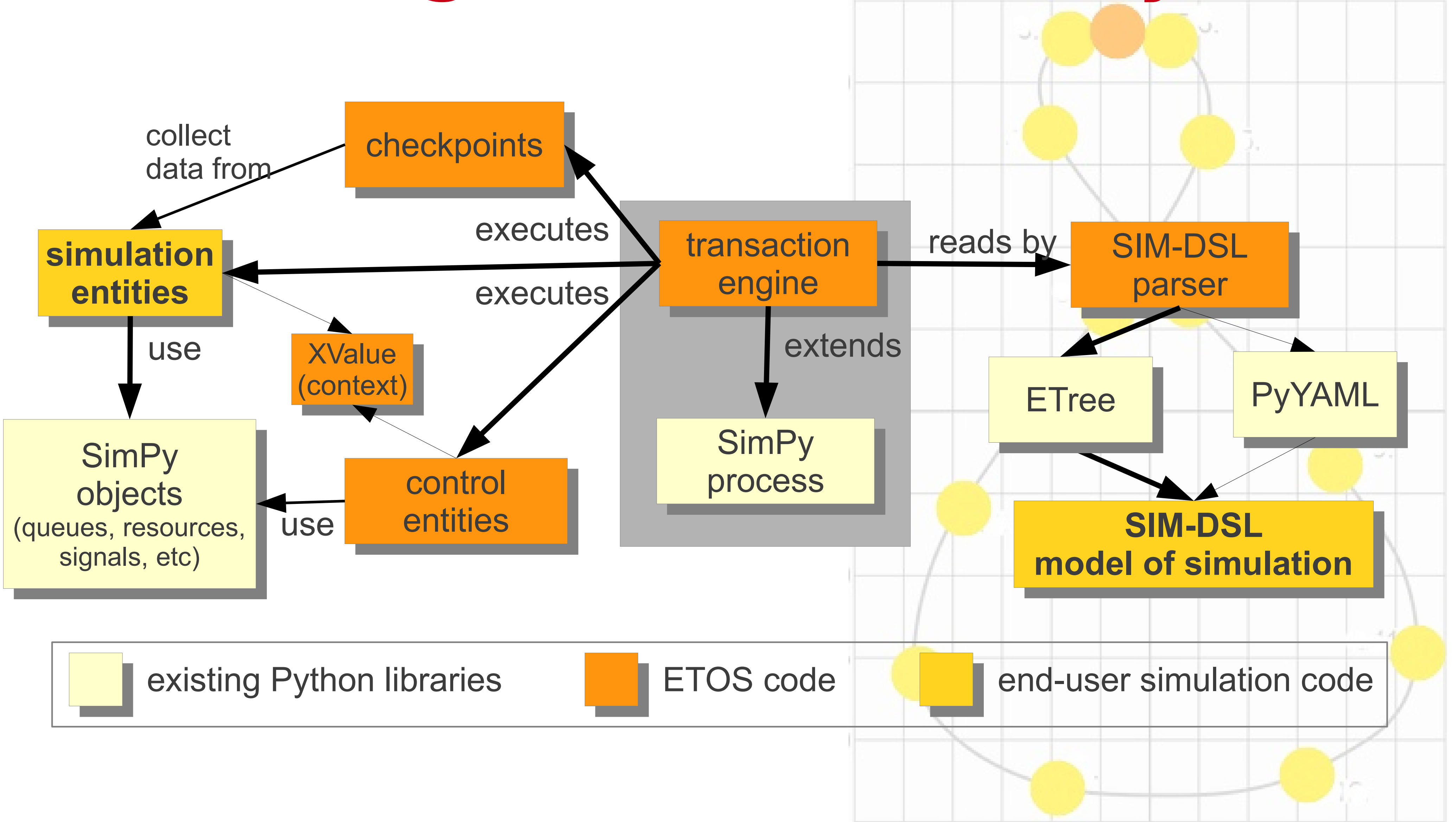
- **declarative description** of parametres and lifetime of simulated objects by high level structured language (e.g. XML)
- **procedural representation** of actions by extended *SimPy* code in the form of generator coroutines
  - the maximal expressivity of declarative code (declarative notation is preferred)
  - *relaxed and extensible* structural notation (elimination of deeply nested „matryoshka“ constructs)
  - the maximal simplicity and clarity of procedural code

