Production harmonizEd Reconfiguration of Flexible Robots and Machinery

Horizon 2020 – Factories of the Future, Project ID: 680435

Deliverable 4.1

Harmonization of generic simulation and specific parameterized models into one Simulation Environment

Lead Author: SIEMENS

Dissemination level:	PUBLIC
Date:	02/21/2018
Revision:	1.2

Version history

Version	Date	notes and comments
0.1	07/20/2016	Deliverable structure, ToC , first input (Siemens)
0.2	09/27/2016	Chapter responsibilities
0.3	10/10/2016	Additional Input; Chapter Refinement
0.4	12/05/2016	Additional Chapter Refinement based on “Simulation Cluster JF” Acceptance of every partner about responsibility for their section
0.5	01/11/2017	Input from POLIMI and HSEL, adjustment in structure
0.6	01/16/2017	Input from SIEMENS, adjustment in structure
0.7	01/20/2017	Input from SIEMENS, adjustment in structure and formatting
0.8	02/02/2017	Input from TUBS and SIEMENS, adjustment in formatting
0.9	02/14/2017	Input from SIEMENS, adjustment in formatting
1.0	03/02/2017	Review Draft version, input from SIEMENS and HSEL, adjustment in formatting
1.1	03/28/2017	Final version after review feedback
1.2	02/21/2018	Amendments according to the reviewer feedback as well as answers to the reviewer comments as WORD-comments.

Author List:

Birgit Obst (Siemens)
Jan Fischer (Siemens)
Benjamin Lee (Siemens)
Wolfram Klein (Siemens)
Nils Weinert (Siemens)
Filippo Boschi (POLIMI)
Maximilian Zarte (HSEL)
Jeffrey Wehrmann (HSEL)
Lennart Bueth (TUBS)
Sebastian Thiede (TUBS)

Abstract - Executive summary

This document describes the objectives, requirements and technical concept for a harmonized generic Simulation Environment as industrial application for specific parameterized simulation models, developed within work package 4, task 4.1, of the PERFoRM project.

Chapter 1 gives an overview of the scope to analyze the dynamic behaviour of a flexible production system and to develop simulation methods and decision rules for single machines and overall production. Therefore fundamental functional requirements and constraints, regarding a generic Simulation Environment for production performance evaluation, show what will be necessary for a detailed technical concept.

Chapter 2 follows with simulation paradigms in production for simulation modelling, addressing discrete event and agent based modelling approaches for representing the production system behaviour in comparison to more continuous and high level system dynamics models.

In Chapter 3 the project pilot use case requirements, for key performance indicator (KPI) evaluation, are transferred to specific requirements for simulation based evaluation as industrial application.

Based on these requirements the methodology for the development of the Simulation Environment is described in Chapter 4. An overview of the selected conceptual architecture is introduced for coupling the Simulation Environment with the middleware.

A description of the prototypical implementation of the concept is reserved for Chapter 5. It shows the functionality of specific simulation model definition with specific simulation tools (here AnyLogic and PlantSimulation) and how to configure and execute an experiment for KPI evaluation within a generic data model and appropriate tool wrapper.

This document closes with conclusions and outlook regarding pilot specific applications in Chapter 6.

Table of Contents

1. List of Figures

2. List of Tables

1. Introduction

This chapter gives an overview of the work package 4 task 4.1 objectives and the technical approach, to what is necessary for set up of simulation methods and decision rules in overall production performance within a *Simulation Environment* (SE) and for integration into architecture as industrial application.

1.1. Objectives and scope

The key objective of work package 4 task 4.1 is harmonization, development and prototyping of simulation techniques and build up of an expandable model to analyze the dynamic behaviour of a flexible production system. This objective is achieved by three steps comprising the definition of requirement specifications, the conceptual development as well as the implementation of the developed concept into simulation applications.

By initially defining the general requirements specification for the simulation techniques including important aspects such as defining appropriate use cases and system boundaries, incorporating key performance indicators (KPIs) are established.

The second step concerned the development of a simulation concept and its techniques using existing approaches and software engines. Based on the requirements specification, this part developed the entire logic of the simulation, comprising a modular structure enabling expandability for new elements, a software in the loop system architecture, with interfaces to other tools regarding data flow, control flow and time synchronisation as well as the main “control logic” for optimization of the specified KPIs. Apart from monitoring and optimizing typical production criteria such as degree of utilization, lead times and failure safety the simulation concept will further integrate energy and resource related aspects to enable dynamic environmental assessment of the production system. Furthermore a concept is developed to provide data for the behaviour models e.g. for instance regarding the optimization of energy and resource demand, production related criteria, the integration of field device information, reconfiguration capability, internal material supply etc.

However, the key function of the “control logic” of the concept is to dynamically adapt to changes in product variants and volumes representing a production system, which can easily produce decoupled from its usually constraining cycle time. In this context the concept will further incorporate important aspects of the internal material supply for the separate production modules. To achieve the aforementioned objectives this aspect is of great relevance, since a constant supply of materials (components, modules etc.) has to be guaranteed at all times. This work step explicitly takes complex interdependencies regarding product and production structure and control (e.g. just in time, capacity and inventory control) into consideration to account for existing intra logistic and reconfigurability concepts supported by simulation.

The third step dealt with the implementation of the developed concepts and its sub modules into a SE to subsequently demonstrate and prototype the results of the concepts. At this point it is stressed that the developed simulation techniques are built on existing simulation software, PlantSimulation and AnyLogic.

1.2. General Requirements on a Simulation Environment

To evaluate the production performance regarding dynamic behaviour of a flexible production system an execution of simulation experiment is used. That means that the production system behaviour of the real in action production plant has to be modelled as a virtual mapping of the production processes and material flow by definition of plant topology, definition of machine behaviour and process parameters and definition of schedule logic. With this simulation model different parameterized scenarios, here called as experiment, can be handled. For each scenario evaluation the initial system state has to be set first. That means the initial state values for every resource as well as the production order and production schedule has to be defined. These values can be defined in different planning scenarios in an offline mode or can be integrated from actual measurement from shop floor and from ERP system in an online mode. Additionally the simulation time horizon and the evaluation mode as single run or as Mote Carlo run, taking statistical variations into account, or even as evolutionary optimization has to be defined within the experiment. To apply such an evaluation in an automated way a generic SE delivers the simulation tool wrapper with all scenario defining and evaluating functionalities for use case specification, experiment configuration and integrated execution of a simulation model using simulation engines of commercial simulation tools within the industrial IT environment of a production plant.

Commercial simulation tools for production system performance evaluation support the virtual mapping of a specific production plant by predefined machine and resource libraries with predefined process behaviour for topology and schedule definition and support in parameterizing the product order and boundary conditions. Using the simulation engine, such a specifically defined virtual simulation model representing the real production plant will be used for performance assessment and decision making. To handle these tools in an industrial environment the application needs tool independent interfaces and has to be executable by non simulation experts and assessment results should be easily available for further treatment and decision making. This leads to an integrated architecture with a simulation tool wrapper, encapsulating commercial simulation tools and enabling all activities and functionalities for input information handling regarding virtual plant set up, here also called as model generation, sequence control definition in calculation and optimization of the evaluation scenario and result access in an automated and standardized manner via middleware, which represents in this project the reference IT environment. These wrapper functionalities are described within the SE.

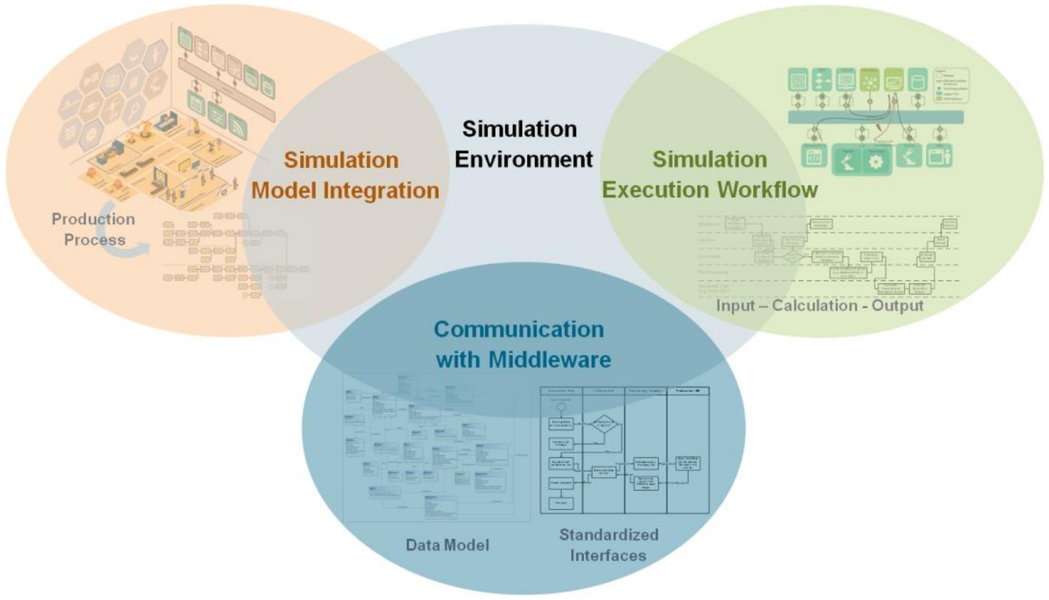

Simulation Environment
Simulation Model Integration
Production Process
Simulation Execution Workflow
Input – Calculation - Output
Communication with Middleware
Data Model
Standardized Interfaces

Figure 1: Functional modules for Simulation Environment

This section gives an overview of the general concept for the functional modules of a SE regarding use case specific simulation model integration, definition of the simulation execution workflow and communication with middleware for data connectivity and exchange.

1.2.1.Simulation Model Integration

The most important and first question to answer is, what should be evaluated by a simulation experiment. For dynamic performance evaluation of flexible production systems specific key performance indicators (KPIs) are relevant for assessment and decision support in production process reconfiguration. The relevant KPIs to be evaluated by simulation define the simulation configuration regarding planning logic, simulation detail level for model behaviour and control logic, and the model result figures.

Decision variables are the free variables, defining the simulation based KPI evaluation process, for alternative simulation experiments within scenario configuration and optimization by planning logic. These free variables might be of different characters, like machine processing skills, machine processing times or machine entrance and exit control logic as well as variants in resource availability or product order.

For behaviour modelling of production process the definition of topology as well as the flexibilities in reconfiguration delivers the requirements in dynamic simulation model configuration. Collection of machine characteristics over all use cases and their requirements in connectivity, are necessary for all topological information and interfaces for automated model set up. Additionally variants in resources and product demands define a configurable input for scenario definition and evaluation regarding scheduling aspects.

As production process simulation tools, PlantSimulation and AnyLogic, were examined regarding the configuration requirements from before mentioned modelling aspects and the claim of technology readiness level 7 (TRL7). Within the tool wrapper, data formats and files for integration of all necessary information for model generation and scenario definition have to be defined and imported, for automated simulation model set up and experiment execution within the simulation tools.

For simulation modelling regarding set up and execution, necessary configuration possibilities have to be defined to concretize:

- the output of simulation model (KPIs)
- the scenarios including decision variables
- the machine characteristics and topology as well as the variants in load (resources and product demands)

1.2.2.Simulation Execution Workflow

In addition to the definition of relevant KPI evaluation, clarification of the kind of simulation application is necessary. Examples could be:

- Offline examination and optimization in performance evaluation
- Online monitoring and optimization in performance evaluation by forecast
- Failure diagnosis by comparison with state estimation by simulation
- Predictive maintenance and service activity scheduling, etc.

The application focus delivers the requirements for process workflow integration of simulation execution for simulation as a service. Regarding application requirements the control sequence for simulation execution will be defined. This includes the simulation model set up and configuration for evaluation as well as the execution itself and the synchronized result handling.

Mechanism for process integration, with automated simulation model set up and dynamic configuration within control sequence for simulation execution, have to be defined and implemented including simulation tool wrapper functions and middleware interfaces. The sequence control functions for execution, so called trigger functions, range from model set up with topology and machine characteristic definition over state initializing using default values and current system data, connectivity presupposed, to KPI evaluation and result deployment.

1.2.3.Standardized Interfaces and communication with Middleware

Simulation execution has to be coupled with necessary data exchange (input and output) and triggering via middleware. Therefore use case applications regarding integration architecture and data delivering and collecting devices regarding data import and export, have to be defined and connected by standardized interfaces. This is independent from online or offline mode, never the less in online mode the information sources deliver actual system states and results are processed during operation; in offline mode the initial states and execution configurations are scenario-based and predefined.

Regarding input parameters (topology, resources, skills, BOMs, …) for simulation model set up and scenario configuration a definition of data model is necessary. This will be used to define data access services via middleware. The data sources could be use case specific legacy tools for data generation or storage.

Regarding output parameters (KPIs, process parameters …) of SE the definition of data model will also be used to define data allocation services via middleware. The simulation results could be used for further analytics or within use case specific applications.

The methods for data access and allocation services are defined regarding standardized interfaces within PERFoRM architecture and implemented as application programming interfaces (API) for dynamic simulation application.

1.3. Integration of Simulation Environment in PERFoRM - Project activities

The SE is one module developed within work package 4 for simulation and visualization methods to support reconfigurability. Other developed modules are e.g. the data analytics for monitoring and visualization of KPIs or the planning logic for scheduling optimization of production. All modules developed within the PERFoRM project will be linked via middleware using standardized interfaces or at least wrappers with technology adaptors. Thus for working together and serve the different pilot use cases.

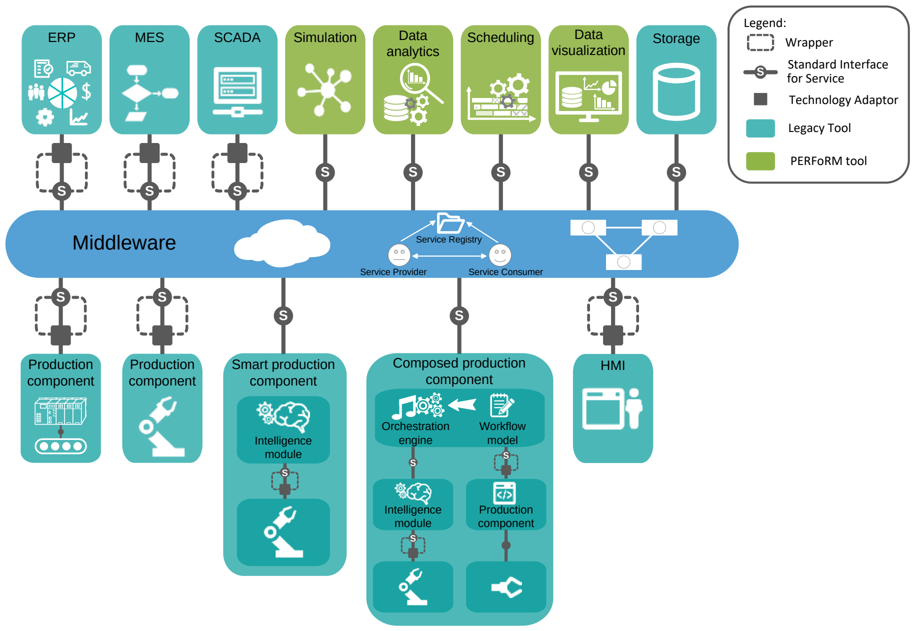


Figure 2: PERFoRM system architecture with simulation module

With this in mind, the next chapter will provide an overview of simulation paradigms in the modelling of production, and then Chapter 3 investigates functional requirements imposed by the use cases. Chapter 4 then transfers the project pilot use case requirements for industrial application to requirements in simulation execution, leading to the SE methodology. Next, in Chapter 5, the prototypical solution implementation is described, with Chapter 6 providing concluding remarks and an outlook towards future development.

2. Simulation Paradigms in Production Modelling

To simulate a real system or its behaviour, modelling can be used as a method of solving problems, which can occur during the production and wear process. In a simulation, a system can be replaced by simple digital objects, which are representing the real system and its behaviour. A complete replacement of a real system is called a model. Working with a model is practical, especially when experiments on a real system cannot be performed, either because the cost of prototyping and testing are too high or the fragility of the system does not allow extensive tests. Furthermore, real physical experiments may take too much effort and time, because the duration can be too long to be feasible. Consequently, experiments which are performed by making use of simulation models have cost related advantages in relation to real experiments with physical equipment. Therefore, simulation software can be used for balancing operating systems by experimenting with their undergoing key parameters, such as process changes of the equipment. Additionally, a simulation model is able to demonstrate long-term influences of process changes instantly, instead of waiting for results of developments in real time. Another advantage lies in the repeatability of simulation models, because many different experiments can be performed in a short time. Animations visualize the simulated process and the graphical illustration of the results enable a simple consideration of decision making due to changing parameters of the real system. Among other areas, simulation software can be used in the areas manufacturing, logistics and finances. Thus, it enables a far reaching range of experiments and the visualization of real life studies without additional costs and influences on real object [1]. As seen in Figure 3, the simulation can be categorized in three major approaches of dynamic business simulation models:

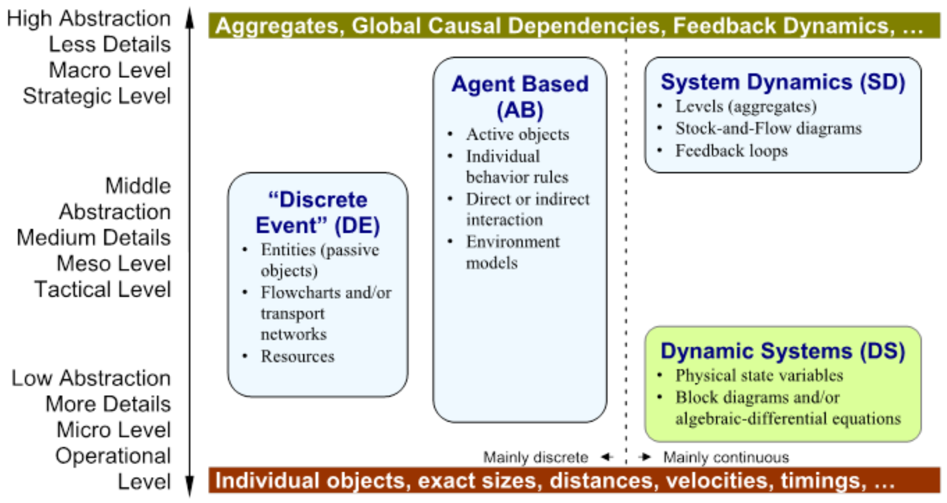


Figure 3: Simulation paradigms for manufacturing [5]

2.1. Discrete Event Simulation

Discrete Event Simulation (DES) is an established simulation paradigm where the simulation time advances in discrete time steps. It goes back to the *General Purpose Simulation System* (GPSS) developed by Geoffrey Gordon 1961 [23]. Discrete events are dynamically generated during the simulation run. At each event, the model's state variables, statistical values, or the list of upcoming events are updated (compare e.g. [24]). An exemplary DES model, in the form of a flow chart, is shown in Figure 4.

