

# Generation of artificial event logs

Ivan Shugurov

Alexey A. Mitsyuk

# Motivation

There is a **need for testing and evaluation** in the development of a process mining algorithm

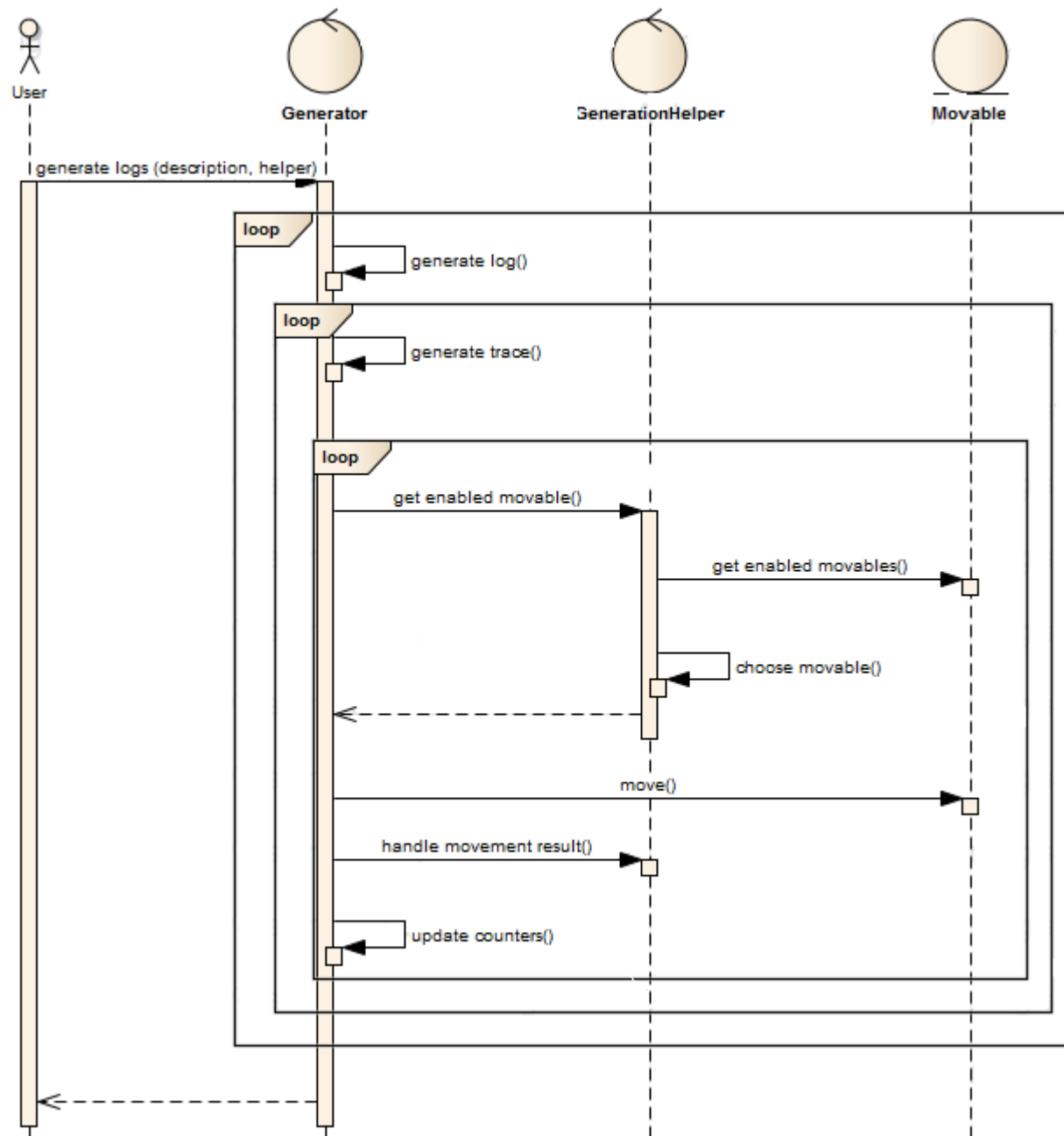
Appropriate (**artificial**) **examples for** the very first experiments and **idea evaluation**

- 1\ Lack of event logs for testing
- 2\ Available real life logs are too complex and inappropriate for preliminary idea evaluation
- 3\ Fast customization of a log sometimes needed

# Related work

- Manual generation
  - Very flexible
  - **Time consuming**
- CPN Tools
  - Writing code in **ML language**
  - Cannot produce events logs in XES format
- SecSy Tool
  - Can create sets of logs
  - Can add deviations
  - Complex settings
  - **Unintuitive user interface**
  - **Special security-oriented format of an event log**
- Process Log Generator (PLG)
  - Generates a collection of models made of typical patterns
  - Generated models are not really BPMN diagrams
  - Flexible settings of model generation, log generation
  - **Can not generate log using a given model**