Implementation objectives

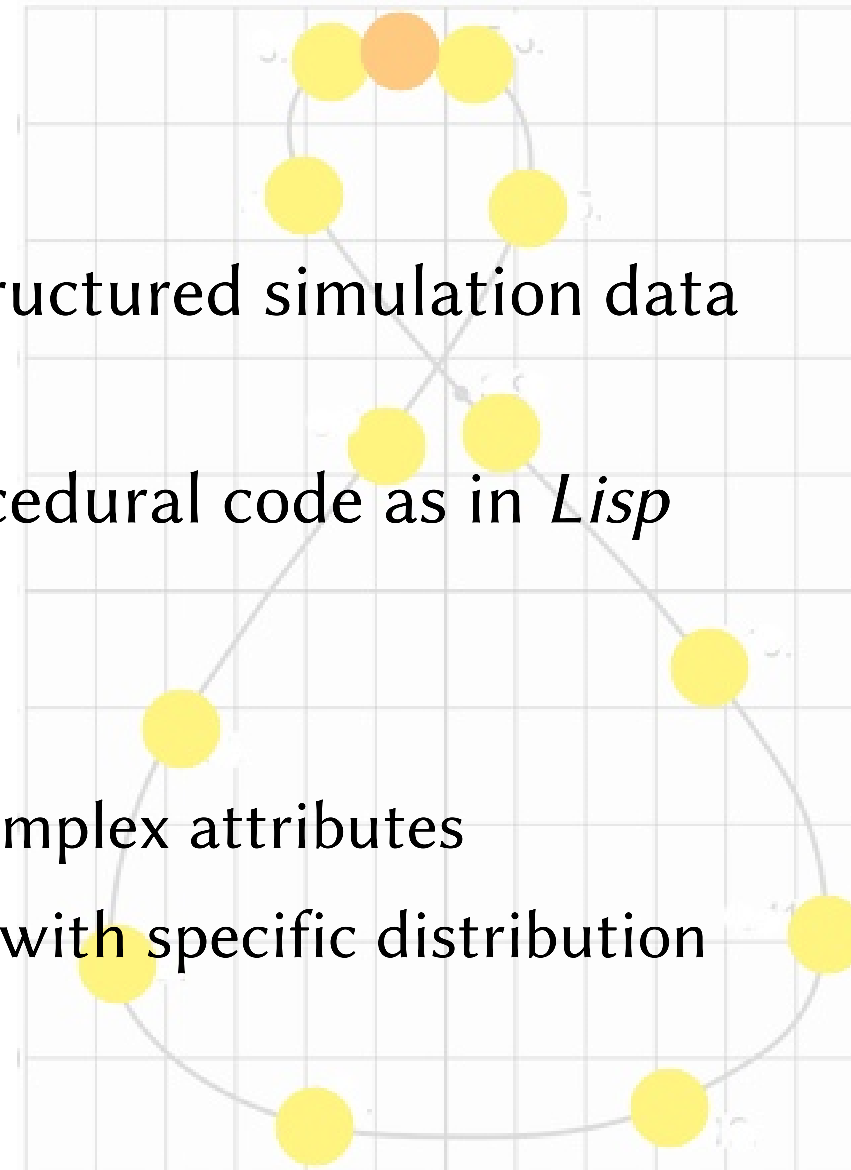


- simplified and integrated support of basic *SimPy* constructs (especially *yielding* arguments)
- re-usability of repetitive code by object oriented inheritance or by *Python* metaprogramming
- time and memory efficiency
    - re-usability of auxiliary objects (*goal*: one instance of auxiliary objects per simulation)
    - optimization of slow operations (multiple re-yielding, creation of coroutines)

Design of ETOS library



SIM-DSL



- **SIM – domain specific language**
- abstract structural notation for structured simulation data and simulation code
- unification of declarative and procedural code as in *Lisp* (homoiconicity)
- SIM-DSL supports:
  - hierarchical tree of nodes with complex attributes
  - specification of random numbers with specific distribution
  - time dependent values
- representation: XML or YAML

(ETOS) Simulation

simulation as a whole (global states)

declarative (SIM-DSL) representation

set of interlinked SIM-DSL documents

procedural (Python) representation

instance of class *Etos.Simulation* derived from class *SimPy.Simulation*

ETOS supports multiple simulation instances which can be executed in multiple threads or processes (e.g. in simple cluster based on Python *multiprocessing* package)