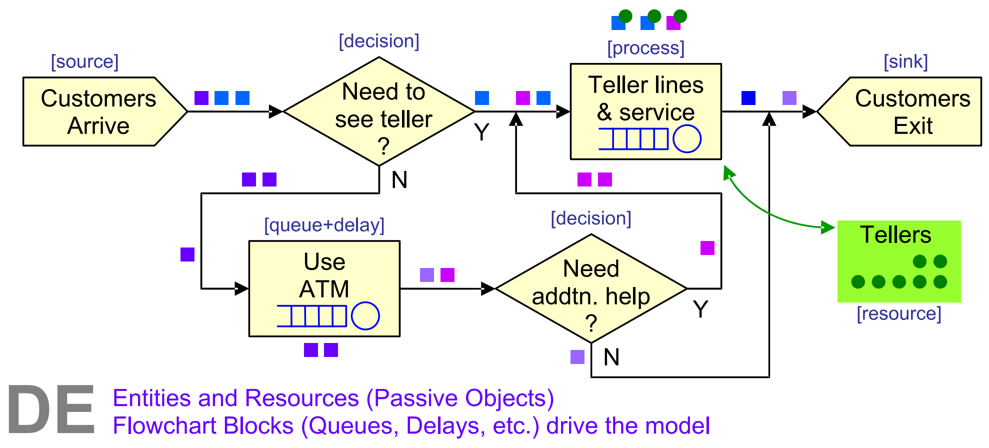


Figure 4: Exemplary Discrete Event Simulation Model [5]

In most DES models, the structural objects that represent e.g. machines, storage areas, sources, or sinks are rather static and passive and the mobile elements (e.g. products, orders, persons) flow through this structure based on a predefined logic. In such a way, DES models are well suited for modelling the systemic and logistic behaviour of factories, where the work pieces flow through the manufacturing stations, based on often simple queuing and control logics like FIFO, LIFO, or Kanban. For the model engineering, model libraries exist that contain the above mentioned elements of the DES model. Some libraries are general purpose libraries that allow the modelling of various application domains while others focus specific domains like factories, supply chains, hospitals, airports, or business processes. More than a hundred tools for DES exist ([5]) of which some famous ones are Plant Simulation, AnyLogic, Arena, Simul8, or Witness.

2.2. Agent Based Simulation

The term *Agent-Based Simulation* (ABS) describes the modelling and simulation of real systems by means of agents that interact within a simulation model [6]. While it has been mainly an academic topic limited to a few specific areas until the early 2000s, it has nowadays been adopted by simulation practitioners and is as well applied to model and simulate *Manufacturing Systems* (MS) [7], [8]. ABS allows one to gain insights into the emergent behaviour of complex systems. These systems are often characterized by non-linear interactions of locally acting connected entities that cannot be modelled in an exact mathematical way [6]. Two different general approaches can be distinguished for the simulation of MS. On the one hand, ABS can be used to model and simulate parts of or whole traditional MS like Flexible MS by representation of its components through agents. On the other hand, it can be used to simulate the behaviour of systems which are already designed on the basis of agents like Holonic MS or Matrix MS [6], [8].

Due to its emergent nature, the behaviour of a *Multi-Agent System* (MAS) as a whole is difficult to anticipate. This and the fact that ABS is the only appropriate simulation method for the simulation of MAS underline its importance. Running simulation experiments should detect and analyze system instability and non-expected patterns of behaviour. Further, as a main output of the simulation a visualization of the system should be presented [9]. As there is no agreement on the question of what an agent is and definitions vary considerably, a further look towards the properties of an agent and agent-based systems is needed to fully understand ABS [6], [10].

Russel and Norvig generally define an agent as “anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors“ [11]. Figure 5: Simple model of an agent and its interaction with the environment from [13]] visualizes the main thought behind that definition. An agent is connected to its environment by some kind of sensor and the perception of the environment by its sensors triggers an action [6]. The first key concept of the definition is thus the situatedness of the agent in its environment [10]. Wooldridge and Jennings distinguish in [13] two levels (or degrees) of agents. The first and uncontentious level, called Weak-Notion-of-Agency, includes agents that have, in addition to the situatedness, these four basic properties [13]:

- Autonomy: Agents act without any direct interventions of a user and have at least some control over their internal state and actions.
- Pro-activeness: Agents do not only react, but they take the initiative to change the environment in a goal-directed manner.
- Reactivity: An agent perceives the dynamic of its environment and responds to changes in it.
- Social ability: Agents interact with other systems or communicates by some kind of agent-communication language.

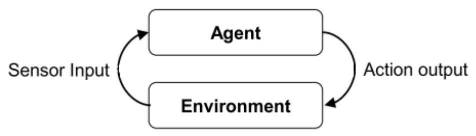


Figure 5: Simple model of an agent and its interaction with the environment from [13]

In terms of a contentious stronger notion of agency Wooldridge and Jennings describe an agent with a more specific meaning. They do not present a clear definition, but introduce the two human-like properties emotionality and mentalism (referring to knowledge, belief or intention). This understanding of an agent is especially used in artificial intelligence research [13]. Different authors add properties like rationality, meaning that the agent is always doing the right thing, and learning ability (e.g. [10] or [14]). One of the most common architectures in this context is the belief-desire-intention model building on these three components of human like reasoning [10].

As objects in object-oriented programming are entities, performing actions and having some kind of autonomy, they are often mistaken with agents [10], [12]. However, there are three main differences to distinguish between both concepts. Firstly, the level of autonomy is much higher for agents as they have control over their behaviour and not only over their state. Secondly, standard objects are lacking of flexible behaviour such as reactiveness, pro-activeness or sociality. Thirdly, each agent has its own thread of control, whereas objects are generally run through a single thread of control [10].

The topic of MAS emerged from the fields distributed artificial intelligence and distributed problem solving. Thus, it can be defined as a “loosely coupled network of problem solvers that work together to solve problems that are beyond the individual capabilities or knowledge of each problem solver“ [10]. An MAS has these four main characteristics:

- Each agent has a limited viewpoint as it has incomplete information or capabilities for solving the problem.
- Asynchronous computation.
- The data is decentralized.
- There is no global system control. [10]

An MAS can accommodate different types of agents, each specialized in its function. Agents communicate (at least partly) with the basic components of their environment and optionally some agents communicate with each other to share information or provide some service. Agents in an agent-based system might seem less intelligent than individual agents. However, due to their ability of cooperation they are capable of solving more complex problems. The proper model of an MAS builds an important building block for the ABS. It basically consists of two parts [15]:

- Model of agents: In this part the active entities are characterized as agents by their own knowledge and behaviour; moreover, the agents' architecture and their way of communication is defined.
- Model of environment: In this part the system's passive entities and their state variable are defined and generally represented by objects; further, all elements of the physical world (e.g. suppliers) and of the information world (e.g. databases) that influence but are not part of the controlled system are defined

Applications in Manufacturing Systems As mentioned before, applications of ABS for the simulation of MAS are generally used to support production planners in design of the MAS and its control architecture [16]. It helps to adjusting the definition of the agents representing parts of the MS and the interaction among them [8]. Examples for that application are:

- Barbosa et al. [8]: A washing machine production line is simulated with the SE NETLogo comparing different system configurations; the agents represent the products, the quality control stations and the resources like a WS.
- Schönemann et al. [16]: A comparison between a DMS and a MMS is done in regard to the system's utilization with and without system failures; the product and WS are represented by agents and the simulation is based on the software AnyLogic; as a result it could have been proofed that utilization for the MMS is much higher than for the Dedicated MS, especially when failures are considered.
- Vrba and Mařík [9]: Here a simulation of a physical existing packing cell for Gillette gift boxes is provided focusing on the behaviour when machine failures occur; the simulation is build up from several agents: work cell agent, conveyor agent, diverter agent (responsible for finding the least-cost product routing), robot agent, RFID reader agent, gate agent, order agent and product agent.

These examples and the most current applications of ABS in MS just cover basic agent capabilities. Learning ability or rationality are not a part of the agents abilities. Often agents are just implemented by simple objects, which do not inhere any goal-oriented representation or a reasoning module [6]. One example of a learning capability included in an ABS is presented here:

- Paolucci and Sacile [15]: The planning, scheduling and control tasks for a bike manufacturer are simulated; the jobs to be scheduled are represented by job-agents and a contract coordinator agent organizes the bidding process between the machine-agents for the job; learning is enabled by introducing an unknown cost function for the scheduling costs which is steadily improved by a non-linear optimization algorithm.

2.3. System Dynamics

The System Dynamics simulation paradigm, originally introduced by Forrester as Industrial Dynamics [25], is "the study of the information-feedback characteristics of industrial activity to show how organizational structure, amplification (in policies), and time delays (in decisions and actions) interact to influence the success of the enterprise" [26]. Consequently, its main areas of application are in the social and economic domains, evaluating the behaviour of macroscopic systems like supply chains, cities, economies, or health care systems. Since, mathematically, it can be seen as a special case of the object-oriented 1D simulation, it will not be further examined for the rest of this work. Figure 6 shows an exemplary System Dynamics simulation model of a supply chain.

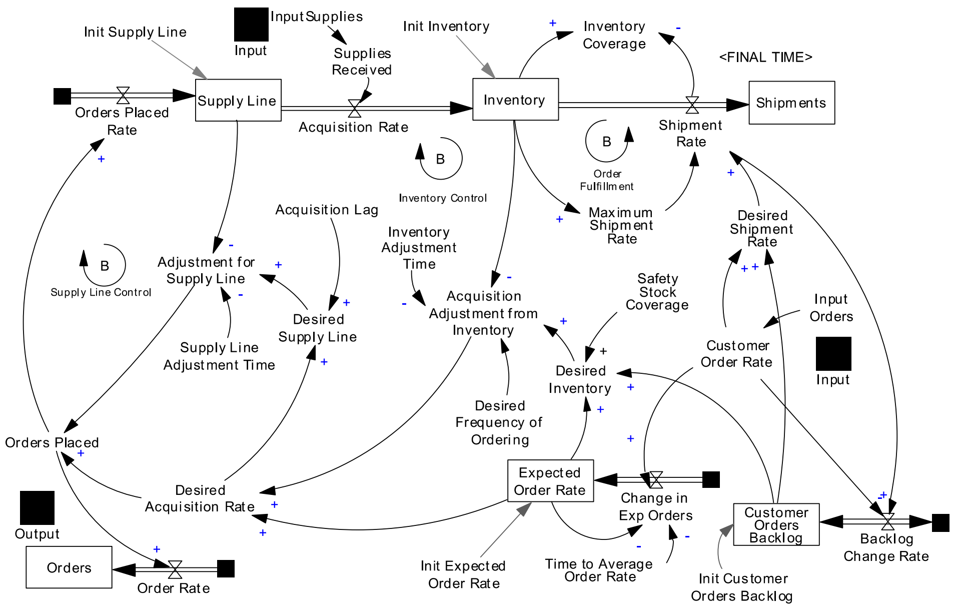


Figure 6: Exemplary System Dynamics Model of a Supply Chain [17]

2.4. Summary

This chapter has presented a short summary of the major paradigms for simulation of production systems. It has briefly reviewed some of the standard uses for simulation, in terms of which questions can be effectively answered. It thereby provides a base for the next chapter to begin the investigation of the PERFoRM use cases' needs and requirements.

3. Analysis of requirements, challenges and opportunities for each Use Case

In order to implement a simulation activity, be able to replicate the factory behaviour and be able to meet the specific needs of each use case, a preliminary analysis focused on different aspects is necessary. These aspects can be classified as:

- Input: Needed to evaluate which and how many resources are involved within use case;
- Boundaries conditions: Needed to define the simulation domain in terms of constraints, rules and stakeholders involved;
- Output: Needed to identify the final use case objective and, therefore, needed to align the simulation activity;

1. **Input:**
    a. **What are your decision variables to be calculated by simulation?**
    b. **What are your flexibilities in reconfiguration?**
    c. **What are your topology requirements?**
    d. **Which data are necessary for your use case?**
    e. **Which data are available for your use case?**
2. **Boundaries conditions**
    a. **Is reconfiguration possible?**
    b. **How often is reconfiguration possible?**
    c. **How simulation should be integrated into your environment?**
    d. **How often do you need an update of reconfiguration recommendation?**
    e. **How is data access, format, frequency, …?**
3. **Output**
    a. **What are your relevant KPIs to be evaluated by simulation?**
    b. **Do you have specific scenarios to be evaluated by simulation?**

Figure 7: Questionnaire used to obtain Use Case Simulation Requirements

As far as input identification is concerned, the potential resources are firstly split in different aspects and then analyzed in order to describe the overall domain in which each Use case is involved. These aspects are:

1. Machines: process description, including workstation identification, with main activities, dynamics and characteristics connected;
2. Worker: automation level definition and activity description for each working station that has to be performed by each worker.
3. Materials: Product description in terms of item and BoM (Bill of Material) definition;
4. Metrics: activity-based management (ABM) implementation. It is a procedure that aims at analyzing the processes of a business to identify strengths and weaknesses. The idea is to analyze the activities related to the company’s operations and to identify opportunities to improve efficiency and profitability, maximizing the value adding activities while minimizing or eliminating non-value adding activities;
5. Measures: analysis and identification of main involved resources (i.e. tool warehouse management, handling system, production planning activities, buffer availability, % scrap, OEE, utilization rate etc.).

Concerning the boundary condition investigation, the analysis has been carried out to figure out all potential stakeholders, constraints and rules of different scenarios to be simulated. Doing this, it has been possible to evaluate the feasibility, the functional and technical requirements needed and to outline the domain of each Use Case where it is possible to apply the simulation activity.

The Output analysis aims at pointing out the final objectives of each Use Case, at aligning them to simulation goals and at implanting sensitivity analysis in order to validate simulation model and result consistency.

During this analysis, the answer to several questions has been obtained leading to set up a concrete and effective simulation activity.

The use case responses for the required simulation boundary conditions are as follows:

Table 1: Boundary Condition for each Use Case

USE CASE	**Type of Simulation**	**Main decision**	**Integration**
	(Scenarios / Optimization / no simulation)	(What aspect is being tested?)	Execution workflow
Siemens	What if scenarios with Monte Carlo	How long can we postpone repairs?	Daily update triggered by production schedule and maintenance schedule (online)
GKN	What if scenarios	How to best integrate/implement the MFC concept (To prove different planning logic)	Design exploration triggered by reconfiguration and planning (offline)
Whirlpool	Potentially all three. Priority order: 1 Scenario, 2 Monte-Carlo, 3 Optimization	Is Value Stream able to satisfy production needs? Is it robust to withstand variation?	Value stream assessment triggered by ERP (offline)
I-FEVS	to be defined	When to switch production configuration (one type of vehicle to another)?	Production scheduling triggered by production order (offline or online)

The use case responses for the required simulation inputs are as follows:

Table 2: Simulation Input for each Use Case

Use Case	Simulation Input	
	Decision variable to be calculated	Topology requirements
Siemens	r= **repair probability** (based on spare parts availability, Knowledge management, fixture and tools, procurement time) p= **failure probability** (based on process parameters, failure modes including human factors) 3_**machine availability** through predictive maintenance 4_**Lead time improvement** by cycle and idle time reduction 5_**scrap rate** through process/product review	1_Machining (lathe)- 2_Hardening- 3_ Welding/Soldering+Assembly
GKN	1_ Machine availability 2_Fixture and tools, 3_Number of operator and operator availability 4_Cycle time, 5_Bottleneck, 6_Available resources 7_Change over time	1_Combination of different process that could be performed within the micro-flexible cell (e.g. Brushing+ Marking, Brushing+ Dimensional inspection) 2_A sequence of 2-3 operation steps, but it will be described in the scenarios as it will depend on specific parts/ application in the cell
Whirlpool	1_ Machine availability through organizational and preventive maintenance 2_fixture and tools, 3_number of operator and operator availability 4_Cycle time, 5_bottleneck, 6_available resources 7_change over time 8_ Type of process	1_Flow line composed of automatic station (foaming, bonding and leakage test) and manual stations (assembly)
I-FEVS	1_Fixture and tolls availability 2_Number of operator and operator availability, 3_ type of product, time unit and resources needed for specific product 4_Process time of machine for specific product, 5_Machine running time, station cycle time, job per hour 6_Shopfloor area dimension and station dimension	1_Flow line composed of automatic station (welding) and manual stations (assembly) where different product types are processed

The use case responses for the required simulation outputs are as follows:

Table 3: Simulation Output for each Use Case

USE CASE	Simulation Outputs	
	(Which KPIs should be determined?)	(Relevant KPIs to be evaluated)
Siemens	Throughput, delivery delay	MTTR, MTBF, On Time Delivery, OEE
GKN	Throughput / Lead time / OEE	Set up Cost, Set up time, OEE, WIP, Lead Time
Whirlpool	1) Lead Time 2) Ratio VA time/Lead time	OEE, C/O Time, C/O cost, WIP, Net saving, IRR, PBT
I-FEVS	OEE, Reconfiguration time (C/O time),Maximum reach, complete process time for specific product	OEE, Reconfiguration Time (C/O Time, C/O Cost, Complete process time for specific product (WIP), Net saving, IRR, PBT, Maximum reach

3.1. Siemens - Predictive Maintenance Scheduling Reconfiguration

As far as the Siemens use case concerned, the following table summarizes the final results of the preliminary analysis. Starting from this, a typical scenario has been depicted below.

Table 4: Results for predictive maintenance scheduling reconfiguration

USE CASE	Description	Type of Simulation	Main decision	Simulation Outputs	Analysis in other tools
	(5 words or less)	(Scenarios / Optimization / no simulation)	(What aspect is being tested?)	(Which KPIs should be determined?)	Is there a connection to the data analysis cluster or some other task?
Siemens	Maintenance scheduling	What if scenarios with Monte Carlo	How long can we postpone repairs?	Throughput, delivery delay	Data Analysis cluster to develop list of potential maintenance tasks.

A factory exists with a subset of machines of interest for predictive maintenance scheduling.

1. Data Analysis provides list of probable failures.
2. Data Analysis provides a list of different conditions which detect a probable depletion of remaining life of each machine. These conditions are figured out by two domains:

- Human analysis carried out by specialized operators that, relying on their expertise, define the working status of each machine through HMI utilization
- 'Smart' analysis carried out by add-on sensors and signal controllers which, evaluating specific variables (vibration, acoustic, emission, temperature, etc.), provide a clear understanding of the working machine

3. A scheduling tool will provide potential scheduling options for repairs
4. Several What-if simulations test out these various scheduling options

5. A user looks over the simulation results and picks the best schedule

3.2. GKN – Micro Fabrication Cell Reconfiguration

As far as GKN use case concerned, the following table summarizes the final results of the preliminary analysis. Starting from this, a typical scenario has been depicted below.

Table 5: Results for micro fabrication cell reconfiguration

USE CASE	Description	Type of Simulation	Main decision	Simulation Outputs	Analysis in other tools
	(5 words or less)	(Scenarios / Optimization / no simulation)	(What aspect is being tested?)	(Which KPIs should be determined?)	Is there a connection to the data analysis cluster or some other task?
GKN – 1	Cell Selection (planning)	What if scenarios with Monte Carlo	Which cells should we include in our factory?	OEE, throughput	data from process simulate to get cell performance?
GKN – 2	Switching cells (operation)	Optimizer	Should I switch the cells at a given time?	OEE, throughput	data from process simulate to get cell performance?

