Integration of Reinforcement Learning and Discrete Event Simulation Using the Concept of Experimental Frame

Thorsten Pawletta, Jan Bartelt

Wismar University of Applied Sciences
Faculty of Engineering / Research Group CEA
{thorsten.pawletta, jan.bartelt}@hs-wismar.de

EUROSIM Congress, Amsterdam/NL, July 3-5, 2023

HOCHSCHULE WISMAR
UNIVERSITY OF APPLIED SCIENCES
TECHNOLOGY, BUSINESS AND DESIGN

Outline

- **Introduction**
- **Basics**
  - Structure of Simulation Based Experiments (SBE)
  - Concept of Experimental Frame (EF)
  - Reinforcement Learning (RL)
- **SBE with integrated RL using EF**
- **Case study with MATLAB/SimEvents**
- **Conclusions**

Introduction

Observation

Simulation models are often (tricky) implemented to fit RL needs

→ complicated model structures and limited reusability of comps & meths

Objectives

(M&S conform model & experiment design)

Clear separation between:

Model Under Study (MUS) and

Context of use (experiment)

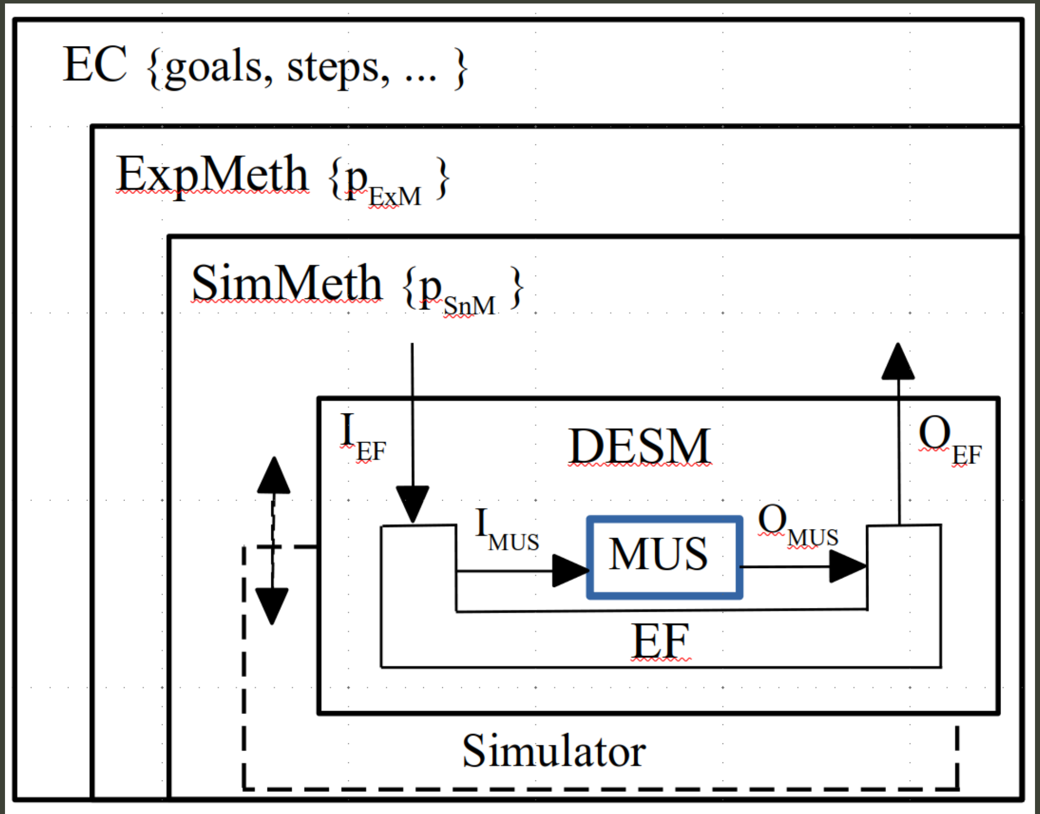
to support:

Independent development and

General reusability of MUS and experiment methods

Basics

Structure of Simulation Based Experiments (SBE)



EC Experiment Control

ExpMeth Experiment Method

SimMeth Simulation Method

Simulator (MATLAB/SimEvents)

DESM Discrete Event Simulation Model

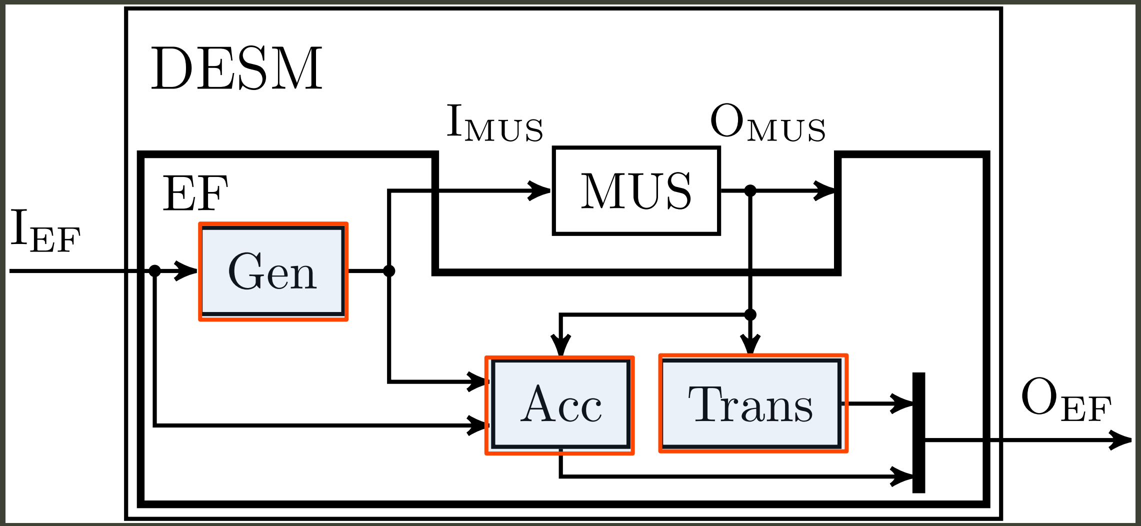
EF Experimental Frame

MUS Model Under Study

The MUS is part of an experiment, involving multiple levels of methods.