# General description of approach



# Common features of generators

- Arbitrary number of generated logs
- Arbitrary number of traces per log
- Arbitrary maximum number of steps during replay – prevent infinite loops

# Petri Nets

# Types of generation

- Simple generation
  - Noise
- Generation with static priorities
- “Time-driven” generation
  - Time
  - Noise
  - Resources



Simple generation

# Simple generation: noise

- ***From internal (existing) transitions*** – use transitions in erroneous order
- ***From artificial events*** – create a list of events which do not correspond to transitions of a Petri net, but may appear in event logs
- ***Skipping events*** – just don't record occurred events into event logs




# Noise

- Applied while executing an event
- ***Noise level*** – probability of a transition to be distorted

# Simple generation: noise settings screen

## Noise settings

Noise level	5
Use internal transitions	<i>selected</i>
Use artificial events	<i>selected</i>
Use skipping events	<i>selected</i>



Generation with static priorities

- ***Priority*** – positive integer
- Always executes transition with the *highest* priority
- If several transitions have the same priority – random one is picked

Time-driven generation

# Time-driven generation: main screen

**General settings**

Number of logs	5
Number of traces	10
Maximum number of steps	100
Minimum interval	10
Maximum interval	20
Hours	13
Minutes	54
Seconds	27
Use noise	<i>selected</i>
Separate start and finish	<i>selected</i>
Remove empty traces	<i>selected</i>
Remove unfinished traces	<i>selected</i>
Use resources	<i>selected</i>
Use complex resource settings	<i>selected</i>
Use synchronization on resources	<i>selected</i>




 Cancel  Previous  Next



# Time-driven generation: noise settings

## Noise settings

Noise level	5
Max timestamp deviation	6000
Use internal transitions	<i>selected</i>
Use artificial events	<i>selected</i>
Use skipping events	<i>selected</i>
Use noise in lifecycle	<i>selected</i>
Use timestamp noise	<i>selected</i>
Use time granularity	<i>selected</i>



# Resources

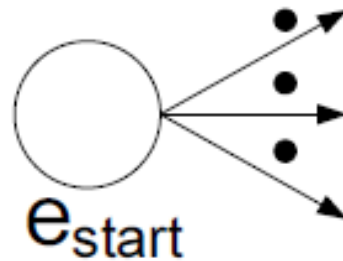
- Resource specified by name
- Resource specified by group, role, name

BPMN

Supported semantic

# Events

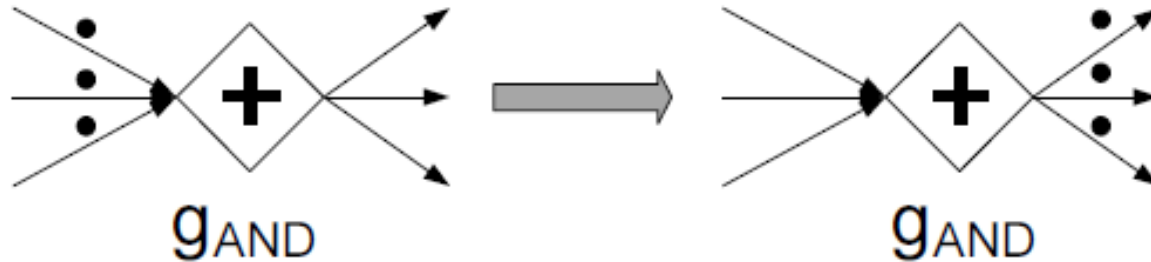
Each process must have one Start and at least one End event