A factory exists, with space for 1 (or more) Micro Flexible Cells (MFCs). Any of a given set of MFCs could be installed in this space. Each MFC could provide different process combinations (Brushing, Marking, Dimensional inspection, Surface inspection) depending on strategically and operational objectives

1. The user wants to determine which MFC would be best for the current expected orders. The user wants to know which process combination really meets the market demands
2. Multiple simulations are developed to test the effectiveness of various MFCs as part of the production line. Several simulations are needed to choose the best process combination
3. The user reviews the KPIs of each simulation and decides which MFCs to install.

A factory exists, with one (or more) MFCs already installed. There are other potential MFCs available that could be installed to replace the current MFC(s).

1. The user wants to determine when to replace an MFC, and with which MFC.
2. An optimizer generates and tests the effectiveness of replacing the MFC at various times, with various MFCs.
3. The user reviews the KPIs of the optimization and decides which MFCs to install.

3.3. Whirlpool – Value Stream Mapping for Reconfiguration Planning

As far as Whirlpool use case concerned, the following table summarizes the final results of the preliminary analysis. Starting from this, a typical scenario has been depicted below.

Table 6: Results for value stream mapping for reconfiguration planning

USE CASE	Description	Type of Simulation	Main decision	Simulation Outputs	Analysis in other tools
	(5 words or less)	(Scenarios / Optimization / no simulation)	(What aspect is being tested?)	(Which KPIs should be determined?)	Is there a connection to the data analysis cluster or some other task?
Whirlpool-1	Build a simulation	Single run	Understanding relationship between KBFs -KPIs	1) Lead Time 2) Ratio VA time/Lead time 3) Bottleneck 4) Other possible problem with production engineering	Data Analysis cluster to provide real data as input
Whirlpool-2	What if?	What if scenario with Monte carlo	Understanding relationship between KBFs -KPIs	Sensitivity of various KPIs to the various KBFs	Real time data connection to statrt and plan order

A factory exists and its configuration is defined.

1. The user wants to know how his factory will respond given the current configuration
2. The user has a list of controllable KBFs (single task capacity of each workstation including Cycle Time, #parts, #set-up, Buffer size and space occupation, HR skills and #workforce) and a list of interesting KPIs (Throughput time, lead time, OEE)
3. A Monte Carlo simulation of the factory is generated to describe the expected distribution of performance, given the current configuration
4. The user reviews the results of the Monte Carlo analysis and discusses bottlenecks, other possible problems with production engineering

A factory exists and its configuration is defined.

1. The user wants to know how improvements in particular KBFs would affect particular KPIs
2. The user has a list of controllable KBFs (single task capacity of each workstation including Cycle Time, #parts, #set-up, Buffer size and space occupation, HR skills and #workforce) and a list of interesting KPIs (Throughput time, lead time, OEE)
3. A Design of Experiments is performed to analyze the sensitivity of various KPIs to the various KBFs
4. The user reviews the DoE results, and decides where to focus effort to improve KPIs

3.4. I-FEVS – Batch Reconfiguration

As far as I-FEVS use case concerned, the following table summarizes the final results of the preliminary analysis. Starting from this, a typical scenario has been depicted below.

Table 7: Results for batch reconfiguration

USE CASE	Description	Type of Simulation	Main decision	Simulation Outputs	Analysis in other tools
	(5 words or less)	(Scenarios / Optimization / no simulation)	(What aspect is being tested?)	(Which KPIs should be determined?)	Is there a connection to the data analysis cluster or some other task?
I-FEVS	Batch scheduling	Scenarios	When to produce which products to meet overall demand	OEE, throughput, delay in delivery	

A factory setup exists, with a list of current orders. The machines can be reconfigured to manufacture multiple types of vehicles.

1. The user wants to plan batches of vehicles to be produced and their configuration line in order to meet deadlines;
2. A centralized scheduling tool proposes various schedules also to understand if a machine asked to produce a new type of vehicle must be firstly reconfigured to perform this task;
3. A set of simulations are generated to analyze the performance of the factory under each schedule;
4. The user reviews the results and determines which schedule to accept.

3.5. Requirements for enabling industrial level simulation software for Agent-based Simulation

Agent-based Simulation (ABS) has been adopted by simulation practitioners to simulate complex coherences in manufacturing systems, as described in the initial chapter about Agent-based Simulation. Especially with the trend of flexible manufacturing, promoted through industry 4.0, ABS can support the evaluation of complex situation, when planning and/or scheduling production within flexible manufacturing systems.

To apply ABS in manufacturing supporting tools are necessary. As today most manufacturing system specific simulation tools are based on discrete event simulation. Those industrial simulation tools offer a hands-on user interface and thus enable the industry to model situation in an efficient way. Tools that are normally used to do ABS are more general-purpose simulation tools and are mostly used in research due to their complex application characteristics. Thus, in order for the industry to be able to use ABS and its advantages, they must be able to apply ABS within commonly used industrial simulations tools (see Figure 8).

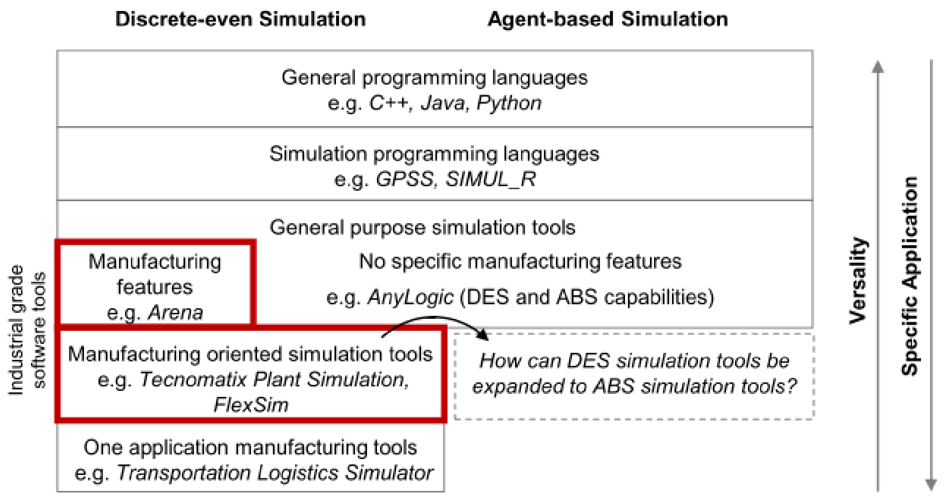

Discrete-even Simulation
Agent-based Simulation
General programming languages e.g. C++, Java, Python
Simulation programming languages e.g. GPSS, SIMUL_R
General purpose simulation tools
Manufacturing features e.g. Arena
No specific manufacturing features
e.g. AnyLogic (DES and ABS capabilities)
Manufacturing oriented simulation tools e.g. Tecnomatix Plant Simulation, FlexSim
How can DES simulation tools be expanded to ABS simulation tools?
One application manufacturing tools e.g. Transportation Logistics Simulator
Industrial grade software tools
Versality
Specific Application

Figure 8: Classification of simulation software

3.6. Additional Requirements

In addition to providing specific information about each use case, the requirement engineering which has been carried out during Task 1.2 has also provided the following requirements which should be taken into account during the implementation of the simulation activities:

- Feedback from simulation to design should be provided in order to optimize the product and the process design
- Simulation and prototyping activities should be performed in the CPPS environment
- Different information systems could be adopted in various process (i.e Quality and Maintenance) in order to allow different department to collaborate with other departments facilitating process interaction
- The simulation process should be able to collect data concerning different department, in order to manage information in a efficient way
- The model of the system should have a user-friendly interface in order to facilitate its usability
- Different information systems could be adopted in various process
- The working environment should be taken into account in order to plan and design different scenarios where ergonomics, health and safety could be guarantee.

3.7. Summary

This chapter has presented a review of activities focused on elicitation of use case requirements for simulation. Questionnaires were carried out with the use case partners in order to understand the scenarios in which simulation could provide value. These scenarios were then more deeply investigated in order to examine the available inputs, required outputs, and boundary conditions in order to obtain a description of the requirements by the use cases for simulation. In the next chapter, these functional requirements are more fully decomposed into technical requirements restricting the concept of the SE.

4. Defining the Simulation Environment Concept Architecture

In this section, the methodology for the development of concept architecture of the SE is described. In subsection 4.1, functional requirements for the SE are further analyzed in order to derive technical requirements that may not have been considered in the use-case scenarios. Then, in section 4.2 the requirements are consolidated in order to provide a concise basis for the development of architectural concepts (section 4.3). Finally, in section 4.4, the SE concept is selected and described. A description of the implementation and development of this concept is then reserved for Section 5.

4.1. Derivation of Technical Requirements

The use case requirements developed in Sections 3.1 – 3.4 resulted in a list of the required functional capabilities, but leaves unconstrained the technical implementation of how these capabilities may be achieved. This section will therefore focus on the derivation of more specific requirements for how the desired capabilities may be achieved.

To begin this task, simulation cluster participants reviewed the required Data Inputs, Data Outputs, Simulation Types, Triggering mechanisms, simulation type, required paradigms, and expected frequency of execution of the simulation cases. For each aspect, the participants reviewed possible restraints on the functionality. The results of this analysis are shown in Table 8 below.

Table 8: Simulation model configuration – input, output and evaluation types

		Use Case			
		Siemens	**IFeVs**	**Whirlpool**	**GKN**
Data Inputs	Production Schedule	🟩	🟩		🟩
	Maintenance Schedule	🟩			
	Control Logic				🟩
	Factory Topology		🟩		🟩
	Parameter / States	🟩	🟩	🟩	🟩
	Legacy Models				*
Data Outputs	Machine Level KPIs	🟩	🟩	🟩	🟩
	Product Level KPIs	🟩	🟩	🟩	
	System Level KPIs	🟩	🟩	🟩	🟩
	Optimal Topology		🟩		🟩
	Optimal Parameters		🟩	🟩	🟩
	Optimal Schedule	🟩	🟩		🟩
Triggering Mechanism	Manual	🟩	🟩	🟩	🟩
	By Time				
	By Event	🟩	🟩	🟩	🟩
Simulation Types	Single Run		🟩	🟩	
	Monte Carlo	🟩		🟩	🟩
	Optimization				
Simulation Paradigm	DES	🟩	🟩	🟩	
	ABM		🟩		🟩
	SD				
Frequency of Execution (Expected)	>1 per Shift				
	~1 per Shift		🟩		
	~ 1 per Day	🟩		🟩	
	< 1 per Day				🟩
	Once				*

By reviewing this analysis, some key aspects become apparent. First, the GKN use case involves two potential cases for simulation – th first of which focuses highly on the planning and design phase, and is expected to be completely manually executed. Further, it will rely heavily upon legacy models already used. These simulations focus on range of freedom and contact detection rather than a system level material flow simulation. These types of simulations, while of definite interest in the reconfiguration of machinery or in the planning of new machinery, are beyond the scope of implementation within the targeted SE. As such, they are not included beyond this step. However, the second potential case in the GKN use case focuses on using simulation models within an operation loop to identify the proper times to perform switching operations from one micro cell to a different unit. The models to be used for this optimization are being defined within Tasks 4.2 and 4.4, and involve the modelling of each individual cell functional unit as an individual agent.

Also, each use case utilizes simulation in a different manner, to their own end, but each supports decision making either directly, through automated optimization, or indirectly, by providing simulative KPI analysis over a set of potential decision architectures.

Furthermore, the use case responses regarding the simulation paradigm show the pragmatic focus of the use case owners. No use case owner provided a strong opinion on the underlying methodology to be selected, rather they tended to focus upon a holistic "solution" that meets their needs. To that end, when cluster participants discussed the use cases to identify the required paradigms, the goal was to identify the minimally viable paradigm capable of meeting the use case owners' demands. For the Siemens and Whirlpool use cases, which to a large degree involve traditional flow manufacturing processes, the DES paradigm was designated as sufficient. However, the GKN and IFEVS use cases focused on novel architectures for manufacturing that are already at most partially implemented. This left some freedom remaining in the design, construction, and implementation of such systems. It also means that traditional material flow simulation tools may not be adequate to address the complexity in these use cases. Thus, as the goal of the Simulation Environment is to provide a single integrated tool in which each of these paradigms could be utilized, it is sufficient here to define that both paradigms should be supported. The specific modelling details of how each use case is modelled is then left for Tasks 4.2 and 4.4, as well as the respective work packages.

The next section reviews the technical requirements described above, and consolidates the requirements of the individual use cases into a single set of requirements on the SE.

4.2. Consolidation of Requirements

In consideration of the functional use case requirements from Sections 3.1-3.4 and the technical use case requirements identified in section 4.1, a concise list of core requirements can now be provided. This list consists of the sum of the functionalities required by the various use cases, and is shown in Table 9 on the next page.

4.3. Simulation Environment Architecture alternatives and choices

Considering the requirements introduced in the prior section, it becomes clear that the SE must accomplish the following three primary tasks:

1. Get simulation inputs describing the factory
2. Manage individual simulations of the factory
3. Return simulation results

In the next subsections, these primary tasks are further decomposed in order to result in a feasible concept. The resulting concepts will then be combined to compose a complete SE, which will then be described in section 4.4.

4.3.1.Get simulation inputs describing the factory

From the stated requirements in Table 9, SE.I1, 4, 10, 12& 13, SE.T1 - 2, and SE.P1 - 2 correspond directly to the specification of the portion of the SE that deals with obtaining inputs from the middleware. The concept for obtaining inputs must be capable of:

1. Accepting the desired types of inputs (schedules, control logic, topologies, and parameters/states)
2. Triggering the next stage of the SE once prompted manually or via events,
3. Processing models of different paradigms.

SE.I1, 4, 10, 12, 13 concern inputs that may be dynamic in nature. For example, planned production schedules will change over time, as products are completed and delivered, or as demands change and production changes to keep pace. Therefore, it must be possible to obtain these inputs at the time of simulation – they cannot be manually "loaded" once and then utilized indefinitely. To this end, one of the stated goals of PERFoRM is the development of architecture (see Figure 2) that allows the exchange of information between services in a flexible manner. In order to successfully fulfil these requirements, the SE should include an interface to the middleware that enables transmission of the required information at the time the simulation service is requested.

Table 9: Consolidated Simulation Environment Technical Requirements

Category		Req ID	Requirements: "The SE must be able to…"
Data Inputs	Production Schedule	SE.I1	… Accept a production schedule as an input.
		SE.I2	… Simulate production according to the Production Schedule.
		SE.I3	… Provide meaningful simulation results, even if no production schedule is provided as an input.
	Maintenance Schedule	SE.I4	… Accept a Maintenance schedule as an input.
		SE.I5	… Simulate the effect of failure in a machine / topology.
		SE.I6	… Provide meaningful simulation results, even if no maintenance schedule is provided as an input.
	Control Logic	SE.I7	… Accept the Control Logic of a machine / topology as an input.
		SE.I8	… Execute the Control Logic of a machine / topology.
		SE.I9	… Provide meaningful simulation results, even if no Control Logic is provided as an input.
	Factory Topology	SE.I10	… Accept the definition of the potential topology of a factory as an input.
		SE.I11	… Provide meaningful simulation results, even if no topology is provided as an input.
	Parameter / States	SE.I12	… Accept the pre-simulation state of the factory as an input.
		SE.I13	… Accept a definition of potential topology as an input.
		SE.I14	… Provide meaningful simulation results, even if no status is provided as an input.
Data Outputs	Machine Level KPIs	SE.O1	… Calculate meaningful KPIs at the machine level.
	Product Level KPIs	SE.O2	… Calculate meaningful KPIs at the product level.
	System Level KPIs	SE.O3	… Calculate meaningful KPIs at the system level.
	Optimal Topology	SE.O4	… Calculate a scoring metric for a set of provided potential topologies.
	Optimal Parameters	SE.O5	… Calculate a scoring metric for a set of provided potential parameters.
	Optimal Schedule	SE.O6	… Calculate a scoring metric for a set of provided potential schedules.
	Export Outputs	SE.O7	… Make available the KPIs and optimal parameters as an output
Triggering Mechanism	Manual	SE.T1	… Be triggered manually by the user, to run with desired inputs.
	By Event	SE.T2	… Be triggered automatically by other tools, to run with specified inputs.
Simulation Types	Single Run	SE.S1	… Perform a single execution of a material flow simulation for the provided inputs.
	Monte Carlo	SE.S2	… Perform a Monte Carlo analysis involving several material flow simulations for the provided inputs.
Simulation Paradigm	DES	SE.P1	… Perform a simulation that models the factory using the DES paradigm.
	ABM	SE.P2	… Perform a simulation that models the factory using the ABM paradigm.
Frequency of Execution	~1 per Shift	SE.F1	… Perform simulations sufficiently fast that they could be executed at least once per shift.
	< 1 per Day	SE.F2	… Perform simulations with sufficient horizon that they must not be executed daily.

SE.T1 & 2 concern the mechanisms by which a simulation may be triggered. Specifically, that it must be possible for other tools to trigger the SE. Here, again the interface to the middleware provides a way for outside clients to request service. In principle, the middleware developed within the scope of the PERFoRM project will support various protocols for accessing services [18]. Similarly, the SE will also support these protocols, to the extent possible. As such, the SE should, as part of its interface to the middleware, be able perform standard actions for web services, such as GET, POST, PUT, DELETE etc.

SE.P1 & 2 and SE.I7 concern the capability of the SE to perform various kinds of simulations. In order to be able to perform such simulations, it must be possible to obtain models of the individual units to be included in the simulation. Whereas the first set of inputs described is likely to be dynamic, here, the set of control logics is not likely to change significantly during nominal operation. This is not to say that a machine will always utilize the same control logic over its lifetime, rather it is probable that as machines are reconfigured, they will also utilize different control logics. However, this set of available control logics would still remain constant. As such, the SE does not require a dynamic interface for the inclusion of control logics. Rather, the SE should instead allow special users to be able to integrate externally developed control logics into the simulation base.

4.3.2.Manage individual simulations of the factory

From the stated requirements in Table 9, SE.I2, 3, 5, 6, 8, 9, 11, 13, & 14, SE.O1 – 6, SE.S1 - 2, SE.P1 – 2, and SE.F1 – 2 correspond directly to the specification of the portion of the SE that deals with managing individual simulation experiments within the SE. The concept for managing these simulations must be capable of:

1. Create a meaningful simulation that uses the specified inputs
2. Execute simulation of the factory
3. Calculate KPIs and other outputs of interest

SE.I2, 3, 5, 6, 8, 9, 11, 13, & 14 concern the capability of the SE to convert the inputs described in the previous section into a meaningful simulation of the factory. It is important to note that the inputs described in the previous section include both those that are static as well as those that are dynamic. Static inputs will be provided upon initialization and configuration of the SE – which, as the previous section indicated, would involve the definition of, for example, Control Logics and allowable units. As these units and logics will already be defined within the SE upon triggering of the simulation, their inclusion requires the simulation manager to include the ability to both store this information and for this information to be implemented upon execution. This essentially requires the inclusion of an internal model library within the SE. Additionally, the SE must be able to account for the dynamic inputs that only become available when triggered by the middleware. The previous section described that this information would be made available within the SE via its interfaces. In this section, we expand the focus to consider that the inputs will arrive from the middleware in some PML-compliant format (for example in an XML file or as JSON objects). This allows the inputs to be provided in a tool-and vendor-independent format, but means that a transformation must first occur such that the individual tools (for example: Siemens Tecnomatix PlantSimulation or AnyLogic) are able to parse the inputs. This necessitates the inclusion of transformation module, which may be tool-dependent.