Actor

simulated object with attributes during its lifetime

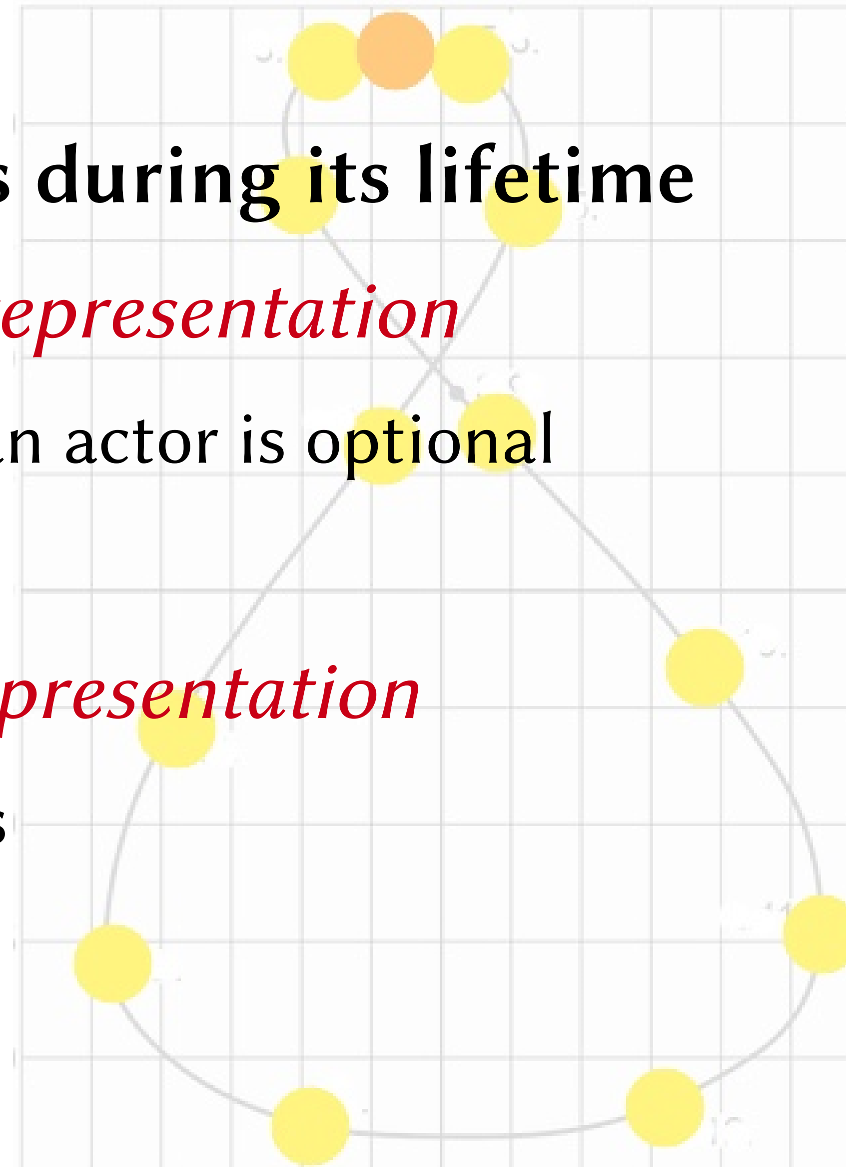
declarative (SIM-DSL) representation

explicit declarative representation of an actor is optional

defines attributes with initial values

procedural (Python) representation

instance of class *Actor* or derived class



(ETOS) Transaction

task executing actions from (part of) lifetime of an actor (actor's carrier)

declarative (SIM-DSL) representation

- **SIM-DSL node** with child nodes describing individual actions
- transaction can be nested (auxiliary constructs)

procedural (Python) representation

- instance of the class *Transaction* (derived from *SimPy.Process* class)
- main method -- *action*: generator based coroutine

(<u>E</u>TOS) Entity

basic building block – individual executing step of transaction

A) simple **activity** of actor = end user entities

B) interface to shared services (resources, storages, etc.)

C) control statement
loop, exception handler, block, etc.

D) subtransaction
independent *SimPy* process, which execute part of transaction
e.g. body of loop

Entity representation

declarative (SIM-DSL) representation

SIM-DSL node encapsulating several types of subnodes:

- attributes of given entity
- common attributes of shared service
- subentities (control entity and subtransactions)