DESM structure using Experimental Frame (EF)

(EF introd. by B.P. Zeigler)



- **DESM** is divided into **MUS** & **EF**
- **EF** specifies the conditions under which a **MUS** is experimented with

Formal definition:

EF = <T, Ω_I, I_{MUS}, O_{MUS}, C, Ω_C, SU, I_{EF}, O_{EF}>

T may differ from MUS

Generator

comp. Ω_I for I_{MUS} and EF other comps

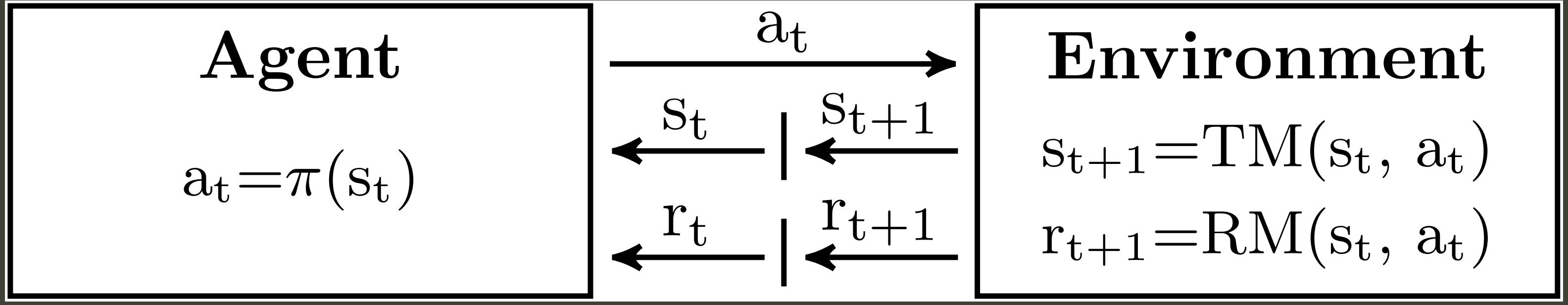
Transducer

comp. values of interest and SU based on O_{MUS}

Acceptor

checks compliance of Ω_C based on $C \subset O_{MUS}$

Reinforcement Learning (RL)



t **order of tupels** (a_t s_t)

a_t action ($a \epsilon A$)

s_t state ($s \epsilon S$)

r_t reward ($r \epsilon R$)

TM transition model

RM reward model

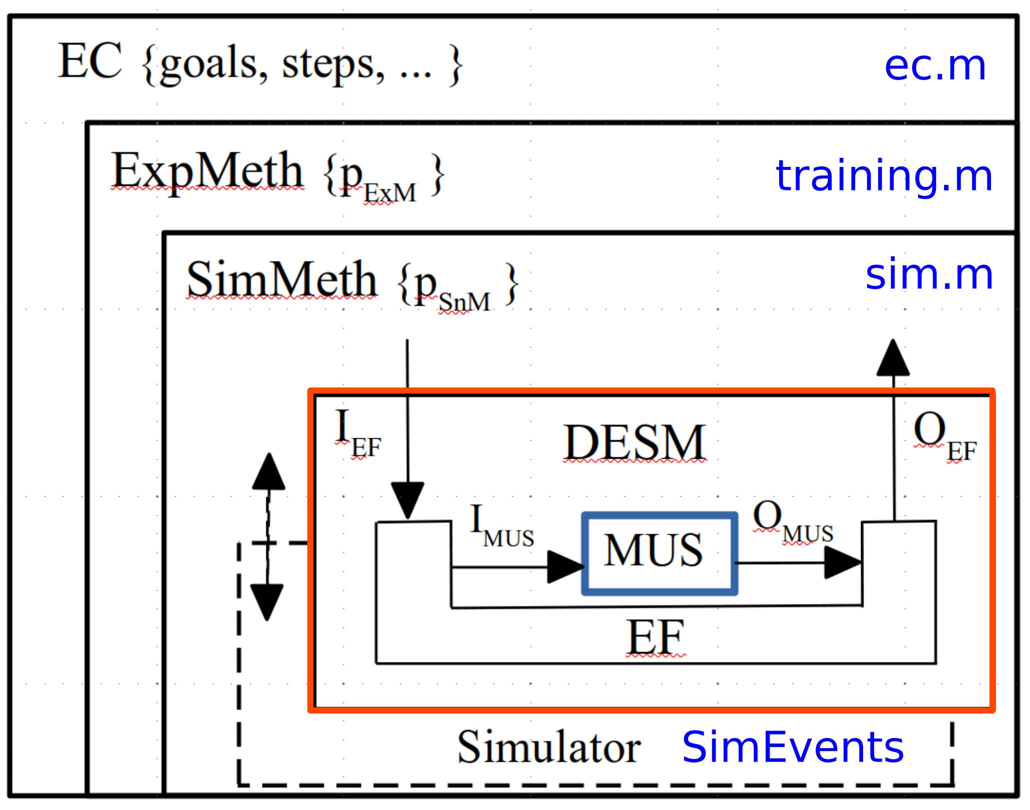
π policy

- Agent observes state s_t of environment
- Agent chooses an action a_t according to its policy $\pi(s_t)$
- Environment executes its TM and RM and responds with (s_{t+1} r_{t+1})
- Agent improves (learns) policy $\pi(s_t)$ to maximize the cumulative reward
  - Various learning approaches (Q, DQN, ...)
  - Training is done by repetition of episodes starting with environment in s_0 to s_{final} | s_{abort}

SBE with integrated RL using EF

Structure of a SBE with integrated RL

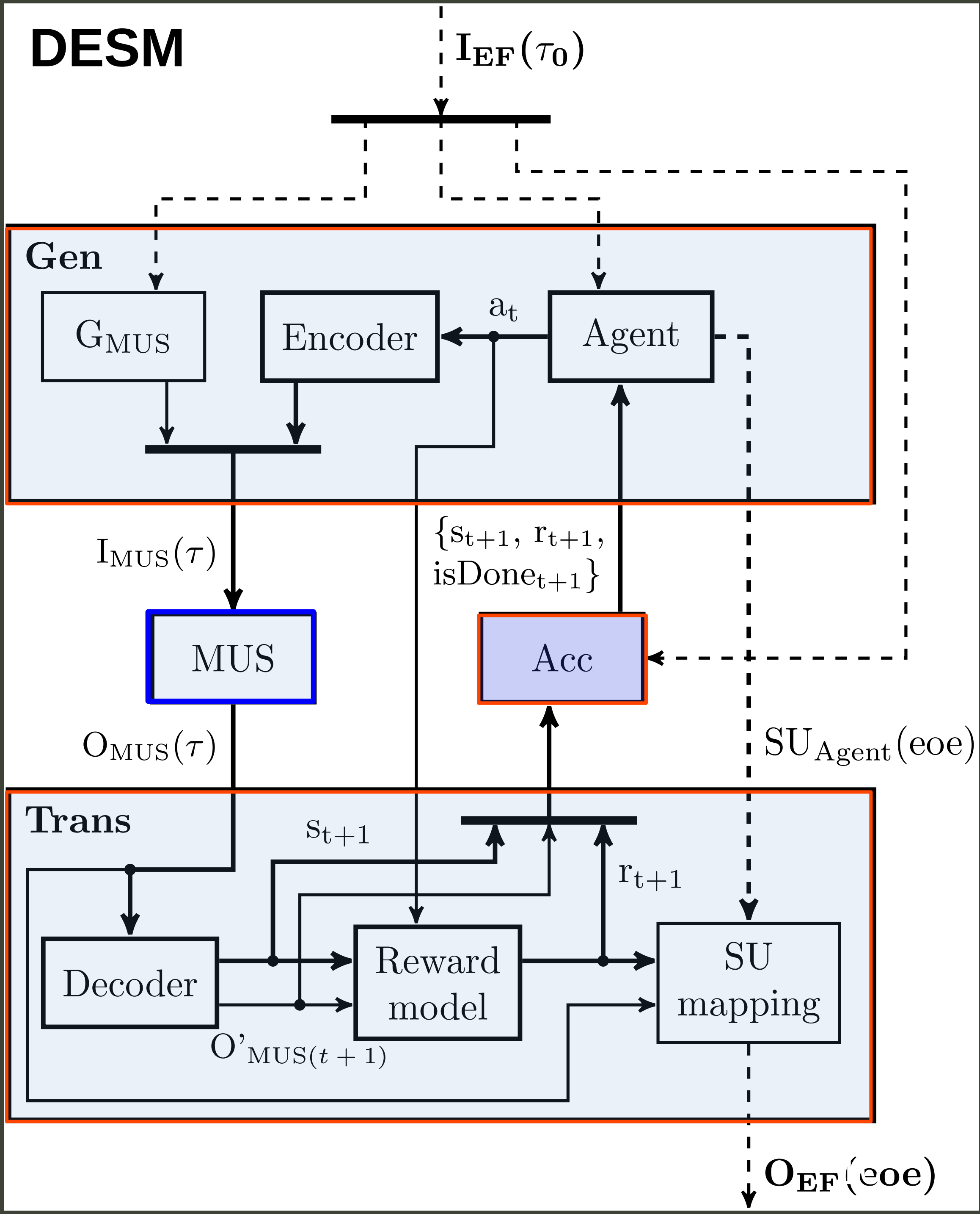
(e.g. with MATLAB/SimEvents)