SE.P1 – P2 concern the capability of the SE to perform simulations that follow different paradigms. Depending on the factory being modelled, it may be advantageous to include or not include agents, and therefore the SE should support both formalisms. This will be achieved by supporting multiple tools which can then support different formalisms and be selected upon configuration.

SE.S1 – 2 concern the capability of the SE to perform either single executions of a simulation or a Monte Carlo-style sampling-based uncertainty propagation of the system. The tools under consideration for development within the context of the PERFoRM project have integrated capabilities to already perform such analyses, and as such no additional new functionalities must be integrated here.

SE.O1 – 6 concern the capability of the simulations within the SE to perform specific calculations, and serves to restrain the set of simulation tools that are useful to integrate. The tools considered within this document (Siemens Tecnomatix PlantSimulation and AnyLogic) are both capable of producing these outputs.

SE.F1 - 2 concern the amount of time that a simulation will require to execute, and serves to further constrain the set of simulation tools that are useful to integrate. The simulations must be sufficiently detailed that they are capable of producing feasibly accurate predictions over longer horizons, but still be executed sufficiently quickly such that their results can be obtained soon enough to be useful. Again, here the tools considered within this document have been found to meet these requirements.

4.3.3.Return simulation results

From the stated requirements in Table 9, SE.O7 corresponds directly to the specification of the portion of the SE that deals with exporting the results of the SE. The concept for exporting these results must be capable of:

1. Defining the set desired outputs to be exported
2. Exporting the set of desired outputs to the middleware

SE.O7 concerns the ability of the SE to distribute the results of the simulation to the querying clients. This necessitates two steps, first, that the desired KPIs be defined, and second, that these calculated KPIs be exported via the same interface utilized to obtain inputs via the middleware. The first step will be accomplished during the configuration of the Simulation Event, as the set of classes of KPIs to be requested are not expected to be varying with time. Note that this does not mean that the specific instances of these KPIs will not change. Rather, consider that when a new machine of a given type is connected to a PERFoRM system, that this means that the set of KPIs generated describing the performance of this machine will be the same as those describing another machine of the same type. The second step will again be accomplished using a transformation module that will convert the tool- and vendor- dependent output language into a PERFoRM-compliant format to be exported via the standard interface.

4.4. Concept Overview

Figure 9 below provides an informational view of the proposed architecture concept for the SE. The informational view displays the main characteristics of the architecture of the SE:

1. Specific Simulation Wrapper
2. Simulation ← → PML translator module
3. Middleware interface
4. Extendable model library database for storing static inputs

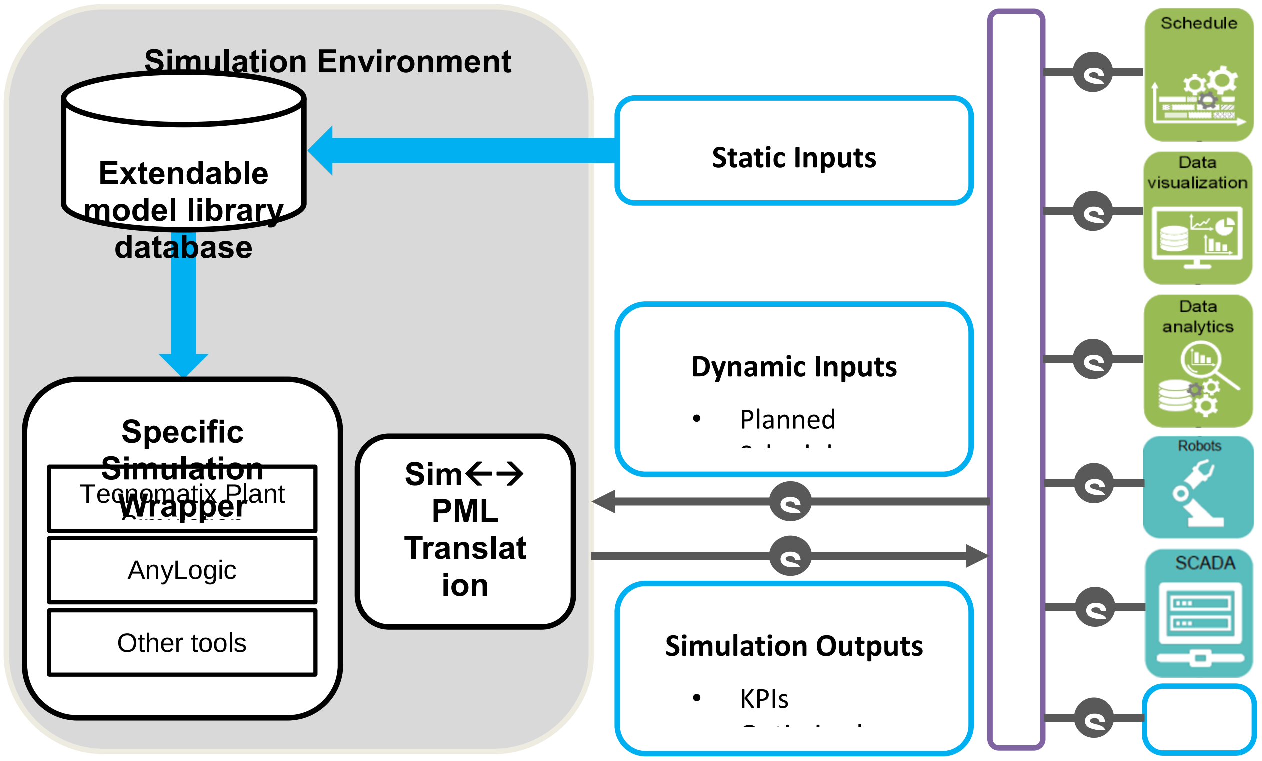


Figure 9: Simulation Environment Informational View

4.4.1.Specific Simulation Wrapper

The Specific Simulation Wrapper has the purpose of performing any direct interactions between the simulation tool and the other elements of the SE. The Specific Simulation Wrapper is responsible for accessing the correct model from the Extendable Model Library Database, and passing this to the simulation tool, beginning the execution of the simulation task. Upon completion of the simulation task, the Wrapper then passes the output of the simulation to the Simulation←→PML Translation module for further processing.

4.4.2.Simulation ← → PML translator module

The Translator module has the purpose of converting the PML-compliant inputs received into a format that the simulation tool can receive and interpret. It is also responsible for translating the outputs of the simulation tool into a PML-compliant format so that the Middleware interface can return the simulation results to requesting client.

4.4.3. Middleware Interface

The Middleware Interface has the purpose of recognizing when a simulation has been requested, accepting the inputs in a PML-compliant form, passing these inputs onto the Translation module, accepting PML-compliant outputs from the Translation module, and passing these on to the middleware.

4.4.4. Extendable model library database

The most complicated module within the Simulation is the Extendable Model Library Database, which has the purpose of storing the definition of static elements to be used within the simulations. This library is populated upon initial configuration of the SE, and includes definition of the classes of production equipment, products, factory constraints, control logics, and other elements that, once defined, may be instantiated at the time of simulation. As the elements within the library database must be instantiable within the Specific Simulation Wrapper, these elements are most likely to be created using the native format of the tool which is to be used to perform the simulation. For example, within Tecnomatix PlantSimulation, the internal language available for programming is SimTalk, so the models and methods within the library database could be primarily defined in SimTalk. However, it is possible to also utilize models defined in other languages (e.g. DLLs) to prescribe the behaviour of elements. Section 5.7 provides a description for how this could be done to enable agent based simulation (ABS) within industrial discrete event simulation (DES) software.

4.5. Summary

In this chapter, the methodology for the development of concept architecture of the SE was described. First, functional requirements for the SE from Chapter 3 were analyzed in order to derive technical requirements. Then, these were consolidated and then used as the basis for the definition of a concept for the architecture of the SE. In the next chapter, the implementation of this concept is described, with a deeper emphasis on the implementation of the SE as software.

5. Implementation and Solution

5.1. Structure

In order to allow the execution of the introduced SE, a detailed implementation concept was developed. Figure 10 shows an overview of the internal structure of the SE. Its structure is based on various case-specific or generic components. The components can be clustered in five categories:

1. An **Interface to the Middleware** that receives and sends data requests and processes the incoming and outgoing data (see also Interface Section below). The interface is directly connected to the coordinator that then evaluates and further processes the incoming or outgoing service request or response. All further implementation aspects in regards to the interface are described in the respective deliverable D2.3.
2. A **Coordinator** that interprets the data that was received from the interface and calls the individual sub-components, services, or simulation tools of the SE (e.g. data-conversion, simulation-runs, pre- or post-processing …). The coordinator is thus the main component of the SE and also utilizes decision logics and workflow descriptions in order to connect, combine and execute the components. Hence, the main task of the coordinator is to connect the incoming service requests with the workflows stored in the workflow storage (see below) and then to execute the respective workflow.
3. **Sub-Components or Services** provide generic functions on a rather high level to the SE e.g. in order to post- or pre-process the incoming and outgoing data or to convert the PERFoRM-ML based XML models into simulation tool specific data models (XML-Conversion). In addition a user interface can be provided in order to allow an online or offline configuration of the SE or in order to show results or the current status of the simulation runs. Furthermore, a co-simulation master is provided in order to allow the parallel and synchronized simulation of various simulation runs, e.g. material flow simulations on production system level can be coupled with detailed physical simulations on process or machine level and thus need a master. One possibility to conduct co-simulations is described by the FMI/FMU standard.
4. Various **Libraries** exist in order to provide tool- and non-tool-specific functionalities, e.g. simulation model components in order to build a simulation model (e.g. process, workstation, conveyor etc.; compare also section 5.6.1). In addition, base models of simulation tools are stored here, in order to allow a partial automatic model generation from a partially existing simulation model that thus only needs to be adapted or parameterized for the specific use case. The libraries should be modelled as tool-independent as far as possible in order to allow a reuse over several tools. Such tool independent and open simulation component descriptions are for example provided by the Modelica standard, albeit for the purpose of 1D-multiphysics simulations and not for factory simulations as intended in this case. However, not every tool allows this modelling, so it is sometimes inevitable to use tool-specific modelling languages and file formats. In the libraries also the various agent and non-agent based planning logics and control mechanisms described in section 4.4.4 are contained, either as executable base models, as interpretable source code, or as encapsulated .dll. Further libraries include the specification file conversion specifications in the form of xslt style sheets that define how the generic input files need to be transformed in order to be interpretable by the respective simulation tool that is utilized. Finally, one of the most important

libraries is the Use-Case/Service specific workflows library. In each workflow it is specified what sub-components/services and what simulation tools need to be executed in order to provide the output to a higher level service or API-function that is provided by the SE. Consequently, a workflow consists of a sequence of certain execution steps and their parameters that need to be conducted (e.g. 1. Pre-processing, 2. File Conversion 3. Simulation Model Generation ... 8. Send results back to Requesting Service). Furthermore, case specific evaluations of these execution sequences can be contained e.g. in order to make the execution of a certain function dependable on the outcome of the result of a previous function (e.g. post-processing may only be required if the simulation results are of a certain quality type). Consequently, workflows can be depicted as activity diagrams as described in the following section. They are stored in an open xml format that can be edited by a user via the GUI or can be added via the API.

5. Finally, **Tool Specific Functionalities** are provided mainly in order to conduct the actual simulation runs. Models are also generated inside the simulation tools if the respective functionality is provided by the tool. E.g. the software Plant Simulation allows the script based execution and generation of simulation models and thus provides the required functionality.

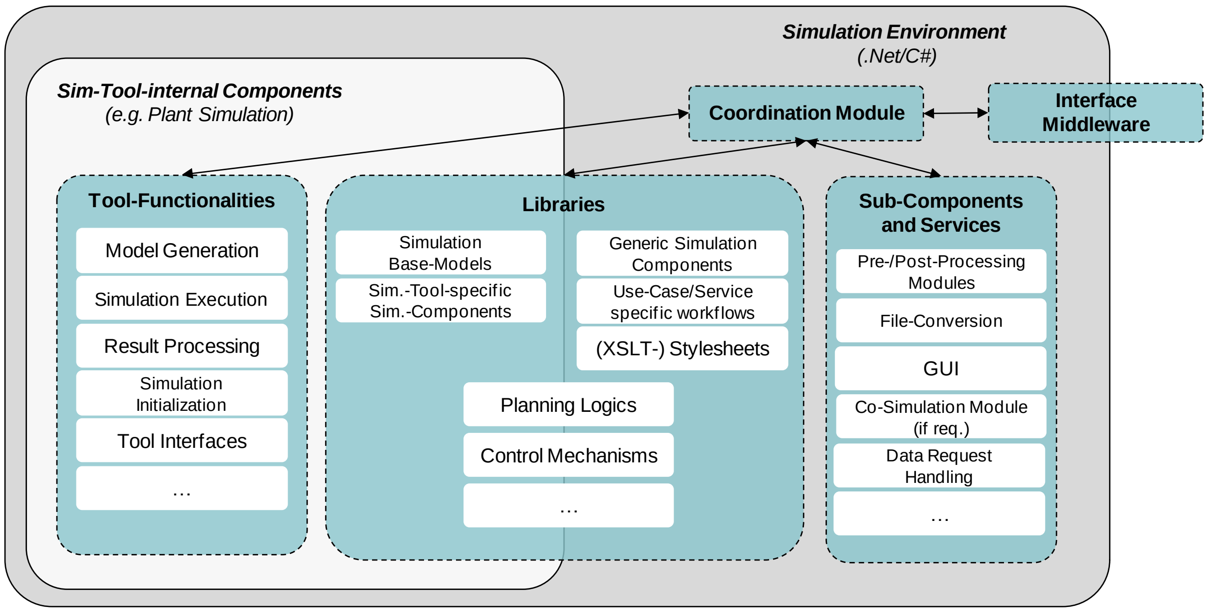


Figure 10: Simulation Environment – Internal View

5.2. Workflows and Behaviour

In addition to the structural view of the SE components, their behaviour can be visualized in activity diagrams that depict the logical and time-based interactions of the components. The components are described in these views as so-called swim-lanes. Figure 11 visualizes an exemplary workflow where a simulation request is received from the middleware and forwarded via the interface to the coordination module. After a validity check of the request, the coordination module generates the required sub-calls and initiates the pre-processing modules (if required) and the simulation itself. Afterwards the required simulation model is updated or generated and the simulation is executed. The results are then sent back to the middleware via the standard interface. In addition to this example, several alternative workflows exist based on the respective simulation request and simulation tool, however their respective workflow will look similar to the one specified in Figure 11. In the remainder of this section, each of these sections will be more deeply investigated.

- Define Simulation Elements (Simulation Expert)

In this preprocessing step, a simulation expert is required to define default elements that describe the entities that will be simulated within the factory context. This consists of defining the types of production equipment that will be utilized, the product types that can be produced, the processes that can be executed using this equipment on these products, the personnel that perform or oversee these processes, and the parameters that define them. Upon initial implementation, this step coincides with the definition of the PML description of the factory. If no legacy models exist, or if no special considerations or models are to be included, then this step requires no additional investment of work than that already required to model the factory components in PML description of the factory.

- Store in Default Model (Model Library)

In this step, the simulation elements described in the section above are stored in a model library, such that they can be recalled upon request.

- Send Simulation Request (External Tool)

In this step, an external tool or user has decided to request a simulation. To do this, the external tool compiles a PMLSimulationResult object that describes the factory state to be simulated, including any planned schedules and configurations to evaluated, as well as a list of KPIs by which they should be evaluated.

- Transfer request to SE (Middleware)

In this step, the middleware receives a request from the external tool, and passes this request onto the SE, which is identified as a registered service.

- Receive Simulation Request (Interface)

In this step, the Interface recognizes that a request has been sent to SE. Before processing this request, the Interface passes the PMLSimulationResult object to the Coordinator, to test whether it forms a valid base for a simulation.

- Check if Valid (Coordinator)

In this step, the Coordinator tests whether the requested PMLSimulationResult object is valid. If the object does not contain enough information to fully specify a simulation, then the Coordinator queries whether a fully defined default model in the model library exists. If not, then an error message is generated and sent to the Interface. Otherwise, the valid PMLSimulationResult is sent to the Post-Processing Module.

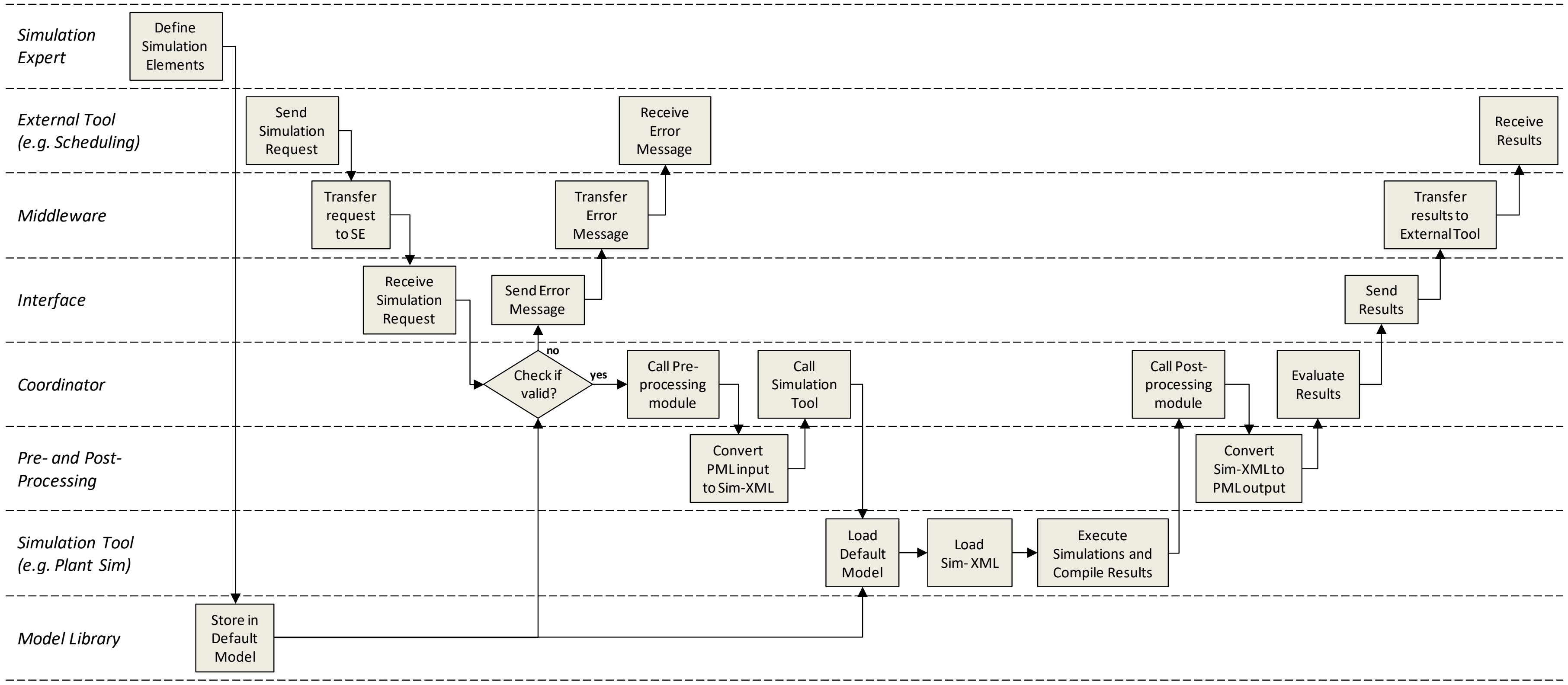


Figure 11: Configuration and Execution of Simulation Environment

- Call Pre- and Post-Processing Module (Coordinator)

In this step, the Coordinator calls the Pre- and Post-Processing module and passes the valid PMLSimulationResult to be translated.