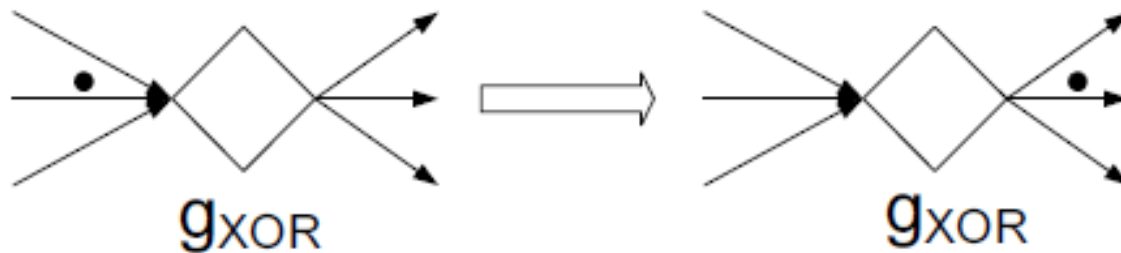
# Activities



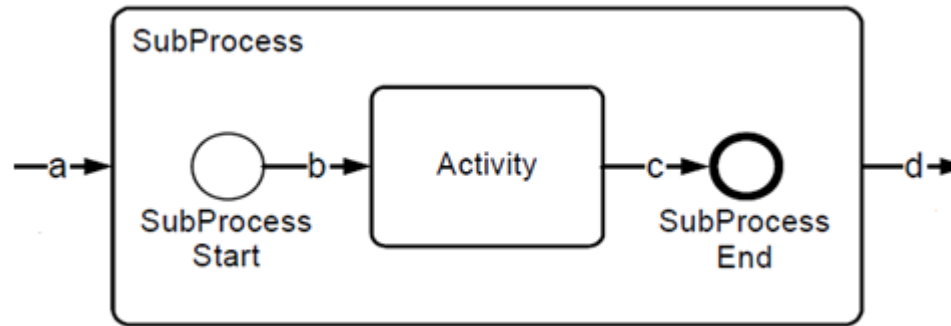
# Parallel gateway



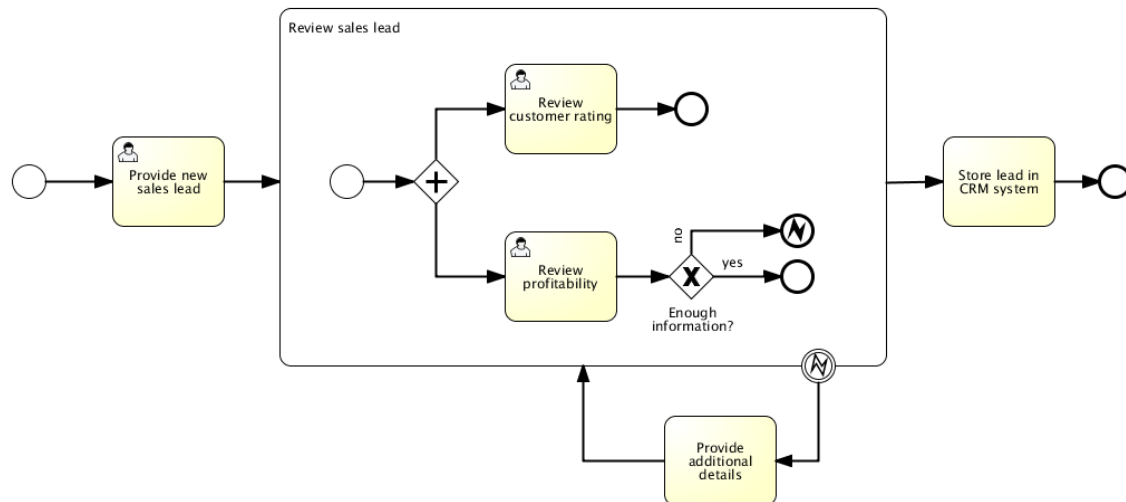
# Exclusive choice gateway



# Sub-processes

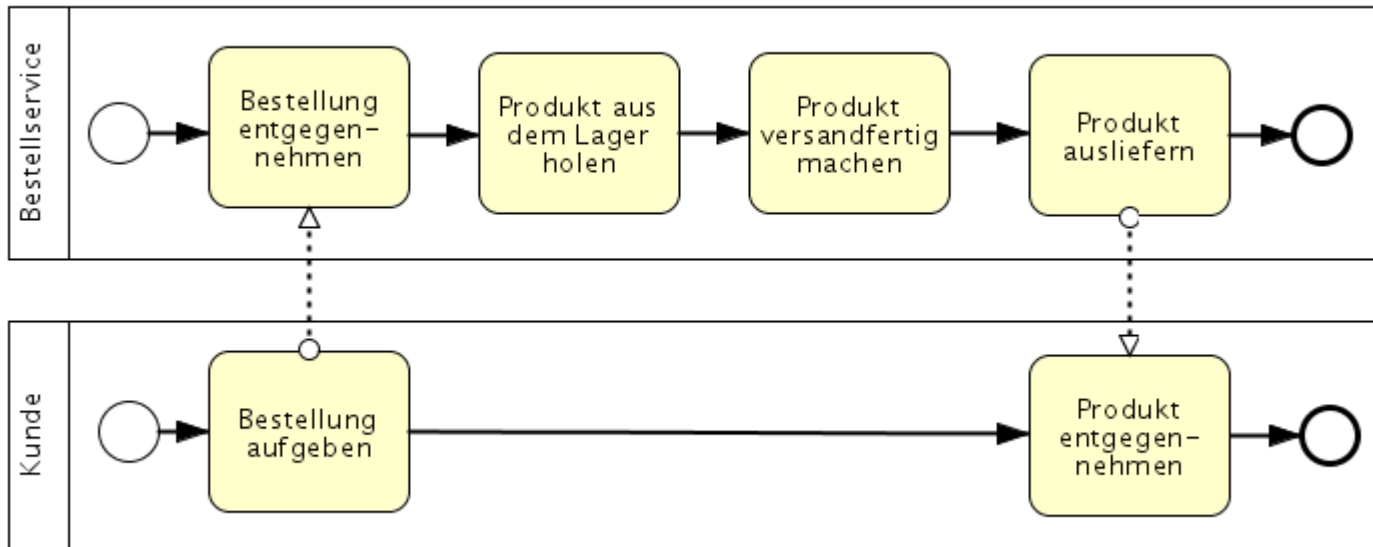


# Cancel events

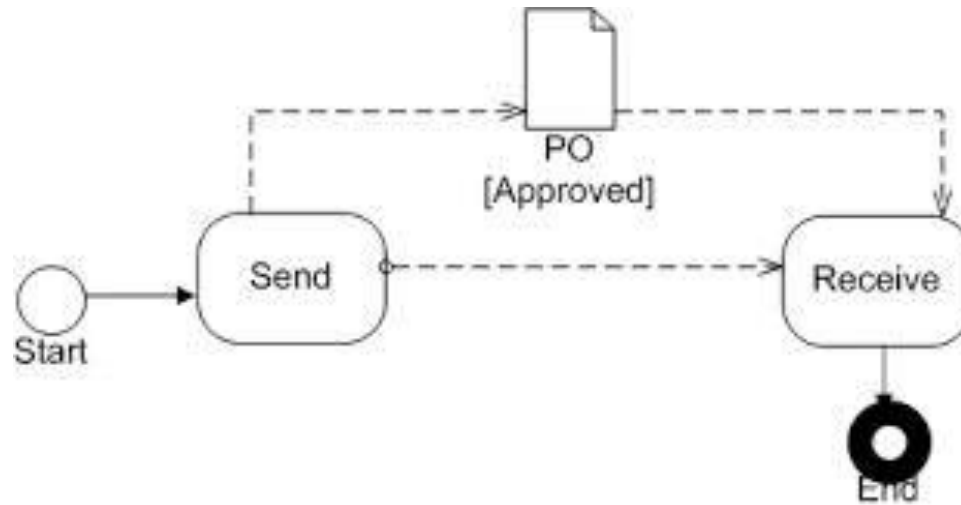




# Pools, lanes, messages



# Data objects



# Firing preferences

- Each sequential flow which follows from exclusive gateway has an integer number called preference
- The higher a preference, the higher a chance of the corresponding flow to get a token
- Preference is bigger or equal to 0 and less or equal to 100
- Sum of preferences is more than 0

$$P(SF_i) = \frac{Pref(SF_i)}{\sum Pref(SF)}$$

# Conclusions

- Implemented easily extendable framework
- Implemented generation of logs for Petri nets
- Implementation of log generation for BPMN is in progress

# Future work

- Improve framework
- Finish log generation for BPMN models
- Incorporate other formalisms

# References

- Wil M. P. van der Aalst, Process mining: discovery, conformance and enhancement of business processes. Springer, 2011.
- A. K. A. d. Medeiros and C. W. Gunther, “Process mining: Using CPN tools to create test logs for mining algorithms,” in Proceedings of the Sixth Workshop on the Practical Use of Coloured Petri Nets and CPN Tools (CPN 2005), ser. DAIMI, K. Jensen, Ed., vol. 576. Aarhus, Denmark: University of Aarhus, 2005, pp. 177–190.
- A. Burattin and A. Sperduti, “PLG: a framework for the generation of business process models and their execution logs,” in BPM 2010 Workshops, Proceedings of the Sixth Workshop on Business Process Intelligence (BPI2010), ser. Lecture Notes in Business Information Processing, J. Su and M. z. Muehlen, Eds., vol. 66. Springer-Verlag, Berlin, 2011.

# References

- T. Stocker and R. Accorsi, “Secsy: Security-aware synthesis of process event logs,” in Proceedings of the 5th International Workshop on Enterprise Modelling and Information Systems Architectures, St. Gallen, Switzerland, 2013
- Shugurov I., Mitsyuk A. A. Generation of a Set of Event Logs with Noise, in: Proceedings of the 8th Spring/Summer Young Researchers’ Colloquium on Software Engineering (SYRCoSE 2014). M. : -, 2014. P. 88-95.