- **EC** sets training-, sim-, and DESM parameters
- **ExpMeth** is the **training** alg. and ctrls computation of episodes
- **SimMeth** ctrls sim run (1 episode)
- **Simulator** executes sim run
- **DESM** implements
  - **MUS** (as part of *RL Env*)
EF (*Agent* & *RL specific Env.*)

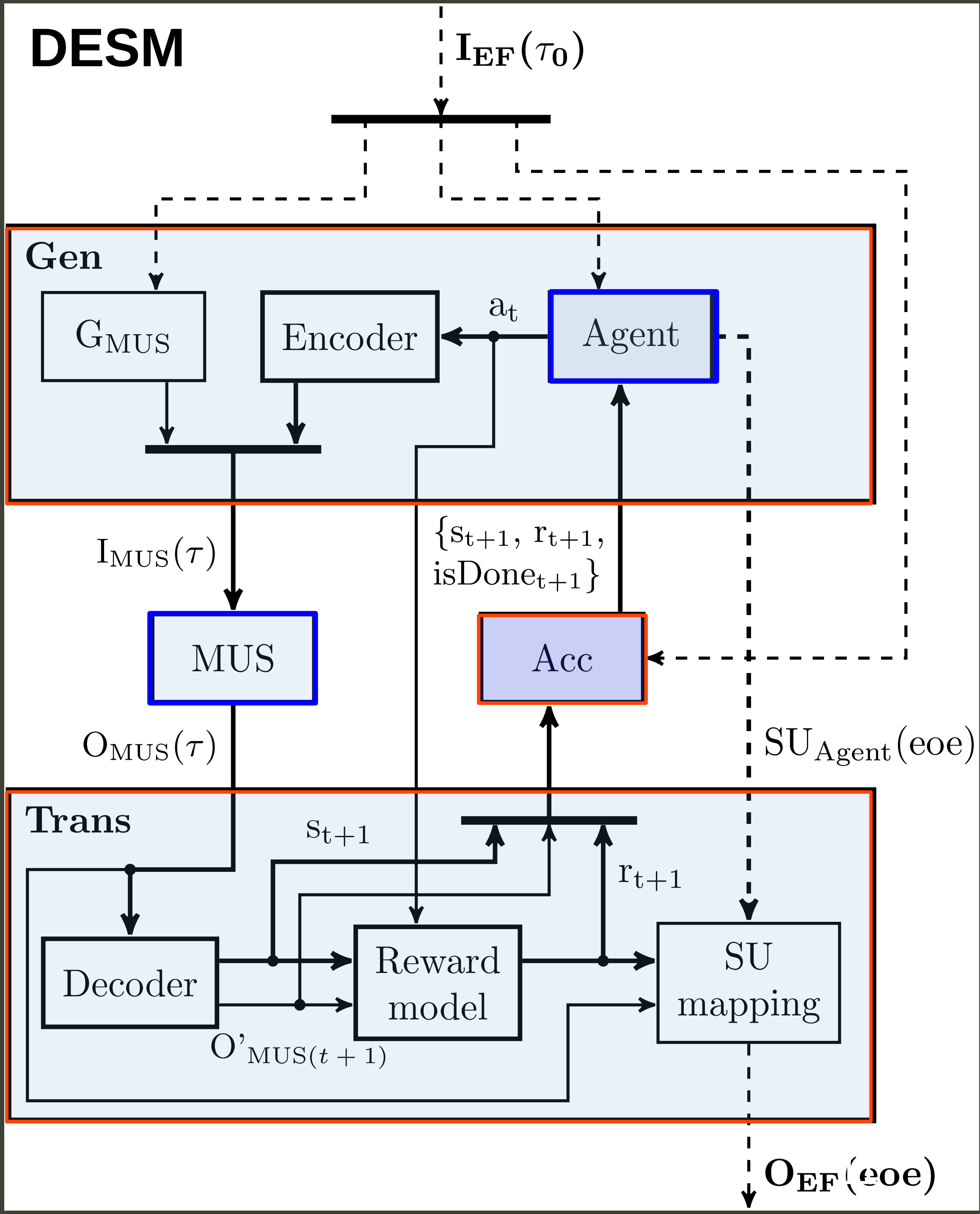
DESM with MUS & EF for RL

- **MUS:** discr event system with **contin time τ**
- **EF** comps are time-triggered or event-driven by MUS outputs → **sequent. order t** of RL