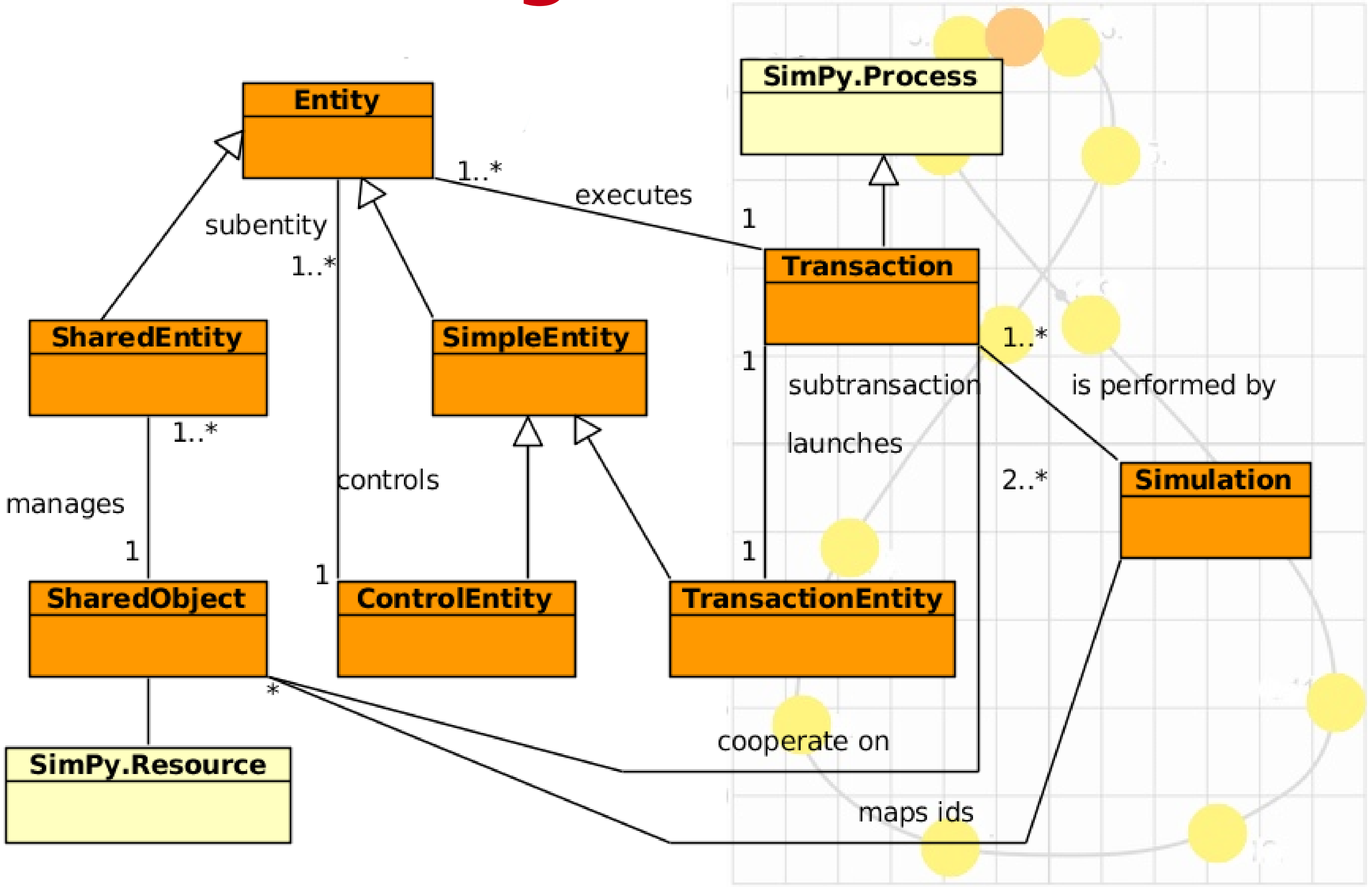
procedural (Python) representation

instance of class inheriting from *ETOS.Entity* class

shared service entity manages a shared object referring to shared service (*Resource*, *Level*, *Store*)

cooperation or competition of several transaction

Class diagram (UML)



XValue – value within context

- values in simulation are often related to simulation time, actors or they are bound to single action (entity)

  example (e-car, cash of recharge):

function of sim.time

random for every action = simulated travel

$$\text{cash} = \text{price} \bullet (\text{capacity} - t_{\textbf{travel}} \bullet \text{power})$$

function of actor's time

random for every actor = e-car

- x-value type and its context are specified only in declarative code

  <power context=“actor“><normal mu=“2500“ sigma=“100“/></power>

Python code is quite simple with minimum of explicit context references:

```
cash = self.price*(a.capacity – self.t * a.power)
```

SIM-DSL XML

```
<transaction>
  <counted_loop count="#5">
    <work>
      <duration>
        <normal mu="30600" sigma="3600"/>
      </duration>
      <hourly_wage context="transaction">
        <lognormal mu="1.73"  sigma="0.57"/>
      </hourly_wage>
    </work>
    <transport>
      <distance context="transaction">
        <lognormal mu="8.0" sigma="1.5"/>
      </distance>
      <fare_per_km>0.09</fare_per_km>
    </transport>
    <checkpoint>
     <measure property="a.balance" type="log"/>
   </checkpoint>
  </counted_loop>
</transaction>
```

duration
is randomly set for each
loop iteration
(implicit context: entity)