- Convert PML Input to Sim-XML (Pre- and Post-Processor)

In this step, the Pre- and Post-Processor converts the valid PMLSimulationResult into a simulation tool-neutral Sim XML format using XSLT. Upon completion, the transformed Sim XML is made available to the Coordinator.

- Call Simulation Tool (Coordinator)

In this step, the Coordinator calls the Simulation Tool, and triggers the tool to load the default model, which was defined by the Simulation Expert.

- Load Default Model (Simulation Tool)

In this step, the Simulation Tool opens the default model, which was defined by the Simulation Expert and contains only static information about the factory. This model must now be updated to include the dynamic information included in the Sim-XML file. Upon loading the default model, the Simulation Tool triggers the simulation model to parse the Sim XML file.

- Load Sim-XML (Simulation Tool)

In this step, the Simulation Tool parses the Sim-XML file, which has been translated by the Pre- and Post-Processing Module and includes the dynamic information about the factory that has been requested to be included in the simulation. If entities in the simulation must be updated, or if new entities (such as new product orders) must be created, then these processes occur now automatically. Upon completion of this process, the simulation model is ready to be executed.

- Execute Simulations and Compile Results (Simulation Tool)

In this step, the simulation model is executed. If a Monte Carlo style analysis has been requested, then a number of simulations are executed and then, for the KPIs requested, average, best, and worst values are calculated. The results are then compiled and written out as a Sim-XML file.

- Call Post-Processing Module (Coordinator)

In this step, the Coordinator has been notified that the Simulation Tool has completed execution and passes the Sim-XML to the Pre- and Post-Processor to be converted into a PML Output.

- Convert Sim-XML to PML Output (Pre- and Post-Processor)

In this step, the Pre- and Post-Processor converts the simulation tool-neutral Sim XML file into a valid PMLSimulationResult format using XSLT. Upon completion, the transformed PMLSimulationResult is made available to the Coordinator.

- Evaluate Results(Coordinator)

In this step, the Coordinator checks whether the PMLSimulationResult is a valid expression. If so, these results are passed to Interface. Otherwise, an empty PMLSimulationResult object is sent to the Interface.

- Send Results (Interface)

In this step, the Interface sends the valid PMLSimulationResult to the Middleware, to be passed on to the querying service or user.

- Transfer Results to External Tool (Middleware)

In this step, the Middleware receives a the PMLSimulationResult from the SE, and passes this object along to the requesting External Tool.

- Receive Results (External Tool)

In this step, finally the results are received by the External Tool, and can be further processed as desired within this tool.

5.3. PERFoRM-ML Data Model – Simulation Aspects

As will be described in the next section, the standard interface to the middleware provides functionality to exchange information between the middleware and services, utilizing the PERFoRM Markup Language (PML). Figure 12 below shows the current specification of PERFoRM-ML [20]. In this section, aspects of PML that are of key importance to the SE are reviewed.

The PMLSimulationResult class has been introduced specifically to address needs of the SE. Specifically, when a simulation is to be requested – an instance of the PMLSimulationResult class supplies information about the desired Start and End times for the simulation, an enumeration of the KPIs for which values should be simulated, as well as a set of configurations or schedules that should be evaluated in order to produce these KPIs. Additional information, for example, the type of simulation task to be executed is then included as attributes of the class.

The SE is able to integrate products, equipment, personnel, processes, planned schedules, and system configurations, and therefore the corresponding PML class definitions of PMLProduct, PML Entity, PMLOperation, PMLSchedule, and PMLConfiguration are of general interest to the development of the SE. However, these aspects have already been explored in sufficient detail in their specification in Deliverable 2.3 [20] and so they will not be further discussed here. On a related note, as the PMLParameter is utilized as mechanism by which generic information can be transmitted via the standard interface, the definition of an enumeration of attributes necessary for the definition of these classes is provided in the XSLT, as described in sections 5.5.4 and 5.6.4.

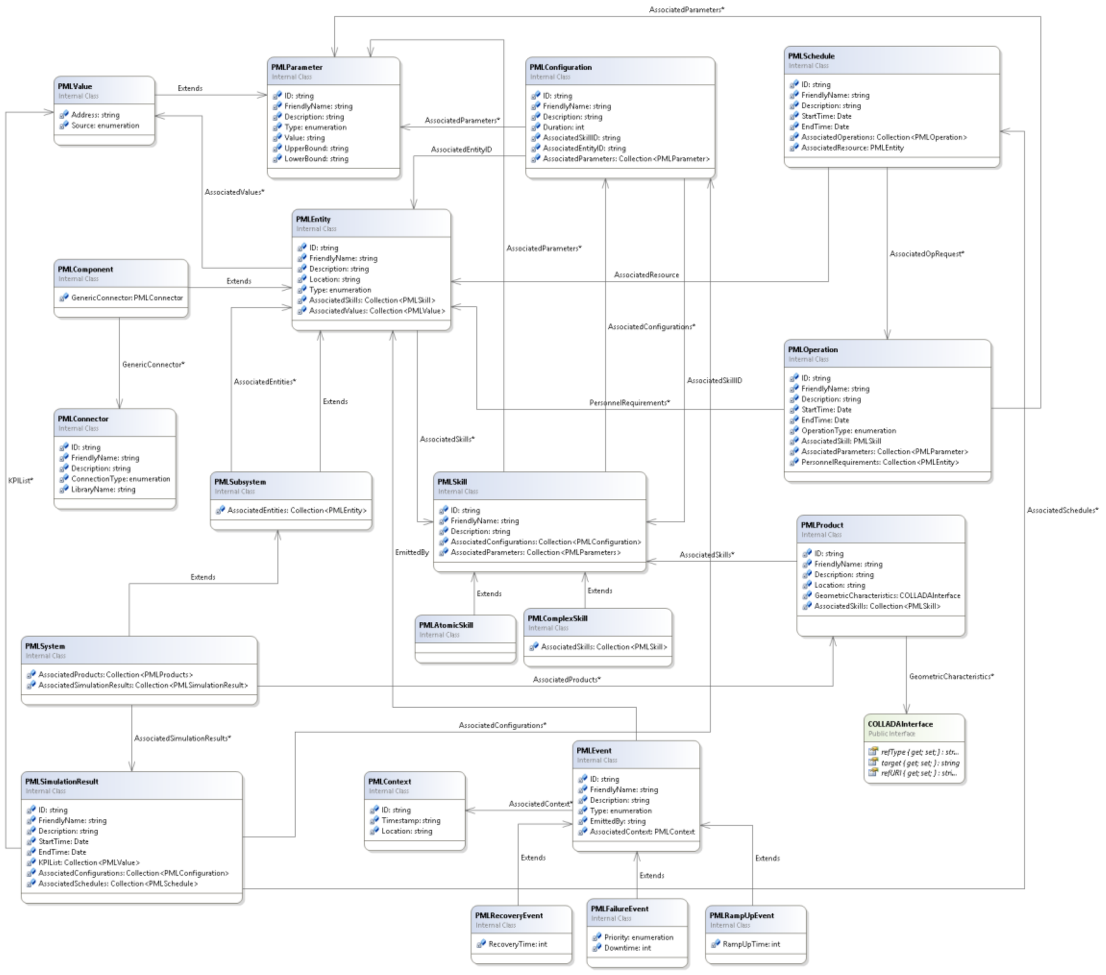


Figure 12: PERFoRM-ML class diagram

5.4. Standard Interface for Simulation

The interface of the SE provides the required functions in order to interact with the other participants via the PERFoRM middleware, i.e. for example start a simulation run. The functions can be separated into two categories:

- Functions provided by the SE, e.g. in order to start a simulation run
- Functions called/required by the SE, e.g. in order to return simulation results

Table 10: Functions required and called by the Simulation Environment

	Function Name	Function Description	Input Type	Input Description	Return Type	Return Description
Called by external service	getSimulationEnvironment Status	asks the SimEnv what its status is	-	no input	String	ENUM{"SIMULATING - BUSY" ,"IDLE - READY" ,"ERROR - UNAVAILABLE", "…"}
	abortSimulation	stop executing any simulation and return to IDLE state	-	no input	String	ENUM{"ABORT SUCCESSFUL" ,"ERROR - UNAVAILABLE","ERROR - NOT SIMULATING" , …"}
	getSimulation	Tells the Sim Env to obtain all necessary information to run a simulation, and then to execute	PMLSimulationResult. ExperimentType	ENUM {"SINGLE", "MONTE CARLO", "…"}	String	ENUM{"REQUEST ACCEPTED", "ERROR - UNAVAILABLE", "ERROR - BUSY", "…"}
			StartDate	date and time at which to start simulation.	-	-
			EndDate	date and time at which to end simulation.	-	-
			Collection <PMLConfiguration>	ID of the hierarchical elements (or a list of elements) that shall be considered in the	-	-
			PMLSimulationResult.KPIList Collection<PMLValue>	list of the particular results requested ENUM{"COMBINED OEE", "TOTAL THROUGHPUT", "ENERGY	-	-
			PMLEntity.ID or list<PMLEntity.ID>	The ID of the services requesting the results of the simulation	-	-
called by Simulation Environment	ReturnSimulationResults	After the simulation has finished, return the results to the service that called the simulation service	collection <PMLEntity.ID>	The ID of the services where the results of the simulation should be sent	-	-
			PMLSimulation Results	The requested results		
	getAll EntityIDs	asks the MW to return the list of all entities in the system	-	no input	list<PMLEntity.ID>	list of IDs for all machines
	getFactory Topology	asks the individual entity for the information	PMLEntity.ID	The ID of the entity for which information is requested	PMLEntity.FriendlyName	common name of the entity
					PMLEntity.Description	What kind of entity is this? ENUM{"PROCESSING", "ASSEMBLY", DISASSEMBLY", "RECEIVING", "SHIPPING", "…"}
					PMLEntity.Type	ENUM {PMLComponent, PMLSubsystem","…"}
					PMLEntity.AssociatedSkills	the list of PMLSkills that the Entity posseses
					PMLEntity.AssociatedValues	the list of PMLValues or PMLParameters of the Entity
					String	X,Y,Z Location of the entity in string format, ex. "100,12, 0"
	getFactory Topology	asks the MW to return the single file describing the entire set of machinery, workers, and other entities	-	no input	Collection <PMLEntity>	details as above for each such property of every PMLEntity
	getProductOrders	ask the MW to deliver the set of Product Orders	-	no input	Collection <PMLProduct>	the list of all ordered PMLProducts, along with their respective attributes
	getValue	get the value of a particular parameter of a particular entity for a particular time range. We will need properties like Machine status- which could be an ENUM of potential states	PMLEntity.ID	The ID of the entity for which information is requested	PMLEntity.ID	The actual entity that responded to the query
			PMLParameter.ID	A unique ID for a parameter describing the parameter desired	PMLParameter.Value	The value queried
			Time: string	either a single time, or a range	Time: string	The time at which the value was measured

Consequently, only the functions of the first category comprise the functions of the actual SE API whereas functions of the second category impose requirements on the API of other software modules or the middleware. The data parameters are either passed directly as functional arguments or are included in a data-model that is passed as an argument. Parameters based on a data model are extractions of the PERFoRM-ML data model as described in Deliverable 2.3 [20]. The following table gives an overview of the provided functions. Those are mainly the *getSimulation()* function that initiates and executes a simulation run depending on a set of parameters. Those parameters describe e.g. start and end date of a simulation run, the required task (i.e. single run, optimization, multi-evaluation-experiment …), the required topology scope of a given model etc. Additional functions allow the abortion of a simulation run and the request about the current status of the simulation (i.e. for example the completion rate).

As the functions of the SE are triggered asynchronously, the SE actively calls functions of other software modules in order to inform them when a simulation run was finished. All functions that can be called by the SE are listed in the following table. The most relevant function is the *returnSimulationResults()* that returns the results of the simulation run in dependence of the provided simulation parameters and the current factory status. In order to identify the addressee of the results, an ID is used that was send with the original simulation request. Additional functions are those that allow the SE to obtain additional data from other modules like the current topology of the factory (*getFactoryTopology()*) or the current production orders (*getProductOrders()*).

5.5. AnyLogic

5.5.1. Tool Capabilities

AnyLogic simulation software by The AnyLogic Company was created in 2000. It is the only simulation tool which supports agent-based, discrete event and system dynamic modelling methods and provides the opportunity to combine these modelling methods (Multimethod Simulation Software and Solutions). So it is possible to use this software to simulate different systems, depends on complexity and level of abstraction. AnyLogic has 2D and 3D graphical interfaces. To create a model, built-in libraries are used, which contains 15 different blocks and user can easily create a model out of them, just placing elements on a flow chart. The ability to customize models with Java language provides the flexibility for users to develop highly detailed and complex simulation models. Moreover, the last version of AnyLogic has a built-in database, which can be used to read the input data for the model and write the output results. It is also possible, to import data from other databases. [2]

AnyLogic can be used for a wide range of applications [3]:

- Supply Chains
- Healthcare
- Marketing
- Pedestrian Flows
- Transportation
- Project and Asset Management
- Business Processes

- Railroads Military
- IT
- Strategic Planning
- Social processes
- Manufacturing and Production

Specifically in Manufacturing and Production, factory simulation can be used to represent real life scenarios, to find bottlenecks, to identify system performance, utilization levels, cycle and lead times and to test layout implementations. AnyLogic is not suited for physical simulations and detailed kinetic simulation of 3D-Objects (e.g. crash-detection between trajectories of robots).

One highlight of AnyLogic is the graphic programming surface. Because of the graphic programming surface which is based on the programming language Java, it is easy to use AnyLogic. The following table presents all graphic programming surface items and a short description about typical application of these items [4]:

Table 11: AnyLogic graphic programming items

Item Name	Typical application
Stock & Flow Diagrams	Used for System Dynamics modelling
State charts	Used mostly in Agent Based modelling to define agent behaviour. They are also often used in Discrete Event modelling, e.g. to simulate machine failure
Action charts	Used in Discrete Event modelling, e.g. for call routing, or in Agent Based modelling, e.g. for agent decision logic
Process flowcharts	Used to define process in Discrete Event modelling

The language also includes: low level modelling constructions, (like variables, equations, parameters, events), presentation shapes (lines, ovals etc.), analysis facilities (datasets, histograms, plots), connectivity tools to e.g. databases, standard images (2D and 3D), and experiments frameworks. AnyLogic allows also users to import CAD drawings as DXF files, and then visualize models on top of them. This feature can be used for animating processes inside objects like factories, warehouses [4].

Additional to the graphic programming items, AnyLogic includes libraries. The libraries are helpful tools to simulate typical simulation problems. The following table describe libraries in AnyLogic [4]:

Table 12: AnyLogic Libraries

Name of the Library	Description of the Library
Process Modelling Library	Is designed to support simulation in Manufacturing, Supply Chain, Logistics and Healthcare areas
Pedestrian Library	Is dedicated to simulating pedestrian flows in a physical environment, like subway stations, security checks etc.
Rail Library	Supports modelling, simulating, and visualizing operations of a rail yard of any complexity and scale
Fluid Library	Allows the user to model storage and transfer of fluids, bulk goods, or large amounts of discrete items, which are not desirable to model as separate objects
Road Traffic Library	Allows users to simulate vehicle traffic on roads in real maps (AnyLogic supports OpenStreetMap)

Another highlight is, that AnyLogic model can be exported as a Java application. The exported Java application can be run separately, or integrated with other software as an additional module. Since 2015, AnyLogic Personal Learning Edition (PLE) is available for free. The PLE license is perpetual, but created models are limited in size (max. 10 agents). For public research in educational institutions, users can obtain a discounted University Researcher license, which does not limit model size and most of the functionality of a Professional license. [4]

5.5.2. Tool Interfaces and Tool Wrapper

The following interfaces in AnyLogic can be used to import and export data:

Table 13: AnyLogic Interfaces

	AnyLogic
File based	– .csv – .txt – .xls, .xlsx, .xlsm – URL
Inter-Process Communication (IPC)	- Java API (Everything that Java can, AnyLogic model can as well, but these interfaces must be extra programmed for AnyLogic)
Data Base Interfaces	– Micrososft SQL Server – Excel/Access – Other databases:

	– Access over ODBC/JDBC

5.5.3.Data Model

To automatically generate simulation models, certain data needs to be available:

Table 14: AnyLogic Data Model

Attribute Name	Var Type	Unit	Comment
Machine_data			Generic Simulation table to generate all machines
Machine_id	int		
Machine_name	string		
Tool_data			Specification of all KPI´s for each machine
Machine_id	int		
Tool_id	int		
Kpi	double		
Status_kpi	double		
Limit_kpi	double		
Unexpected_mt_time	double	minutes	
Expected_mt_time	double	minutes	
Plan_data			Generic Simulation table to generate all production plans
Plan_id	int		
Plan_values			Specification of all production plans
Plan_num	int		
Start_time	date		
Order_id	int		
Order_data			Specification of all orders for the production plans
Order_id	int		
Prod_type	int		
Lot_size	int		
Delivery_date	date		
Task_type	string		
Task_start_times			Specification of all task for each order
Plan_id	int		
Prod_type	int		
Start_time	date		
Lot_size_task	int		
Prod_data			Generic Simulation table to generate all products
Prod_id	int		
Prod_type	int		
Prod_name	string		
Product_processes			Defines the topology of the orders at the machines
Prod_type	int		
Tool_id	int		
Step_dur	double	minutes	

5.5.4.Transformation from PERFoRM-ML

Although AnyLogic is able to read and import data from external data sources, it is not able to automatically process various types of data sources (in different syntax and semantics) in a way that it can automatically perform simulation runs. This is only possible if standardized data formats are provided as input that allow the standardized processing of external data within AnyLogic. Otherwise, for each factory, a new AnyLogic translator would have to be programmed. Consequently, AnyLogic needs to be connected to the PERFoRM middleware via the Simulation Environment.

In order to transform data models for tool specific use, several options exist. One of the most used approach is the utilization of so called style sheets as defined themselves by xml-files.

In order to transform input data, the AnyLogic solution uses Apache Camel to transform the incoming and outgoing data from and to PERFoRM ML via XSLT Transformation. To do so, it uses the XSLT component from Apache Camel, which produces outputs into specific folders (input and output). The input folder is being read by AnyLogic on start up. The simulation uses the input data to perform the simulation and generates output results, which are served in the output folder. The web-services waits until a change in the output folder occurred and returns the results after another XSLT transformation.