DESM with MUS & EF for RL

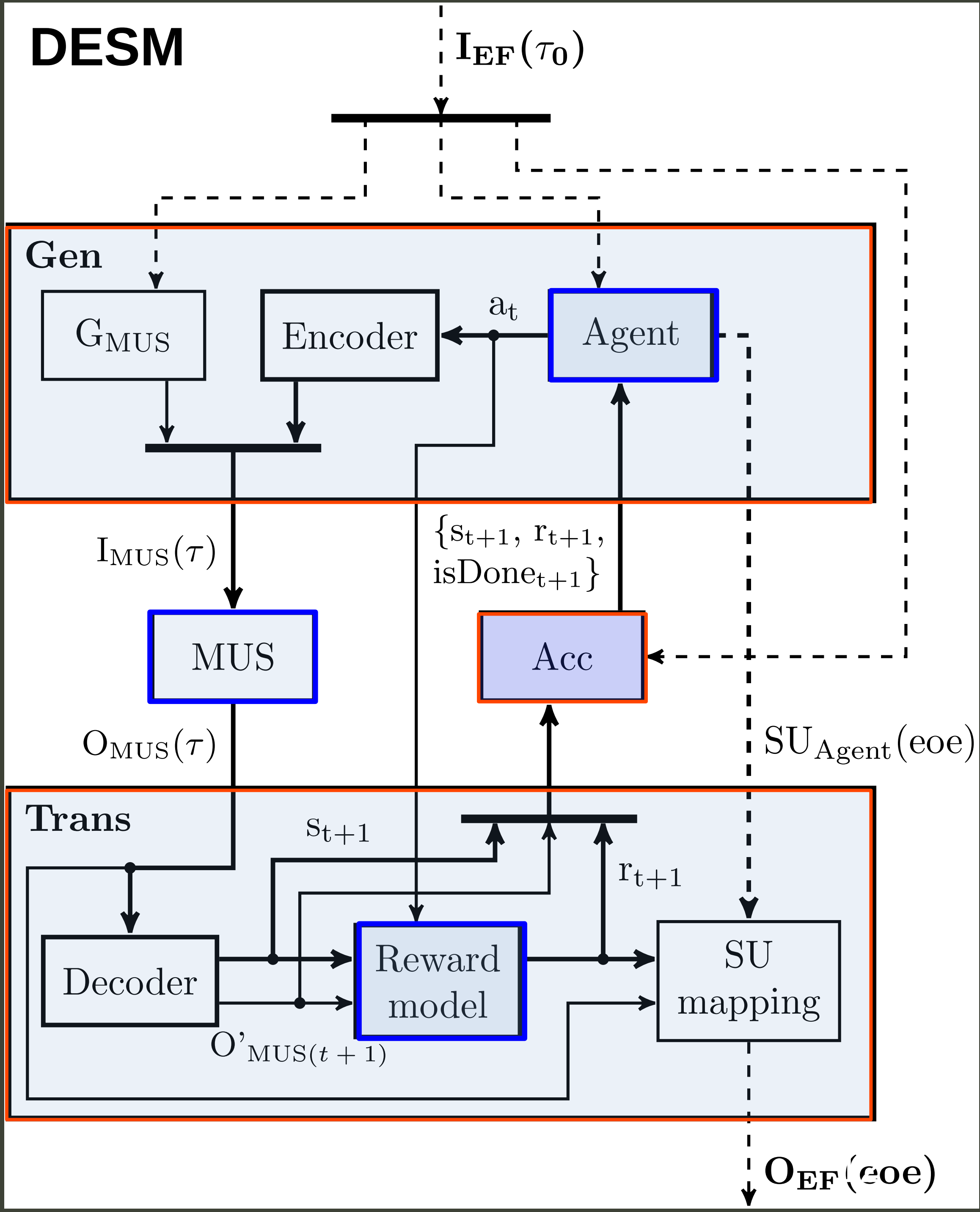
- **MUS:** discr event system with **contin time τ**
- **EF** comps are time-triggered or event-driven by MUS outputs → **sequent. order t** of RL
- **Agent** is a **Generator** a_t=**Agent**(s_t, r_t, isDone)



DESM with MUS & EF for RL

- **MUS:** discr event system with **contin time τ**
- **EF** comps are time-triggered or event-driven by MUS outputs → **sequent. order t** of RL

- **Agent** is a **Generator** a_t=**Agent**(s_t, r_t, isDone)
- **Reward Model** (RM) is a **Transducer** comp → not part of MUS, r_{t+1}=**Reward**(s_{t+1}, O'_{MUS})

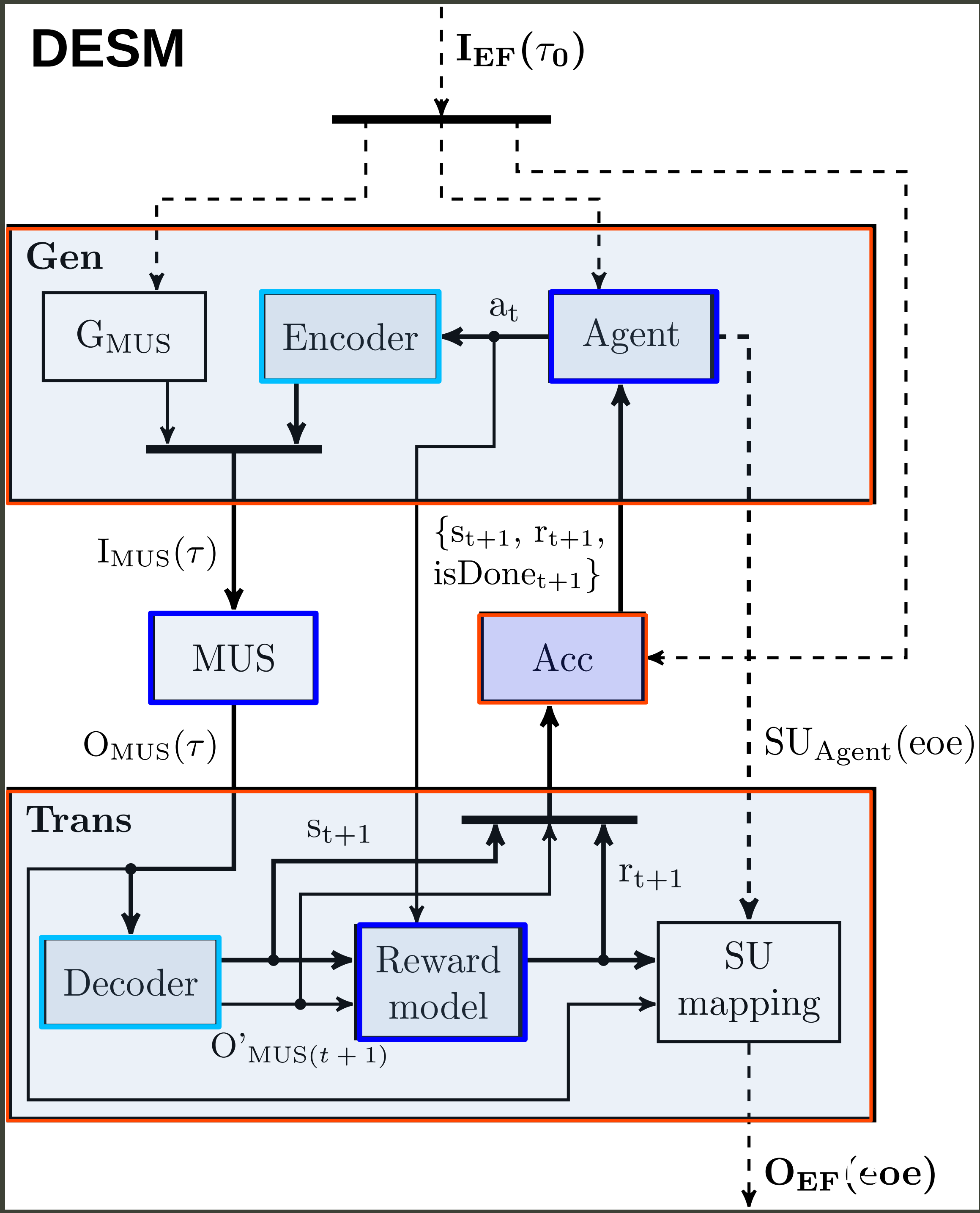


DESM with MUS & EF for RL

- **MUS:** discr event system with **contin time τ**
- **EF** comps are time-triggered or event-driven by MUS outputs → **sequent. order t** of RL