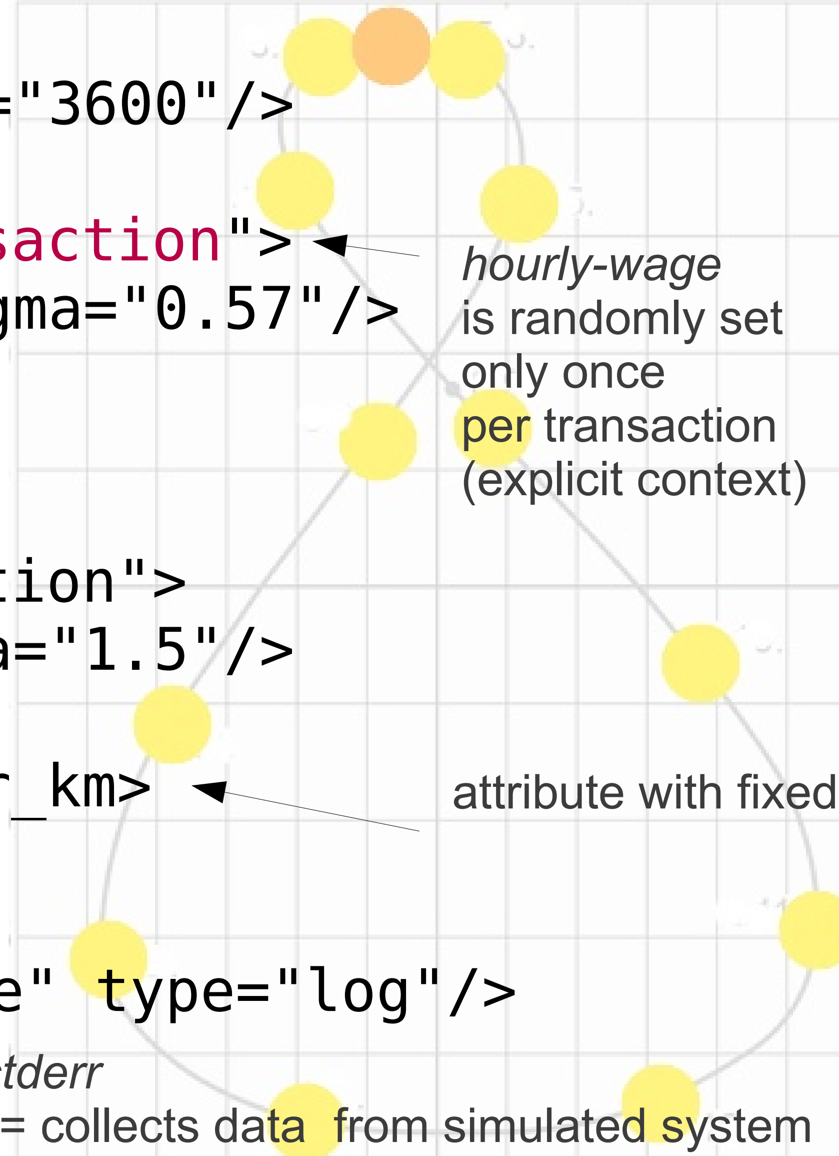
control
entities

model
entities

hourly-wage
is randomly set
only once
per transaction
(explicit context)