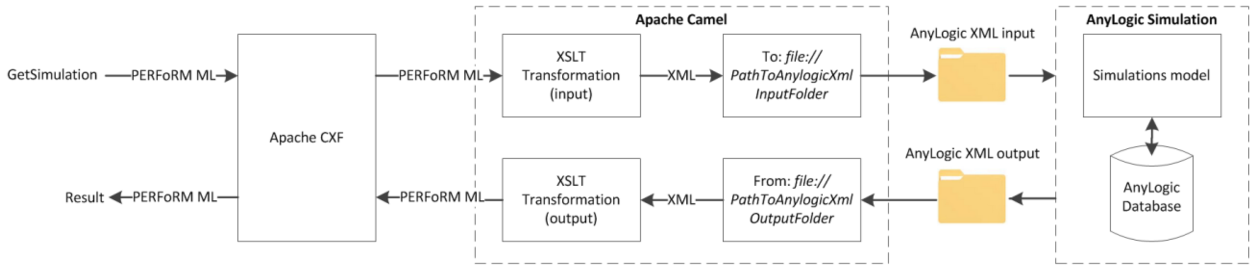


Figure 13: Dataflow view of a simulation run

5.5.5.Agent-Structure

The generic AnyLogic simulation model consists of four population of agents (see Figure 14). All these populations of agents are organized by the agent *"Main"*. The *"Main"* agent is responsible to route the created agents in the SE.

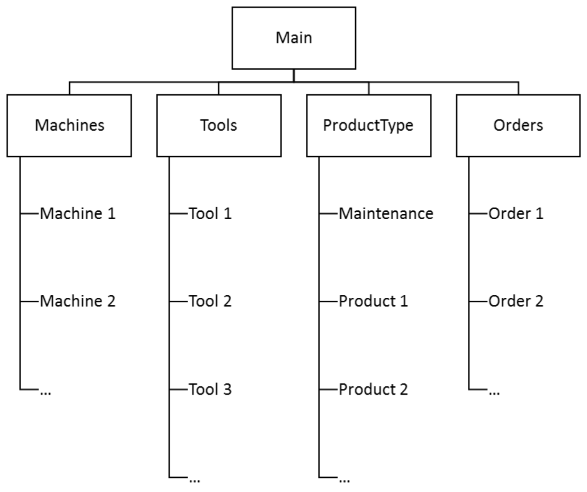


Figure 14: Agent structure of the generic simulation in AnyLogic

The population of agents contain information about the behaviour of the agent in the SE. The behaviour of the agents is described by parameters (see Table 14), functions, and discreet event modelling elements from AnyLogic. The number of agents and their associated parameters depend on the input data. For example, if the production plan contains five machines, the generic simulation model will create five single *"Machine"* agents with all parameters needed e.g. machine name, machine id. Each of these *"Machine"* agents use one or more tools to process the product agents. These tools are specified in the population of agent *"Tools"*. The agent "Tools" is responsible to monitor the KPI´s of the machines. In addition to the required machines, the production plan contains information about production tasks and maintenance tasks. These tasks are defined through the population of agents *"ProductionTypes"*. Maintenance task agents introduce maintenance at the affected machine agent and production task agents introduce the next production step at the affected machine for a product. All these tasks are organized in the population of agent *"Orders"*. The agent *"Orders"* contains information about e.g. lot size, delivery date, start time of the production/maintenance task, which are required to analyse the performance of the given production plans.

5.6. Plant Simulation

5.6.1.Tool Capabilities

Plant Simulation is a simulation tool that can perform material flow simulations of discrete event systems, mainly used for factories/production systems. However, due to its generic modelling approach, Plant Simulation is also applicable to other domains as e.g. airport logistics, hospital logistics, supply chains, or mining industry. On the other hand, Plant Simulation is not suited for physical simulations on a process level, as e.g. FEM or CFD simulations. Also 3D-kinematic simulations, e.g. in order to evaluate the trajectories of robot arms, are out of scope.

As described in chapter 2, typical application scenarios in a factory context are for example the optimization of production parameters like buffer sizes, lot sizes, schedules, production capacities, or shift-systems. Plant Simulation provides different modes for executing simulation runs, e.g. (i) single, manually conducted simulation runs (ii) automatically conducted sets of simulation runs, e.g. in a Monte-Carlo approach, by using the experiment manager or (iii) optimizing certain parameters in a metaheuristic optimization approach using the integrated genetic algorithm.

Plant Simulation has several features and characteristics that make it well suited to be employed in the PERFoRM approach for simulating flexible and reconfigurable production systems, of which some of the most relevant ones are listed in the following:

- Object oriented thinking: Plant Simulation utilizes an object oriented approach for the generation of simulation models, embracing object oriented paradigms like inheritance or data encapsulation. Simulation models are then setup by instantiating model components (via drag and drop or dynamically via generation scripts) like machines or transport systems from class libraries.
- Many predefined library components for discrete factories and process industry domains as for example production process, assembly process, storage system, conveyor, employee, fork-lift etc. (compare Figure 15) that allow a quick setup and parameterization of simulation models for regular simulation tasks like bottleneck analysis.

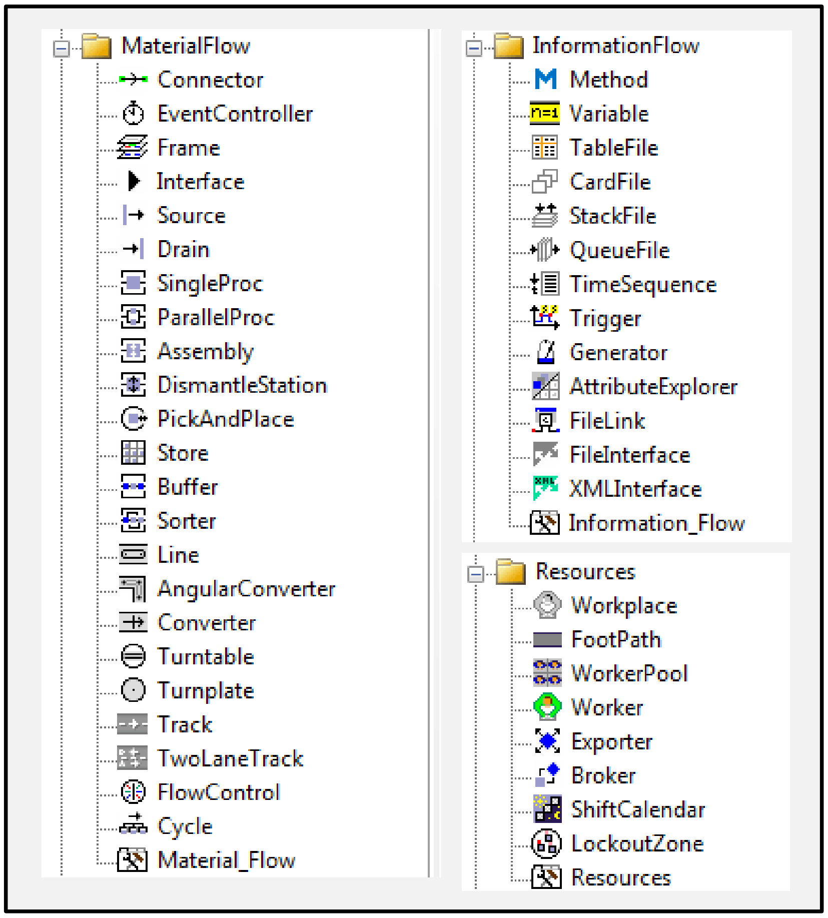


Figure 15: Extraction of components of the Plant Simulation Library

- Configurable and customizable: Although many library components are predefined for specific purposes, the functionality of all of them can be customized and extended by using the Plant Simulation internal script language “SimTalk”. SimTalk allows for example the programming of customized routing and control logics that are not provided as standard routing options. In addition, using SimTalk, simulation models can be completely automatically generated and executed.
- The connection of Plant Simulation to other soft- or hardware systems, data files, or the integration in other software environments (like the PERFoRM SE) can be enabled via various interfaces and import/export mechanisms that are supported by Plant Simulation. Those are for example, SQL, .dll, OPC, server-client-communication (compare Table 15). This also allows the utilization of Plant Simulation in a hardware- or software-in-the-loop approach for testing the plc-code in a virtual commissioning approach.
- Although physical or kinematic 3D modelling is not the focus of Plant Simulation, 3D simulations of factories, mainly for the purpose of animation, can be conducted.
- Value streams derived from a static and retrospective value stream analyses in a factory can be made dynamic by being included in the simulation.

- Many options for user friendly evaluation and visualization of simulations are available, like customizable HTML-reports, charts, KPIs etc.

5.6.2. Tool Interfaces and Tool Wrapper

As already mention in the previous section, Plant Simulation provides a range of interfaces for file import and export, for inter-process communication (IPC), and data base interfaces that are listed in the following Table 15.

Table 15: Plant Simulation Interfaces

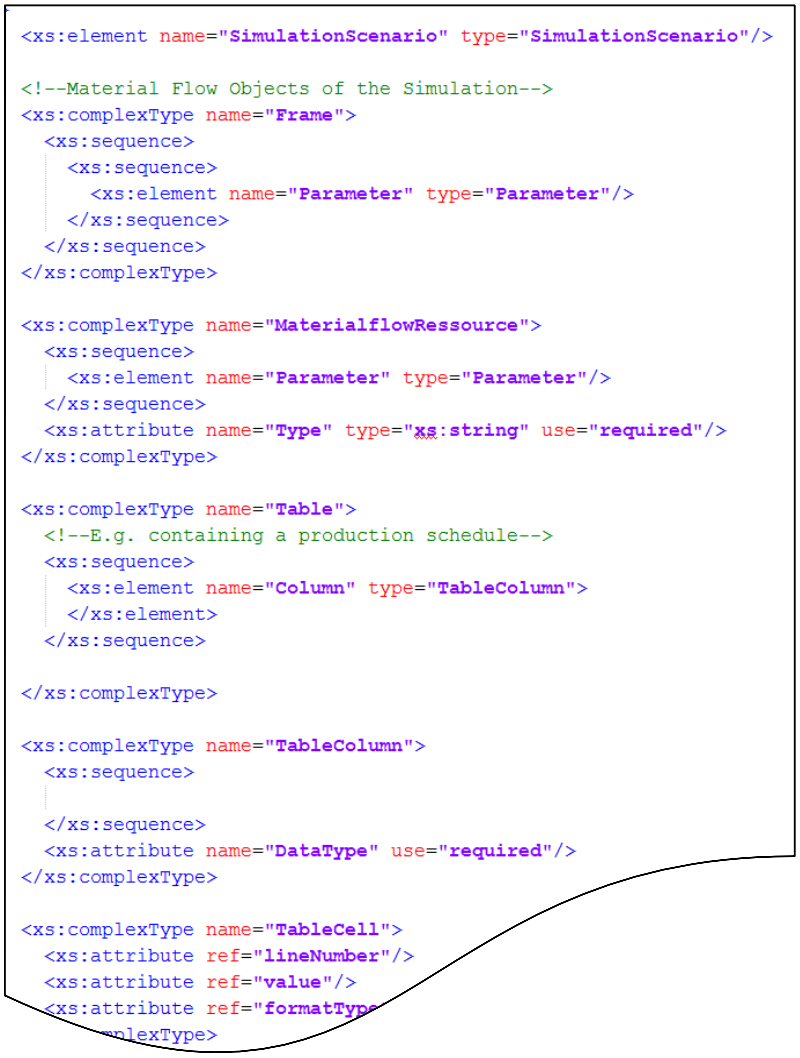
	Plant Simulation
File based	– .xls – .txt – .xml
Inter-Process Communication (IPC)	– C-Interface (.dll‘s) – COM-Interface – DDE (Server and Client) – OPC-DA (however no OPC-UA!) – Socket-Connection (TCP/UDP, Server and Client) – (Teamcenter, HTML, Simit)
Data Base Interfaces	– SQLite – Oracle11g – ODBC

5.6.3. Data Model

The plant simulation data model consists of two parts:

1. A formal data-schema/data-model that defines the syntax of how all basic elements of simulation-models and simulation-scenarios are described based on an xml-schema file (.xsd) (compare Figure 16).
2. A library that lists all the available plant simulation components (e.g. material flow source, process unit, product, production-table etc.) and their respective properties, value ranges, and (if required) units (compare Table 16).

In this way, the fundamental data structure, i.e. how elements and results are stored, is separated from the actual semantic content. This has the advantage that not for every application scenario the underlying .xsd file has to be rewritten. In addition, the semantic contents, i.e. what model elements are available often change during each version of the simulation tool.



```
<xs:element name="SimulationScenario" type="SimulationScenario"/>

<!--Material Flow Objects of the Simulation-->
<xs:complexType name="Frame">
  <xs:sequence>
    <xs:sequence>
      <xs:element name="Parameter" type="Parameter"/>
    </xs:sequence>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="MaterialflowRessource">
  <xs:sequence>
    <xs:element name="Parameter" type="Parameter"/>
  </xs:sequence>
  <xs:attribute name="Type" type="xs:string" use="required"/>
</xs:complexType>

<xs:complexType name="Table">
  <!--E.g. containing a production schedule-->
  <xs:sequence>
    <xs:element name="Column" type="TableColumn">
    </xs:element>
  </xs:sequence>

</xs:complexType>

<xs:complexType name="TableColumn">
  <xs:sequence>

  </xs:sequence>
  <xs:attribute name="DataType" use="required"/>
</xs:complexType>

<xs:complexType name="TableCell">
  <xs:attribute ref="lineNumber"/>
  <xs:attribute ref="value"/>
  <xs:attribute ref="formatTyp
  mplexType>
```

Figure 16: Extraction of schema file for Plant Simulation Data model

The schema file is build based on the idea that the effort for automatically generating simulation models from it (within Plant Simulation) is minimized (compare also section below "Automatic Model Generation"). This means that ideally most of the data items in the data model can be interpreted in a standard import loop that does not need to be adapted for each component.

Table 16: Extraction of the library of Plant Simulation components utilized in the PERFoRM simulation approach

Plant Simulation Object (z.B. XX.SingleProc)	ObjectType		
Attribute Name	*VarType*	*Unit*	*Comment*
SingleProc	**MaterialFlow.SingleProc**		Generic Simulation Component that usual
ProcTime	Time	Seconds	Average Process Time - can be left empty
SetupTime	Time	Seconds	
EntranceControl	EntranceStrategy	Method/String	
ExitControl	ExitStrategy	Method/String	
Xpos	Distance	int	needed so that machine location can be c
Ypos	Distance	int	needed so that machine location can be c
Store	**MaterialFlow.Store**		
Xdimension	Integer		
Ydimension	Integer		
Xpos	Distance	int	needed so that machine location can be c
Ypos	Distance	int	needed so that machine location can be c
EntranceControl	EntranceStrategy	Method/String	
ExitControl	ExitStrategy	Method/String	
Buffer	**MaterialFlow.Buffer**		
Capacity	Integer		storage spaces between operations
Xpos	Distance	int	needed so that machine location can be c
Ypos	Distance	int	needed so that machine location can be c
EntranceControl	EntranceStrategy	Method/String	
ExitControl	ExitStrategy	Method/String	
Conveyor	**MaterialFlow.Conveyor**		
Length	Real	[m]	
Speed	Real	[m/s]	
Source	**MaterialFlow.Source**		
ModeOfCreation	String [enum]	{Interval, Number, Delivery Table, Trigger}	
Start	Time	Seconds	
Stop	Time	Seconds	
Interval	Time	Seconds	
CreationTable	Table [CreationTable]		
Sink	**MaterialFlow.Drain**		
EntranceControl	EntranceStrategy	Method/String	
ProcTime	Time	Seconds	
Set-up Time	Time	Seconds	

5.6.4.Transformation from PERFoRM-ML

For the creation of Plant Simulation specific data models from the PML data model, xslt style sheets are used in order to transform the xml files. This approach is analogous to the one described for AnyLogic tool (see section 5.5.4) and is thus not further explained here.

5.6.5.Automatic Model Generation

In order to automatically generate simulation models, the Plant simulation data models are imported via the xml interface of plant simulation and loaded into “string-tables”. This means that a variable type conversion has to be manually conducted afterwards based on the provided variable type information. In order to generate the model, the imported table is then iteratively looped and the simulation objects are created based on the plant simulation internal programming language SimTalk. Therefore especially the command *createObject(SimulationFrame, InstanceName, XPosition, YPosition)* is utilized that generates simulation objects of a specific ClassType in a SimulationFrame with an InstanceName at a certain position (x,y). The classType can be almost any of the classes that PlantSimulation provides in its classLibrary but also customized/use-defined classes that are e.g. composed of several other classes.

The simplified pseudo code for automatically creating simulation models from a string-based input table that contains all simulation-elements looks as follows:

```
For each Element in XML-Input-Table Do
    Create Simulation-Object of type Element.Type with name Element.Name
    For each Parameter in Sub-Table of Element
        Identify variable type (e.g. Boolean, integer, string)
        Set Element.parameter-name.value := stringToVarType(parameter.value)
    End {For-Parameters}
End {For-Elements}
```

This approach of course requires that the naming conventions as defined in the component library (Table 16) are identical to the names that Plant Simulation uses. Otherwise mapping tables can be used where alternative data model names are prescribed that differ from the proprietary Plant Simulation names.

5.7. Concept for Integration of ABS within a Commercial DES Tool

As Borschev and Filippov have illustrated, DES and ABS are not contradictory methods but can easily correspond [5]. This fact can be used to combine the elements of DES with the functions of a DES tool to implement an ABS in an industrial grade software tool based on DES. As an ABS is generally run in discrete time steps the timing tools of a DES like system clock and event list can also be applied for it. Further, both methods are built up on entities which are passive in case of DES and active as well as individual in the case of ABS. An evaluation of simulation software for modelling manufacturing systems revealed that industrial grade software tools are providing all similar functionality like intuitive handling, object-oriented structure and custom programming. These functions offer several starting points to transform the elements of a DES into an ABS logic, as shown in Figure 17.

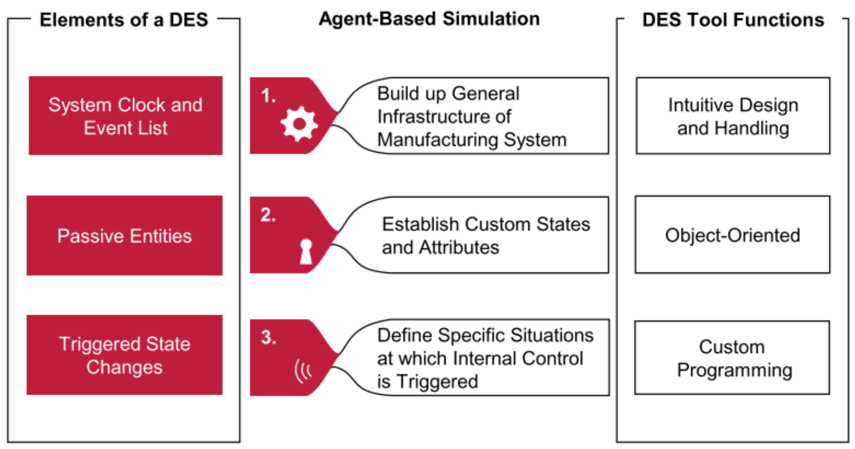

Elements of a DES
System Clock and Event List
Passive Entities
Triggered State Changes
Agent-Based Simulation
1.
Build up General Infrastructure of Manufacturing System
2.
Establish Custom States and Attributes
3.
Define Specific Situations at which Internal Control is Triggered
DES Tool Functions
Intuitive Design and Handling
Object-Oriented
Custom Programming

Figure 17: Three general steps for implementing an agent-based logic into a DES tool

Figure 17 shows the three main steps to transform the elements of a DES into an ABS logic. Further, on the left hand side it is exemplified which general elements of a DES are used and the right hand side displays the functions of the simulation software that are used. The breakdown into these three steps results from the simple model of an agent and its interaction mentioned in the subsection about ABS (see Figure 5). In the first step the environment is built, in the second step the agent is implemented, and in the third step the interaction between the agent and environment are introduced.