- **Agent** is a **Generator** a_t=**Agent**(s_t, r_t, isDone)
- **Reward Model** (RM) is a **Transducer** comp → not part of MUS, r_{t+1}=**Reward**($s_{t+1,}$ O'_{MUS})

- **Encoder** & **Decoder** transform the differing state/action representation of MUS & RL
  $I_{MUS}(\tau)$=**Encoder**(a_t), [$s_{t+1,}$ O'_{MUS}]=**Decoder**($O_{MUS}(\tau)$)
  MUS + Decoder are the TM in sense of RL



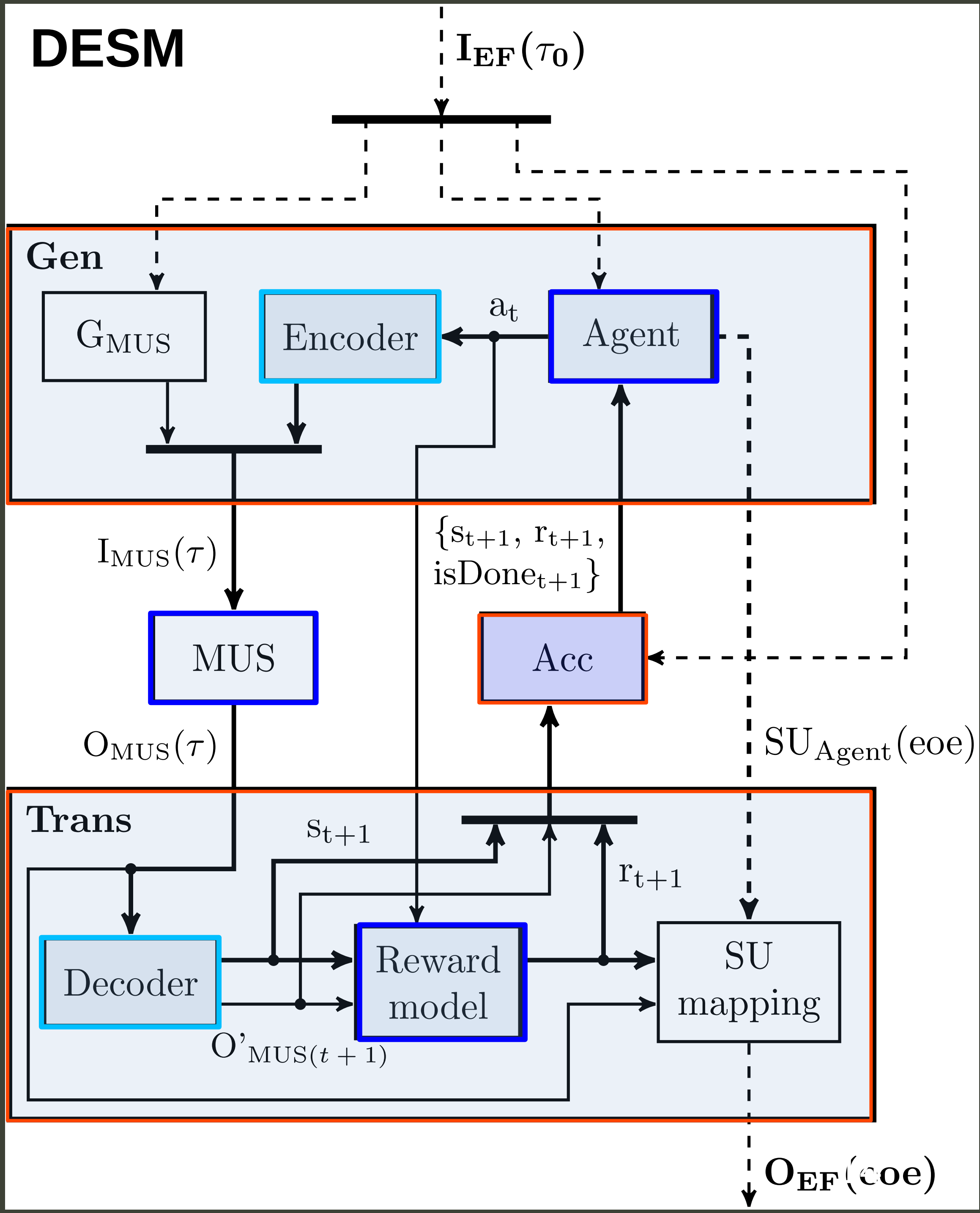
DESM with MUS & EF for RL

- **MUS:** discr event system with **contin time τ**
- **EF** comps are time-triggered or event-driven by MUS outputs → **sequent. order t** of RL

- **Agent** is a **Generator** a_t=**Agent**(s_t, r_t, isDone)
- **Reward Model** (RM) is a **Transducer** comp → not part of MUS, r_{t+1}=**Reward**(s_{t+1}, O'_{MUS})

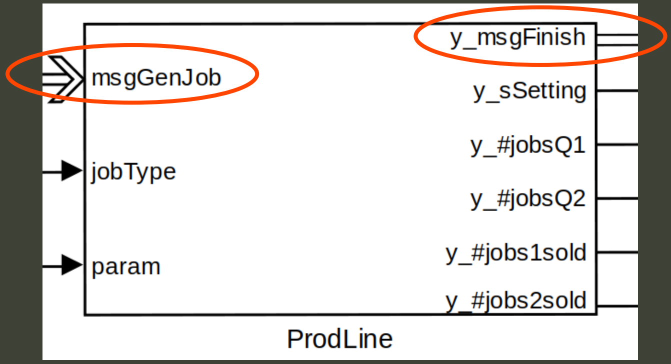
- **Encoder** & **Decoder** transform the differing state/action representation of MUS & RL
  I_{MUS}(τ)=**Encoder**(a_t), [s_{t+1}, O'_{MUS}]=**Decoder**(O_{MUS}(τ))
  MUS + Decoder are the TM in sense of RL