attribute with fixed

logging to *stderr*
checkpoint = collects data from simulated system



Python

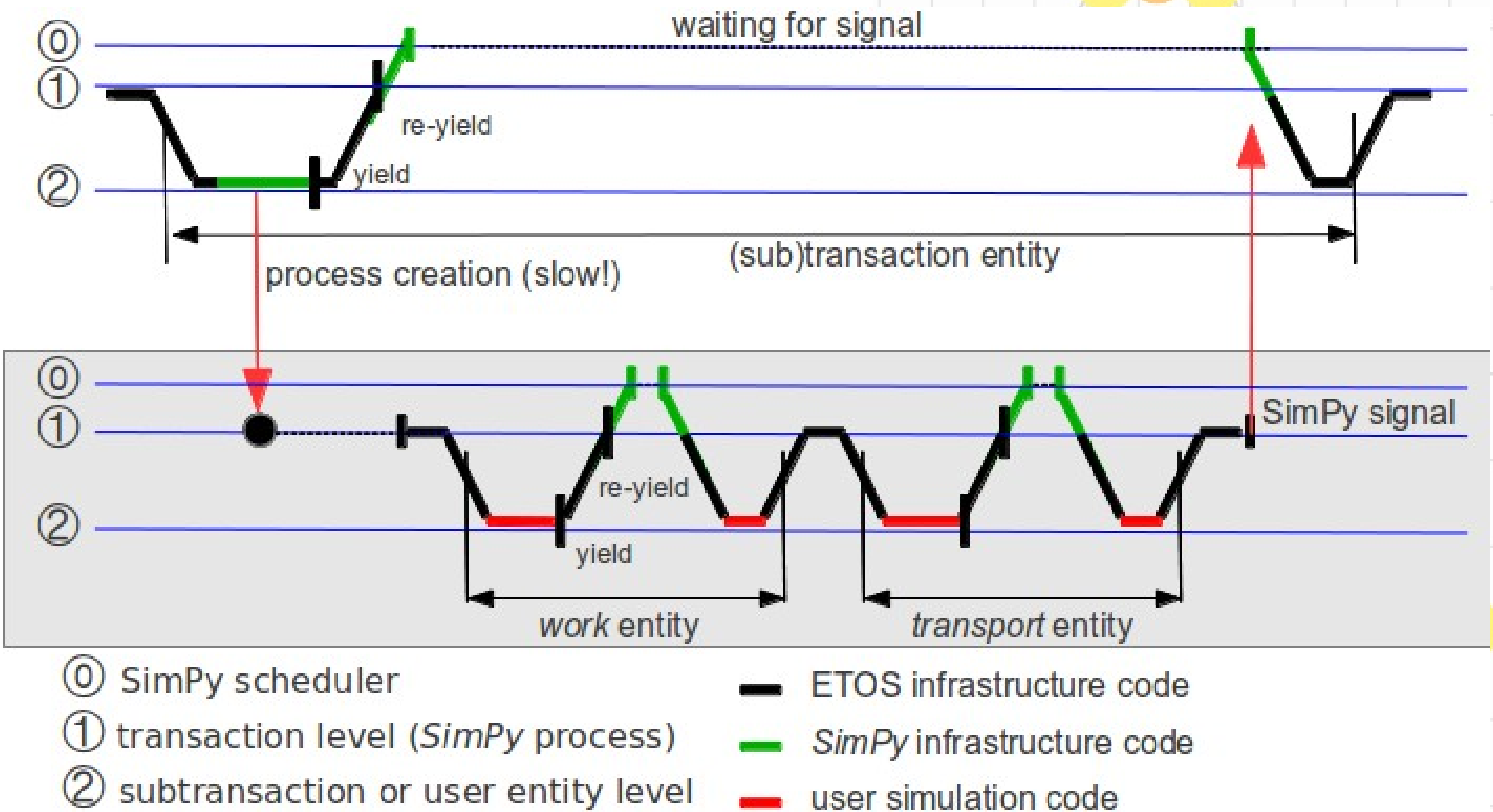
```python
class Person(Actor):
    def __init__(self, simulation):
        super().__init__(simulation)
        self.balance = 0.0

class Work(SimpleEntity):
    tag = "work"
    def __init__(self, transaction, xmlSource):
        super().__init__(transaction, xmlSource)
        self.attributeSetter(("duration", float),
                             ("hourly_wage", float))

    def action(self):
        income = self.duration * self.hourly_wage / 3600.0
        self.actor.balance += income
        yield self.hold(self.duration)

registerModule(SummerSolstice)
sim = Simulation()
sim.start(xmlFile, actor = Person(sim))
```

SIM-DSL node name

registration
of entity attributes
(name and type)

registration
of external names
from entire Python module

Transfer of control (sub-transaction)



Example of result

- simulation of e-car traffic
  100 e-car/day, 20 day
- **inputs**:

  Škoda Octavia Green-E-Line

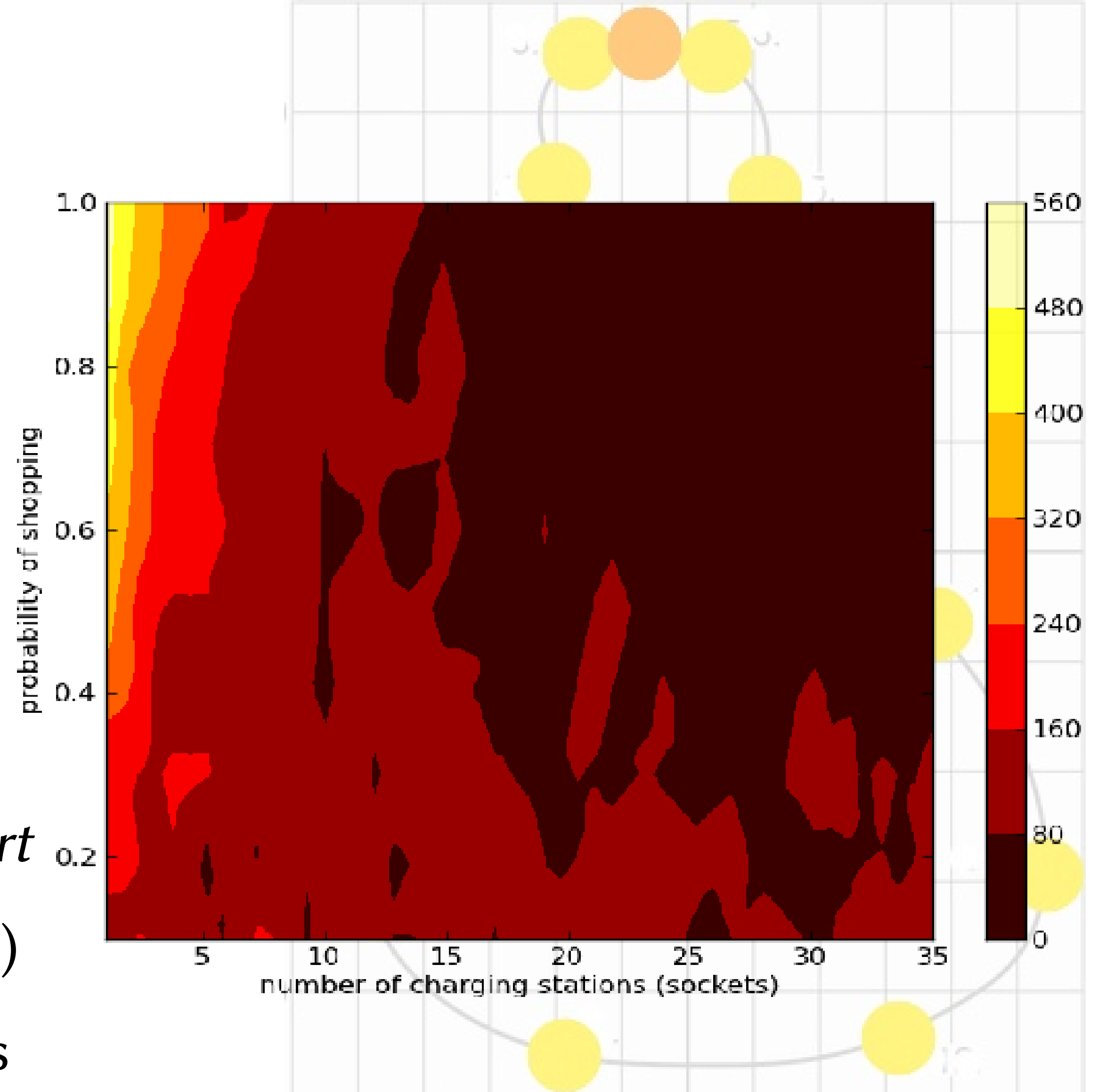
  home charging station

  fast (CC) charging station

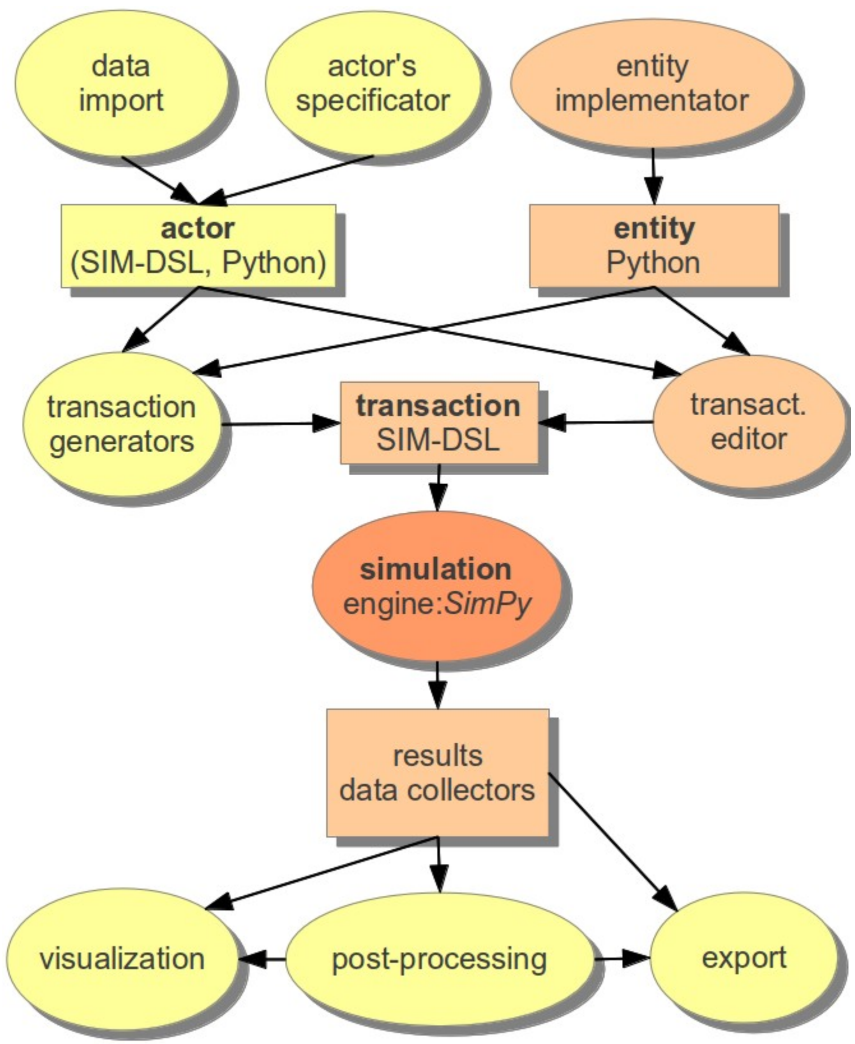
  simple shopping-center model
- **output**: *matplotlib contour chart*