1st Step - Build Up General Infrastructure of Manufacturing System In the first step the general infrastructure of the agent-based MS for the agent is built up, representing the environment of the agent. This step is generally equal to building up a traditional simulation of a MS as described in the VDI guideline 3633 [19]. The only difference is that the infrastructure might be more complex to offer the needed flexibility for the MS and its elements. Hence, elements like work stations (WS), buffers, sources and drains are introduced to ensure a steady material flow. A WS and a buffer together can be regarded as a working unit. The elements can be either simply connected without any further logistical consideration or a logistic concept can be introduced. This could for example be implemented by means of conveyor lines or tracks with a transporting unit that connects the working units. As an agent-based MS is in general build up to propose a certain flexibility all working units should be connected to all other working units so a product can freely move between them.

In Figure 18 an example for a general infrastructure in Plant Simulation with nine working units is displayed. All working units are connected by conveyor lines with all other working units. Further, a starting buffer behind the source is connected with the working units. All elements can be easily added by drag and drop and should suit the designated application task. Another option is to implement an automatic procedure for allocating and building up the elements by custom programming.

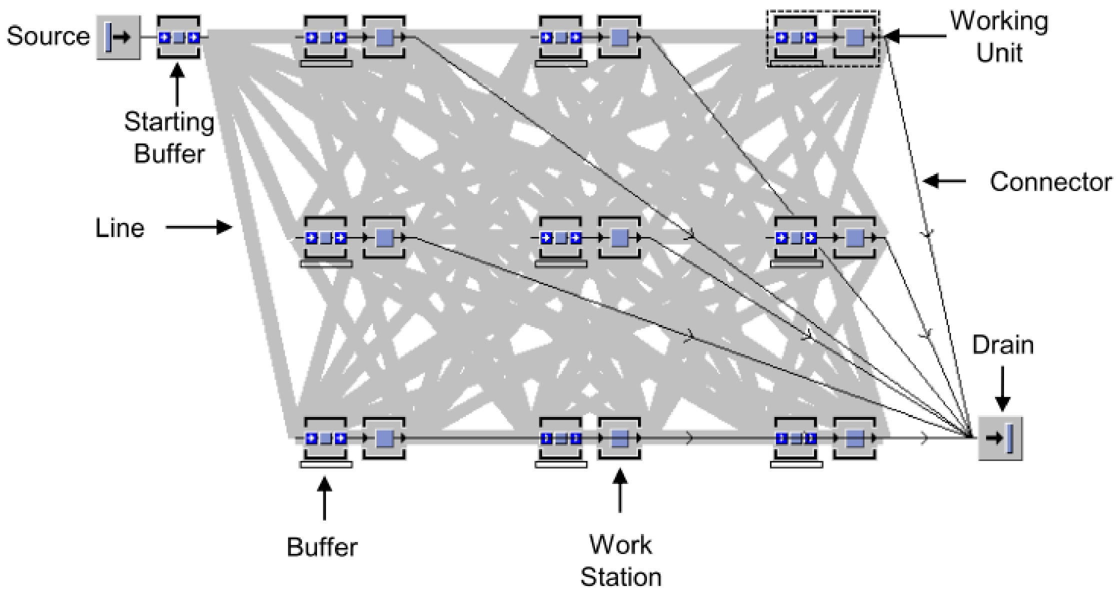
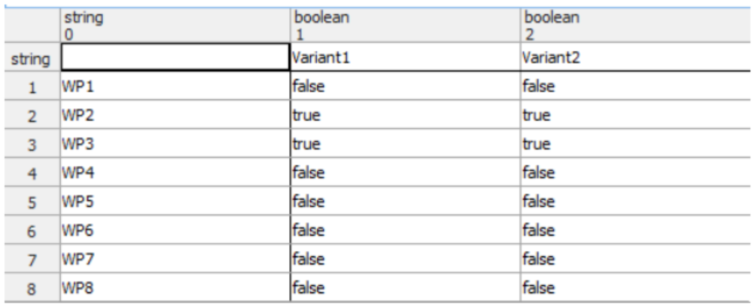


Figure 18: General infrastructure in Plant Simulation with main elements

2nd Step - Establish Custom Status and Attributes In this step it is determined which elements will be regarded as agents and what their individual and unique knowledge will be. This step is important, to provide the agents with some kind of autonomy. As exemplified in the subsection about ABS this is one main property of agents. To achieve autonomy, passive entities of a DES are equipped with individual attributes and a unique state by using the object-oriented structure of the DES tools. Thus, the entities gain a certain autonomy which is a first important condition to regard them as agents. The agents to be introduced could include but are not limited to the following elements:

- Product Agent
- Buffer Agent
- Work Station Agent
- Line Agent
- Drain Agent

All of these elements could possibly take decisions or communicate with the other elements. Therefore, they are regarded as potential agents. Which elements exactly will be taken as agents and how large the level of autonomy is, is dependent from the purpose of the simulation model. A possible attribute for the product agent is the process time for each work package. This attribute is inherent to every product variant and includes the time each WP needs for being processed. Consequently, each product “knows” how long it will need for processing and can hand this information to other agents. Another obligatory attribute is the actual status of a product regarding its actual WP. Thus, the product is aware of which process step is to be machined next. The WS should be equipped with the Boolean attributes giving information which work package of which product variant can be processed. Attributes can be implemented in tables as the example in **Fehler! Verweisquelle konnte nicht gefunden werden.** illustrates. By extending the agents with more attributes different functions can be implemented and the autonomy of the agents is improved.



	string 0	boolean 1	boolean 2
string		Variant1	Variant2
1	WP1	false	false
2	WP2	true	true
3	WP3	true	true
4	WP4	false	false
5	WP5	false	false
6	WP6	false	false
7	WP7	false	false
8	WP8	false	false

Figure 19: Attribute of a workstation for process able work packages in a table

3rd Step - Implement Control Logic The third step can be regarded as the main step to provide the model with an agent-based logic. Here the interaction of the agent with its environment is defined by implementing pro-activity, reactivity and the social ability. To do this the DES capability that the entities trigger state changes when an event happens is used to implement an internal control of the agents. Events like a product entering a WS (entrance control) can be easily combined with the call of a custom programmed method. In this method the rules for communication between the agents and the goals by which the agents act can be set. Thus, the agent is represented perceiving its environment by sensors, reacting and making decisions.

To conclude, Figure 20 shows which property of an agent (see ABS subsection) is achieved in which step by which means. By building up the infrastructure of the MS in step 1 it is ensured that the agent later can be regarded as situated and connected to some kind of environment. In the 2nd step the agents are determined and provided with an internal state to gain autonomy. In step 3 the agent is provided with pro-activeness, reactivity and social ability. Pro-activeness is reached through the goal directed manner in which the agents act through control strategies which will be implemented in the custom programmed lines of code. Reactivity is reached through forcing an action of the agent when a specific event like entering a WS happens. Additionally in the custom programmed methods the way of communication and thus the social ability of the agents is realized. After the 3rd step all capabilities of an agent which were mentioned in the previous description of ABS are assigned in the simulation model and its elements. Thus, the model can now be regarded as an agent-based MS.

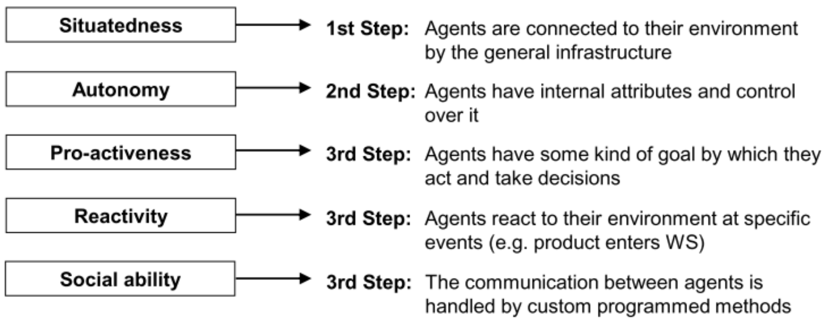

Situatedness
1st Step: Agents are connected to their environment by the general infrastructure
Autonomy
2nd Step: Agents have internal attributes and control over it
Pro-activeness
3rd Step: Agents have some kind of goal by which they act and take decisions
Reactivity
3rd Step: Agents react to their environment at specific events (e.g. product enters WS)
Social ability
3rd Step: The communication between agents is handled by custom programmed methods

Figure 20: Properties of agents: Steps of implementation

5.8. Application and Integration

5.8.1. Application and Integration to a Use Case Example

In Chapter 3 of this document, the use cases were analyzed in order to develop a set of functional requirements on the capabilities of the SE. Then, in Chapter 4, these functional requirements were transformed into technical requirements that were then used to drive the development of a concept for the architecture of the SE. In this chapter, this concept has been elaborated and detailed, resulting in a concept that could be implemented in each use case, being customized to the needs in each location. In this section, we verify that this concept is indeed capable of meeting the functional requirements identified for an illustrative use case.

Figure 21 provides a deeper examination of the internal activity of the SE for this use case, and the remainder of this section investigates key aspects of these processes.

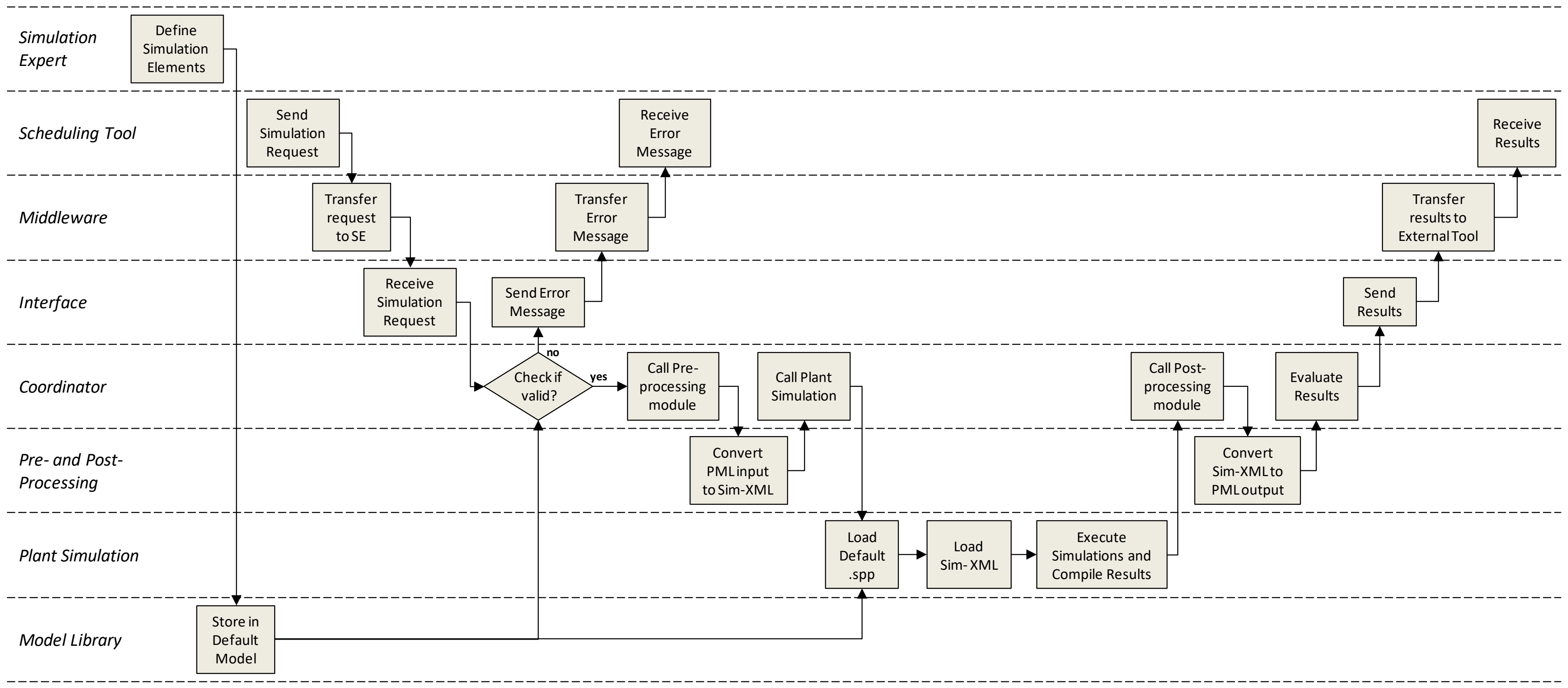


Figure 21: Configuration and Execution of Simulation Environment for Siemens Use Case

An overall goal of the Siemens use case is the exploration of flexibility that can be gained when preventative maintenance becomes possible to plan. This strategic goal is to be accomplished via a chain of processes and services shown in Figure 22 below. First, data from various sources is made available to a Data Analytics service, which then produces a description of the failure likelihoods for specific pieces of equipment in the factory. A user then has the capability to define tasks that could be performed to remediate and prevent these potential failure events, which are then integrated into a master production and maintenance planning schedule. As the scheduling tool that creates this master schedule is deterministic, it is unable to completely address the probabilistic nature of the failure of components, and therefore may provide inaccurate estimates of expected behaviour. Thus, it is in this case advantageous for a Monte Carlo simulation to be executed to more fully evaluate the potential schedules developed by this scheduling tool. This simulation will be conducted within the SE.

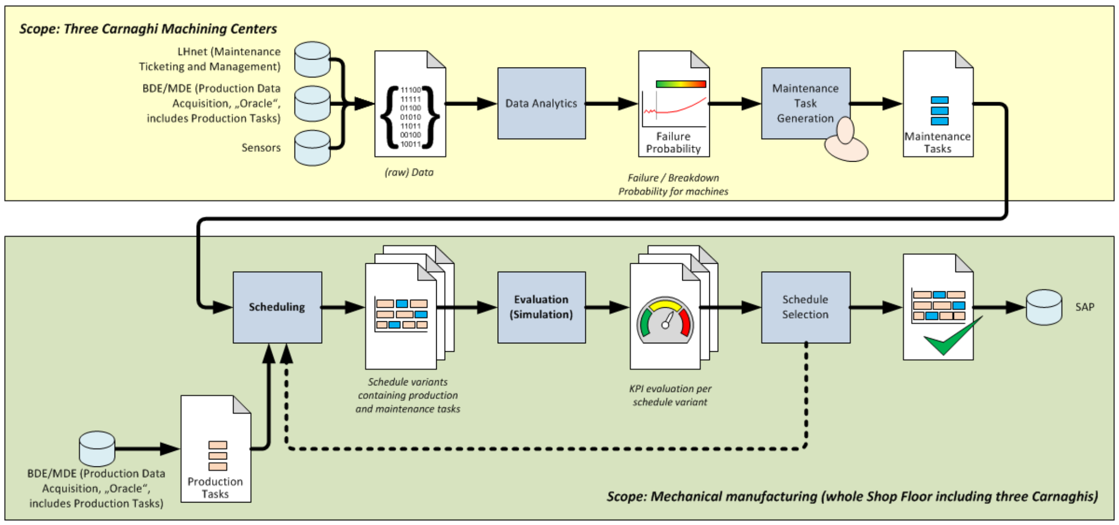

Scope: Three Carnaghi Machining Centers
LHnet (Maintenance Ticketing and Management)
BDE/MDE (Production Data Acquisition, „Oracle", includes Production Tasks)
Sensors
(raw) Data
Data Analytics
Failure Probability
Failure / Breakdown Probability for machines
Maintenance Task Generation
Maintenance Tasks
Scheduling
Schedule variants containing production and maintenance tasks
Evaluation (Simulation)
KPI evaluation per schedule variant
Schedule Selection
SAP
BDE/MDE (Production Data Acquisition, „Oracle", includes Production Tasks)
Production Tasks
Scope: Mechanical manufacturing (whole Shop Floor including three Carnaghis)

Figure 22: Process Flowchart of the Siemens Use Case

- Define Simulation Elements (Simulation Expert)

In this step, the Simulation Expert must define types of entities that will need to be later utilized during simulation. Specifically, the user should focus on any entities that cannot be sufficiently modelled using standard library elements. Therefore, the primary elements of concern will be briefly discussed here.

Large compressors can be abstracted into four key parts, as shown in Figure 23:Rotor, Stator, Housing, and Mounting. Within this project, the Siemens Use Case focuses on the manufacturing of the Stator components, in order to manage the complexity of the process. As already described in Deliverable 7.1 [21] the manufacturing processes executed are mainly machining or assembly-oriented, in which one of a kind products and small lot sizes are accomplished via a high share of manual labour. In particular, the assembly stages have a high degree of complexity.

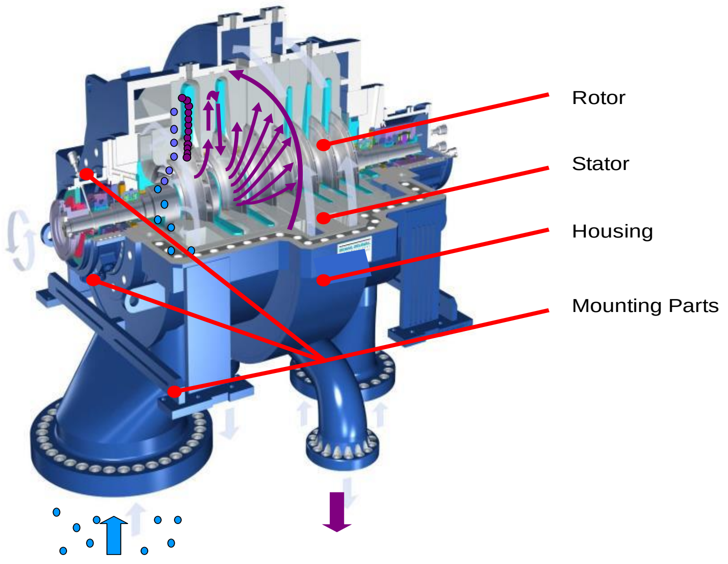


Figure 23: Parts Composing a Large Compressor

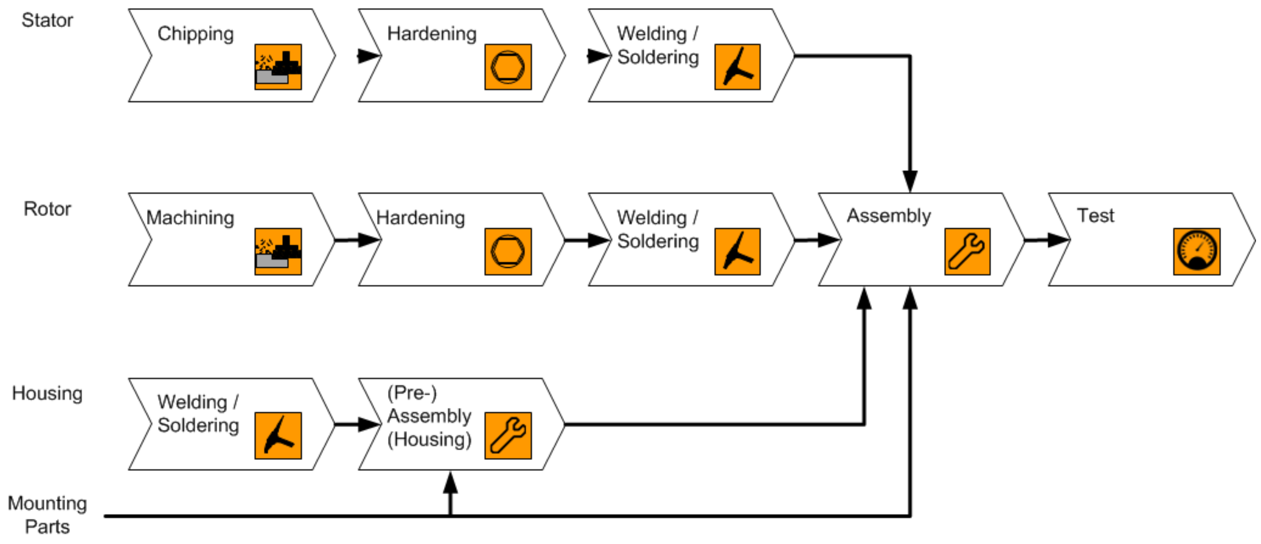


Figure 24: Simplified Overview of Manufacturing Process for Large Compressors

As shown in Figure 24, the Stator turbine blades are first machined from incoming materials, then experience a hardening process to strengthen the steel, are next joined together to form complete profiles, and then assembled with the Rotor, Housing, and Mountings to produce the final product, which can then be tested. Throughout the process, one may also find quality control checks and reworking processes to ensure conformity in the end product.