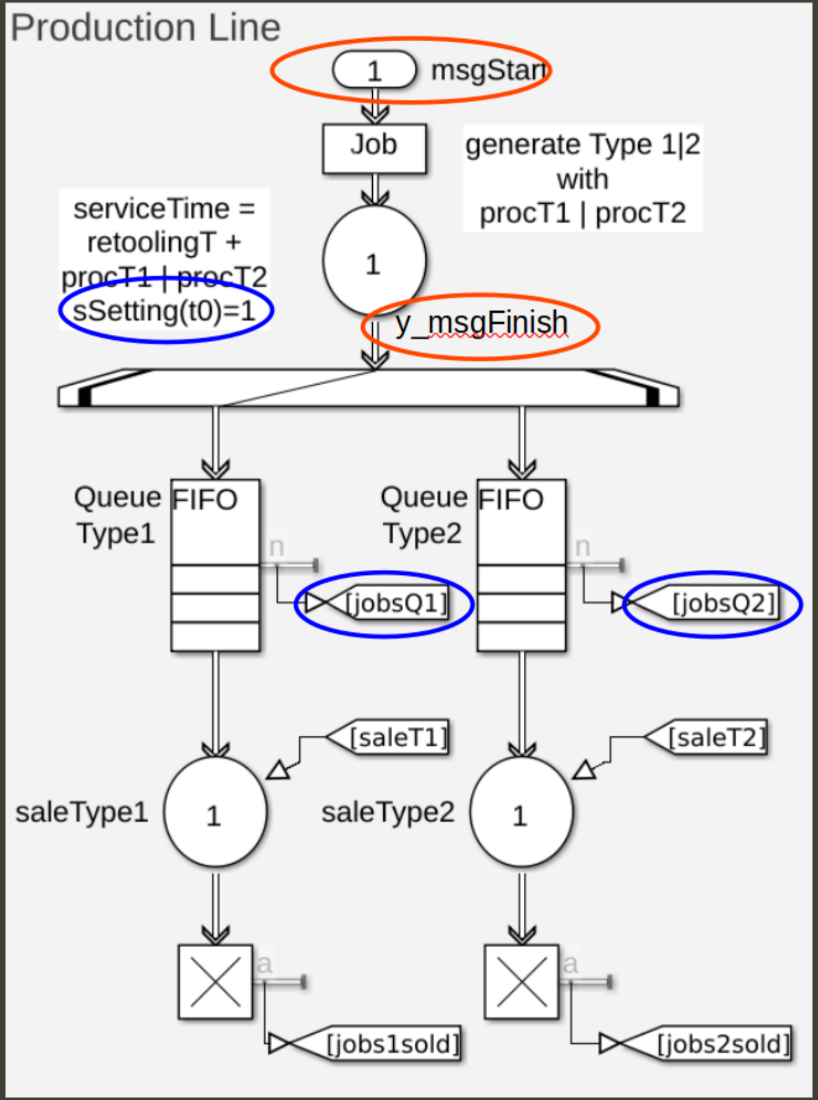
- **Acceptor** checks run conditions and sets isDone=0|1 for terminating RL episode