  (nicknamed: „ Hell of shopping“)

  number of out-of-battery events



Conclusions I
Separation of roles



key roles:

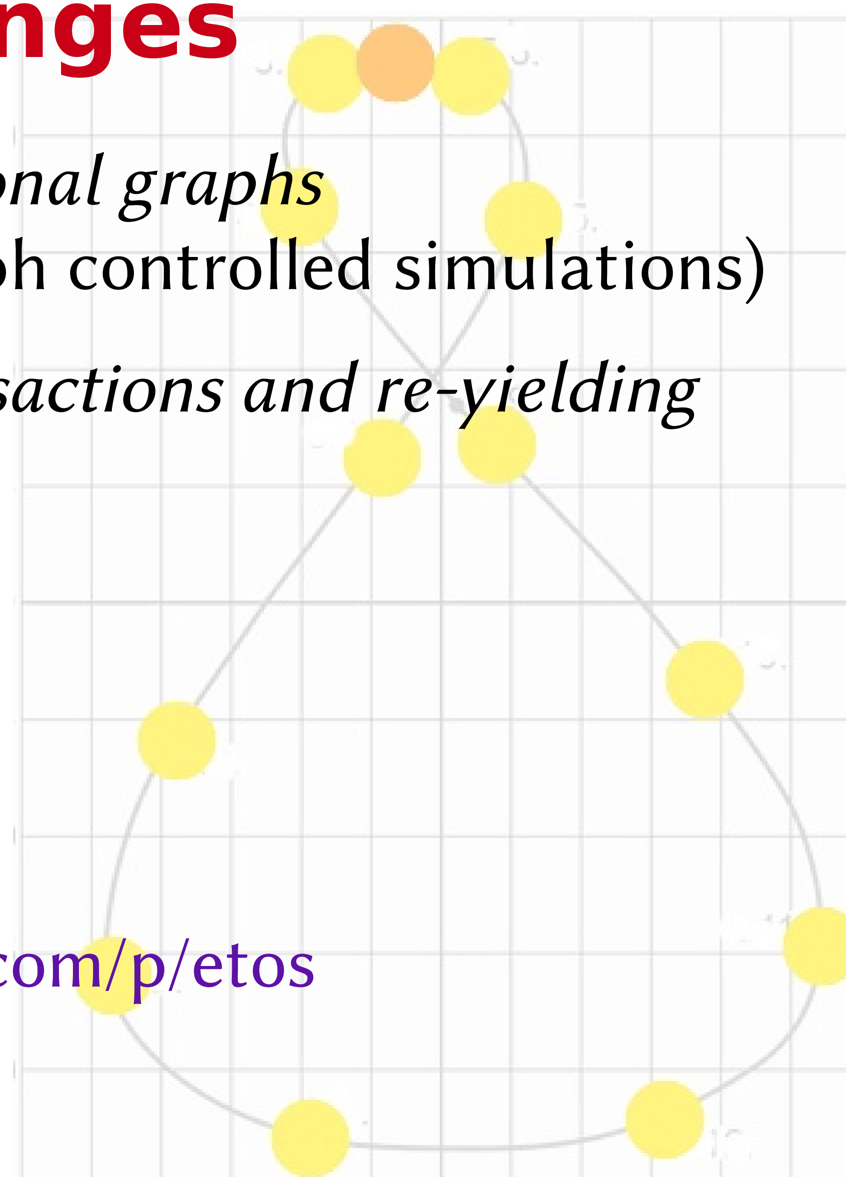
- *specifiers of actors* (input, global states, SIM-DSL)
- *entity implementors* (activity specification, SimPy programming)
- *transaction editors* (high level model, SIM-DSL)
- final processing (statistics, vizualization, ...)

Conclusions II
New challenges

- *SIM-DSL representation of directional graphs*
  (for more complex inputs and graph controlled simulations)
- *automatic optimization of subtransactions and re-yielding overheads*
- documentation with examples
- active software project

  project is hosted on *Google Code*

http://code.google.com/p/etos



Thank you for your attention

Dziękuję za uwagę