Case study with MATLAB/SimEvents

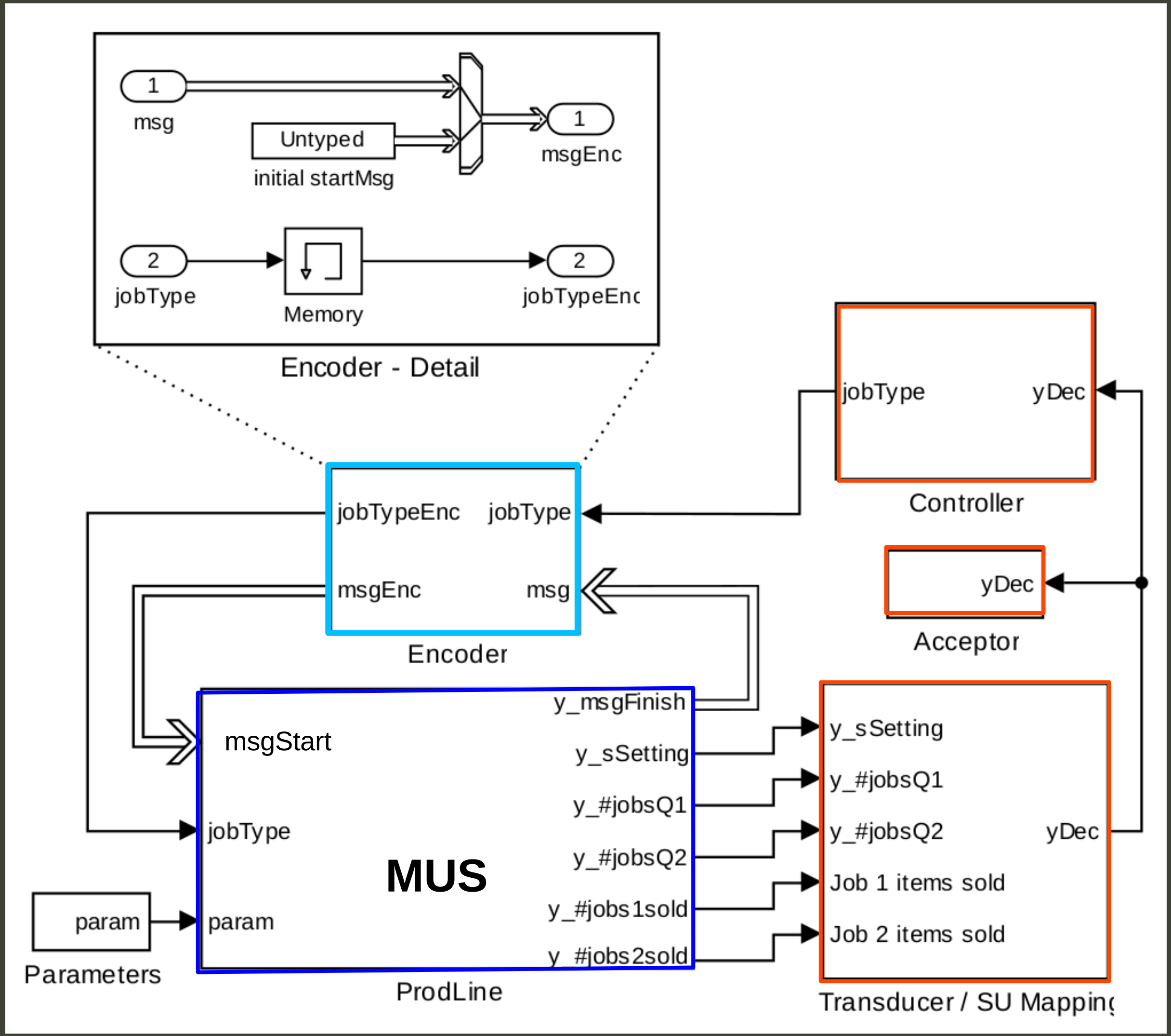
The MUS: simple ProductionLine





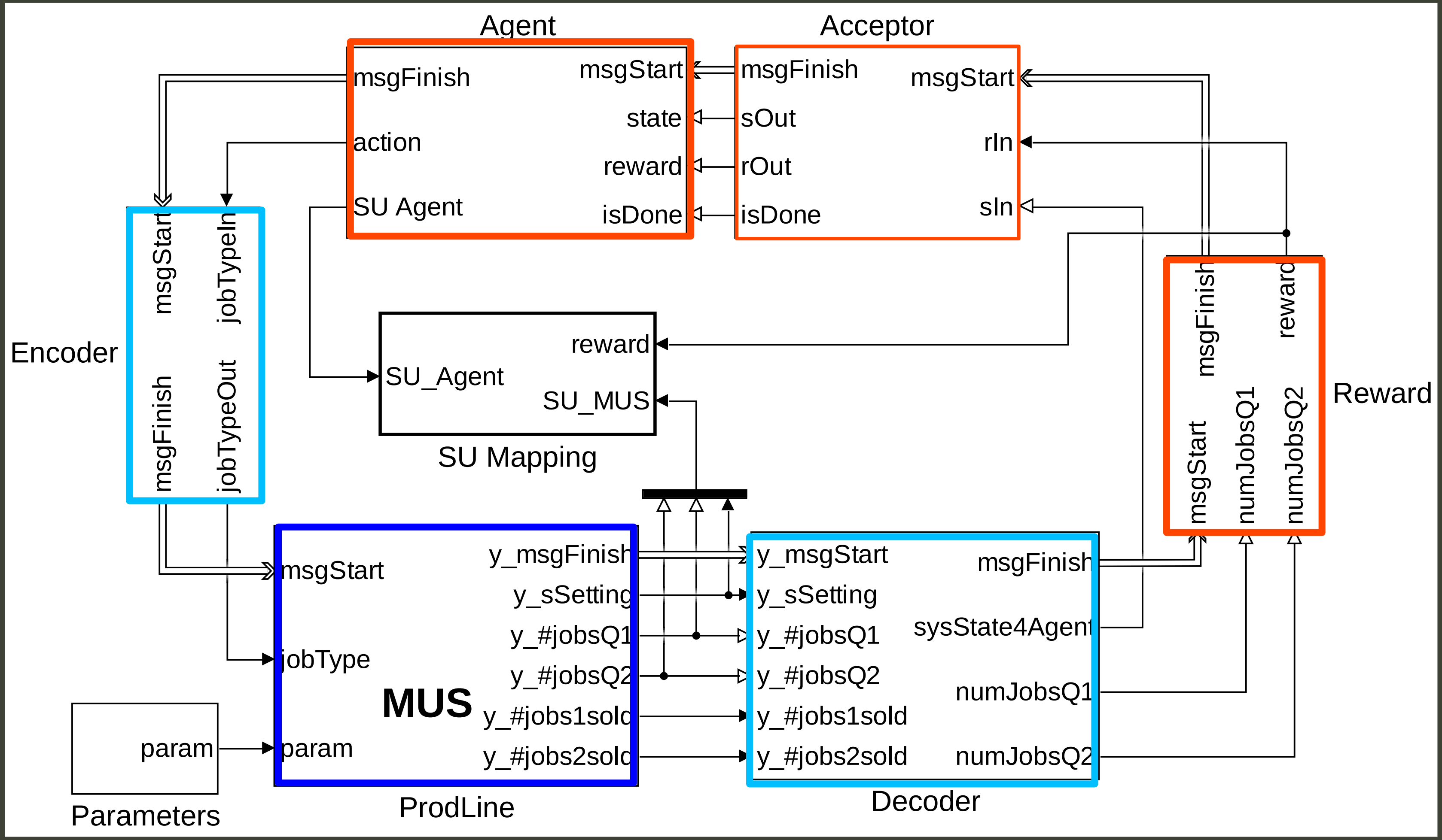
- Generation of different **jobTypes** is triggered by input event **msgStart**
- Service time of 1st server depends on inputs jobType and param
- Changing jobTypes require retooling time
- Jobs are routed to type-specific downstream queues/servers with different service time → output event **y_msgFinish**
- **Goal:** find best control strategy for injecting jobs to balance content of downstream queues

Structure of MUS and EF using a heuristic ctrl



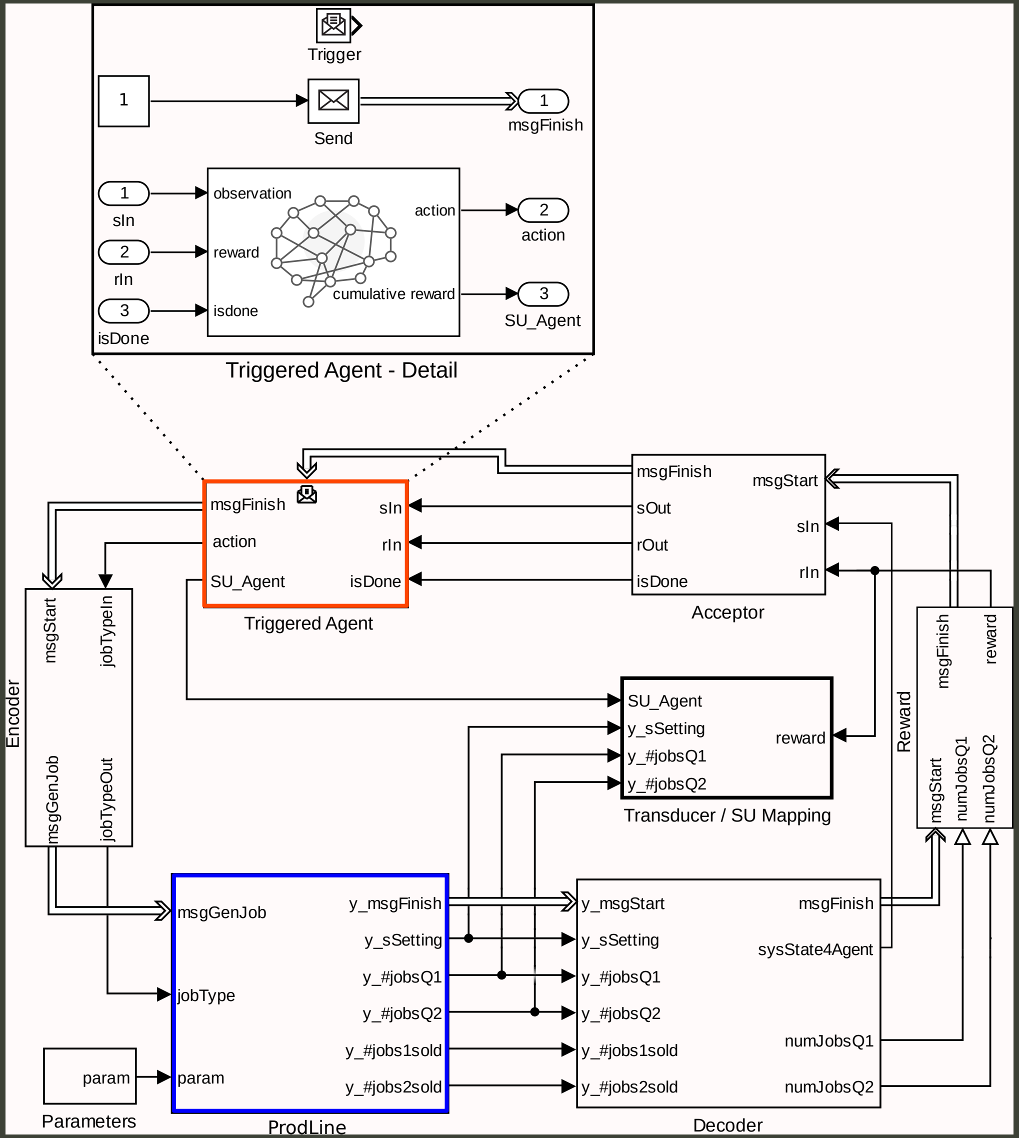
- **MUS ProdLine**
- **EF Generator Controller** computes next jobType based on Transducer output yDec
- **EF Generator Encoder** provides MUS conform inputs (msgStart, jobType)
- **EF Transducer** transforms MUS outputs for Controller (yDec with queue diff., …) and computes SU
- **EF Acceptor** checks queue diff. and time condition $\tau < \tau_{final}$ to exit sim run

Structure of MUS and EF using a Q-Agent



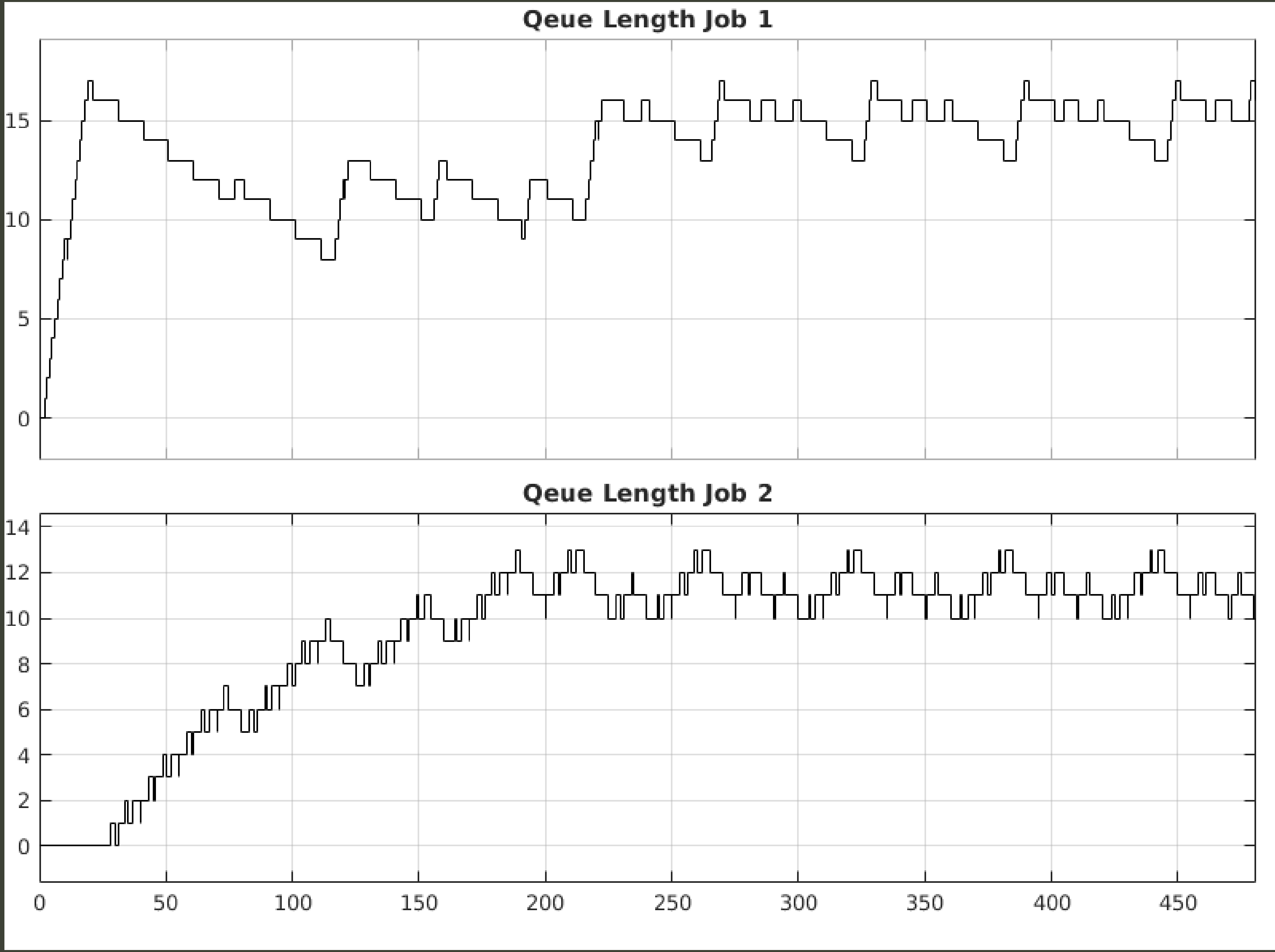
- **<u>Same</u> MUS Prodline**
- **EF is triggered by MUS output events**
- **EF Generator Q-Agent** computes action values a_t
- **EF Generator Encoder** transforms a_t to MUS inputs (msgStart, jobType)
- **EF Transducer Decoder** transf MUS outputs to RL conform values $\mathbf{s_{t+1}}$
- **EF Transducer Reward** computes r_{t+1}
- **EF Acceptor** checks all values and sets isDone

Structure of MUS and EF using MathWorks' RL Agent



- **Same MUS Prodline**
- **Nearly the same EF**
  - Q-Agent is replaced by **MathWorks' RL Agent**
  - **BUT RL Agent** isn't designed for event-driven simulations → using Trigerred Subsystem as a workaround
- **RL tbx** provides an **ExpMeth train** and a specific **SimMeth sim** (different from regular sim method)

Training Result after 5000 episodes



Conclusions

- Structure of SBE and concept of EF provide a clear methodological approach for integrating Discrete Event Simulation (DES) and RL
- The MUS and the experiment methods can be developed independently and reused in different contexts
- Case study depicted the reusability of a MUS in three different experiments

Backyard Slides

Operation of Decoder in the EF (for the Example with Q-Agent)

- **when MUS output event y_msgFinish**(τ), then

  Transducer.Decoder

  transforms MUS outputs
  y_sSetting(τ),
  y_#jobsQ1(τ),
  y_#jobsQ1(τ)

  to the **single RL state s_{t+1} sysState4Agent**

  computes other values of interest

  activates Transducer.Reward via event

$$qlQ1 = max(y_{numJobsQ1}, qlength_{max})$$
$$qlQ2 = max(y_{numJobsQ2}, qlength_{max})$$
$$s_{t+1} = (sSetting - 1) \cdot (qlength_{max} + 1)^2 + qlQ1 \cdot (qlength_{max} + 1) + qlQ2 + 1$$

Operation of Reward in the EF (for the Example with Q-Agent)

- **Transducer.Reward**

  computes

  r_{t+1} value based on
  the values of interest
  $numJobsQ1_{t+1}$
  $numJobsQ1_{t+1}$
  instead of RL next state **s_{t+1}**

  activates Acceptor
  via event

$$r_{t+1} = \begin{cases} 100 & | \; qlQ1 \geq 10 \wedge qlQ2 \geq 10 \\ qlQ2^2 & | \; qlQ1 \geq 10 \wedge qlQ2 < 10 \\ qlQ1^2 & | \; qlQ1 < 10 \wedge qlQ2 \geq 10 \\ \frac{qlQ2^2 \cdot qlQ1^2}{100} & | \; else \end{cases}$$

Operation of Acceptor in the EF (for the Example with Q-Agent)

- **Acceptor** monitors condition

  $\tau < \tau_{final\ (endOfEpisode)}$

  and sets **isDone= 0|1** to go on or exit episode by **Agent**

  s_{t+1}, r_{t+1} are passed to the **Agent**

  activates Agent via event

$$\textit{ACCEPTOR EXTENSION USING VALUES OF INTEREST}$$
$$if\ (qlQ1 - qlQ2)^2 \geq diff_{max},\ then\ isDone = 1,\ else\ isDone = 0$$