Of these processes, the machining stages, as performed by 3 CNC machines, have been selected for deeper analysis with respect to prognosticating failure. As such, the Data Analysis cluster will produce predictions of failure for various components within the machines, each of which having differing effects on the resultant productivity, energy consumption, or capacity. To accommodate this more easily, a library component will be defined to capture these aspects as default parameters (see Figure 25). The remaining process stages will be captured using standard *SingleProc* elements, which include only the standard whole-machine failure model.

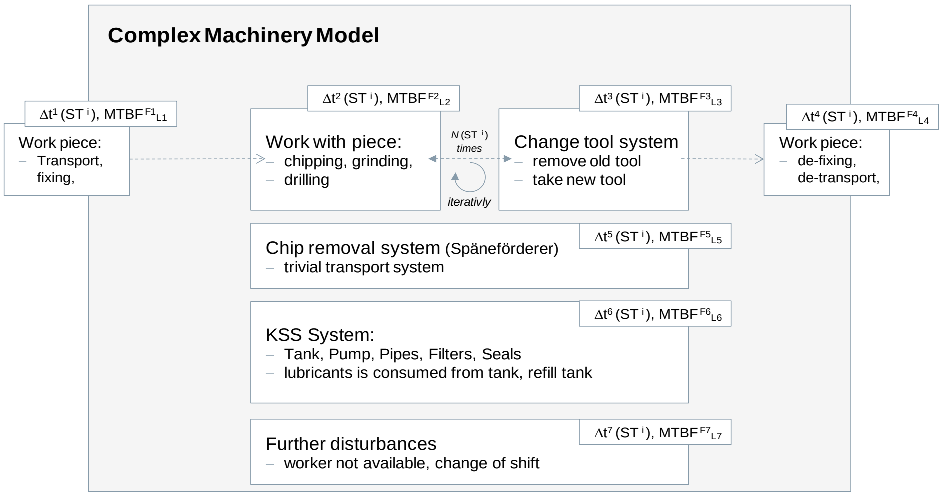
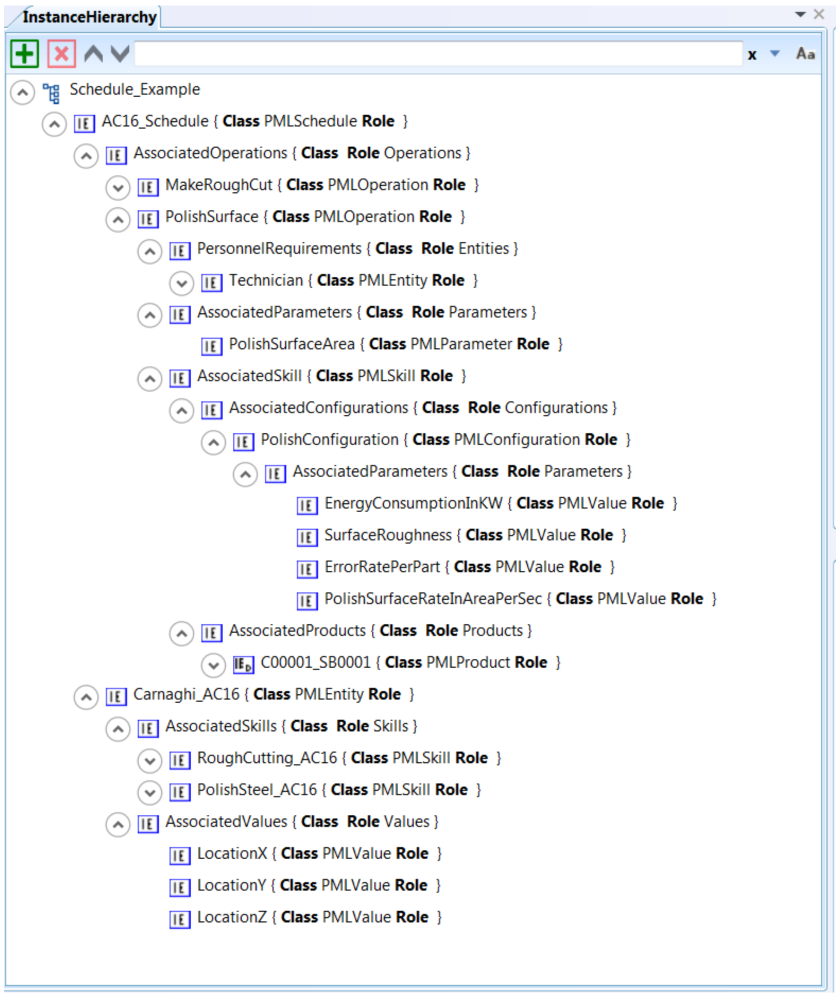

Complex Machinery Model
$\Delta t^1 (ST^i)$, $MTBF^{F1}_{L1}$
Work piece:
– Transport,
fixing,
$\Delta t^2 (ST^i)$, $MTBF^{F2}_{L2}$
Work with piece:
– chipping, grinding,
– drilling
$N (ST^i)$ times
iterativly
$\Delta t^3 (ST^i)$, $MTBF^{F3}_{L3}$
Change tool system
– remove old tool
– take new tool
$\Delta t^4 (ST^i)$, $MTBF^{F4}_{L4}$
Work piece:
– de-fixing,
de-transport,
Chip removal system (Späneförderer)
– trivial transport system
$\Delta t^5 (ST^i)$, $MTBF^{F5}_{L5}$
KSS System:
– Tank, Pump, Pipes, Filters, Seals
– lubricants is consumed from tank, refill tank
$\Delta t^6 (ST^i)$, $MTBF^{F6}_{L6}$
Further disturbances
– worker not available, change of shift
$\Delta t^7 (ST^i)$, $MTBF^{F7}_{L7}$

Figure 25: Modelling Component Failure in Machining Equipment

Next, the products to be produced and the personnel can be considered. In this example, the products do not include agent-like capabilities, and therefore require no additional model definition. Similarly, the personnel that oversee and execute the production tasks are not modelled as complex agents, and therefore also require no additional model definition.

- Send Simulation Request (Scheduling Tool)

In the Compressor use case, simulation is utilized to evaluate potential master production schedules for robustness under normally varying factory floor conditions. As described in Deliverable D2.3 [20], a PMLSchedule references the PMLOperations that should occur on particular PMLEntities. An example of a portion of a PMLSchedule is shown in Figure 26.



InstanceHierarchy

```
Schedule_Example
  AC16_Schedule { Class PMLSchedule Role }
    AssociatedOperations { Class  Role Operations }
      MakeRoughCut { Class PMLOperation Role }
      PolishSurface { Class PMLOperation Role }
        PersonnelRequirements { Class  Role Entities }
          Technician { Class PMLEntity Role }
        AssociatedParameters { Class  Role Parameters }
          PolishSurfaceArea { Class PMLParameter Role }
        AssociatedSkill { Class PMLSkill Role }
          AssociatedConfigurations { Class  Role Configurations }
            PolishConfiguration { Class PMLConfiguration Role }
              AssociatedParameters { Class  Role Parameters }
                EnergyConsumptionInKW { Class PMLValue Role }
                SurfaceRoughness { Class PMLValue Role }
                ErrorRatePerPart { Class PMLValue Role }
                PolishSurfaceRateInAreaPerSec { Class PMLValue Role }
          AssociatedProducts { Class  Role Products }
            C00001_SB0001 { Class PMLProduct Role }
    Carnaghi_AC16 { Class PMLEntity Role }
      AssociatedSkills { Class  Role Skills }
        RoughCutting_AC16 { Class PMLSkill Role }
        PolishSteel_AC16 { Class PMLSkill Role }
      AssociatedValues { Class  Role Values }
        LocationX { Class PMLValue Role }
        LocationY { Class PMLValue Role }
        LocationZ { Class PMLValue Role }
```

Figure 26: Portion of a PMLSchedule – Polishing a Stator Turbine Blade

In the example, a portion of the schedule for the machine "AC_16" is shown. Two operations have been requested: *MakeRoughCut* and *PolishSurface* for part ID *C00001_SB0001*. Necessary parameters for the schedule are included, for example *PolishSurfaceArea* and *PolishSurfaceRateInAreaPerSec*, or the *EnergyConsumptionInKW*. The necessary Personnel class *Technician* is also referenced.

- Call Simulation Tool (Coordinator)

When the Coordinator has converted the PMLSimulationResult into a Sim-XML conformant file, then PlantSimulation can be executed. This is done via the preconfigured wrapper which has the program path to the local copy of PlantSimulation. Then using the command line argument, the tool can be called.

- Load Default Model (Simulation Tool)

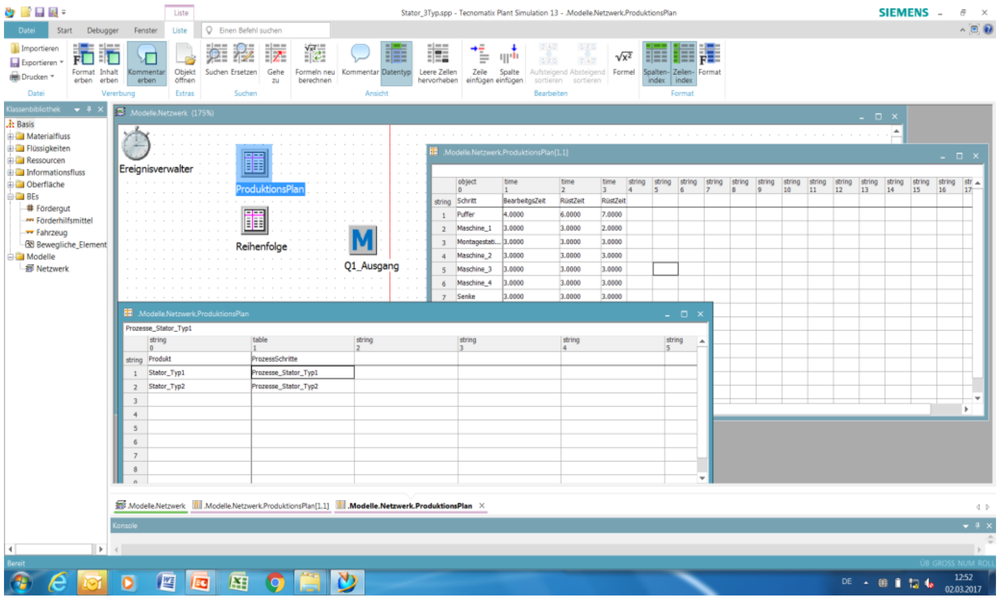
The default model elements created in the first step can be loaded as an input argument with the command line:

```
CMD
"/c C:\\Program Files\\Tecnomatix\\Plant Simulation 12.1\\PlantSimulation12_2.exe" -f "D:\\baseModel.spp"
```

which would then load the base model for the factory, including the structured model elements.

- Load Sim-XML (Simulation Tool)

Upon loading, the PlantSimulation baseModel.spp automatically begins executing any *init()* methods stored within any submodel before executing the model. In this example, the *init()* call, among other actions, leads the model to parse the Sim-XML file containing the updated schedule and system state definition. Machines will have their states updated to, for example, declare which product parts are currently being processed on which machines – or which machines are currently available / unavailable. The schedule is then stored as a Table element within the model so that it can be accessed during runtime (see Figure 27).



.Modelle.Netzwerk.ProduktionsPlan[1,1]

	object 0	time 1	time 2	time 3	string 4	string 5	string 6	string 7	string 8	string 9	string 10	string 11	string 12	string 13	string 14	string 15	string 16
string	Schritt	BearbeitgsZeit	RüstZeit	RüstZeit													
1	Puffer	4.0000	6.0000	7.0000													
2	Maschine_1	3.0000	3.0000	2.0000													
3	Montagestati...	3.0000	3.0000	3.0000													
4	Maschine_2	3.0000	3.0000	3.0000													
5	Maschine_3	3.0000	3.0000	3.0000													
6	Maschine_4	3.0000	3.0000	3.0000													
7	Senke	3.0000	3.0000	3.0000													

.Modelle.Netzwerk.ProduktionsPlan

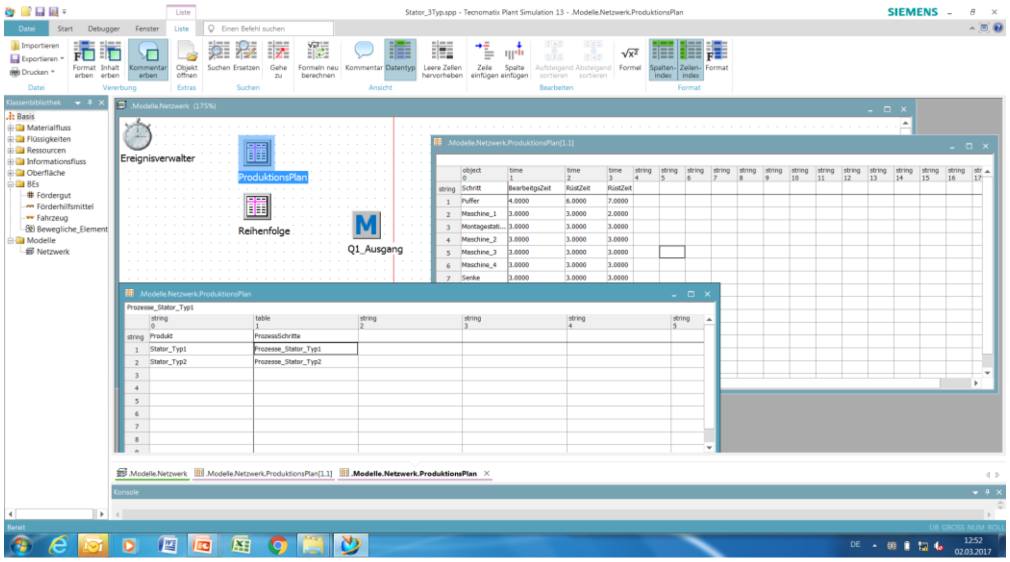
Prozesse_Stator_Typ1

	string 0	table 1	string 2	string 3	string 4	string 5
string	Produkt	ProzessSchritte				
1	Stator_Typ1	Prozesse_Stator_Typ1				
2	Stator_Typ2	Prozesse_Stator_Typ2				
3						
4						
5						
6						
7						
8						

Figure 27: Example Portion of a Planned Schedule Stored within a PlantSimulation Model

- Execute Simulations and Compile Results (Simulation Tool)

When the *init()* methods terminate, the DES-style simulation begins executing and runs for the requested simulation horizon having been specified. The results of a single simulation are then entered into the next column of the Table *SimulationResultKPIs*, as shown in Figure 28.



.Modelle.Netzwerk.ProduktionsPlan[1.1]

	object 0	time 1	time 2	time 3	string 4	string 5	string 6	string 7	string 8
string	Schritt	BearbeitgsZeit	RüstZeit	RüstZeit					
1	Puffer	4.0000	6.0000	7.0000					
2	Maschine_1	3.0000	3.0000	2.0000					
3	Montagestati...	3.0000	3.0000	3.0000					
4	Maschine_2	3.0000	3.0000	3.0000					
5	Maschine_3	3.0000	3.0000	3.0000					
6	Maschine_4	3.0000	3.0000	3.0000					
7	Senke	3.0000	3.0000	3.0000					

.Modelle.Netzwerk.ProduktionsPlan

Prozesse_Stator_Typ1

	string 0	table 1	string 2	string 3	string 4	string 5
string	Produkt	ProzessSchritte				
1	Stator_Typ1	Prozesse_Stator_Typ1				
2	Stator_Typ2	Prozesse_Stator_Typ2				
3						
4						
5						
6						
7						
8						

Figure 28: Table *SimulationResultKPIs* Compiling Results from Individual Simulations

Upon termination of the requested number of simulations, this table is then analyzed to determine the average, best, and worst values for the KPIs requested.

- Receive Results (Scheduling Tool and Visualisation)

After the simulations have finished, the middleware routes the results to the Scheduling Tool, which shall then receive the rating scores for the schedules. Here it would be possible for the Scheduling Tool to use this information to drive the optimization of the schedule for robustness, or the visualisation tool can present a concise visualisation to guide the user to select a schedule.

5.8.2.Further Application and Integration Testing

In addition to normal development testing – it will be advantageous to be able to test the SE in an additional location. Such testing would improve confidence that the SE is capable of functioning outside the well-controlled development environment, and also provide additional resources for identifying shortcomings before being introduced into a factory setting. Currently, it is targeted that the interfaces and core functionality will be tested using the facilities of the SmartFactory in Kaiserslautern, as part of activities in WP6.2.

The *getSimulation()* method of the PERFoRMBackboneInterface includes the ability to send a planned production / order schedule as well as definition of the configurations for the existing machinery. In a first set of testing, the standard interface and the connection to the middleware will be tested by sending and receiving dummy versions of these arguments to the SE, and by returning a dummy version of the simulation results. Then, these tests will be followed by a full execution and demonstration of the SE. The goal of this demonstration will then be to utilize the various capabilities of the SE in combinations that may be utilized within the use cases. For example, the Siemens use case focuses primarily upon flexibility in scheduling, and may not consider reconfigurability of the shop floor. This aspect could, however, be demonstrated using the SmartFactory manufacturing line. A more detailed description of the actual test cases to be evaluated can be found Deliverable 6.2 [22].

5.9. Summary

In this chapter the implementation aspects of the SE that was conceptually introduced in chapter four were shown. This includes, amongst others, the description of input and output file conversions, tool libraries, and interfaces within the SE as well as the interface (API) that is provided to the outside (i.e. the middleware). The integration of two relevant simulation-tools for productions system simulation, Tecnomatix Plant Simulation and AnyLogic, were described. Finally, it was shown how the implementation will be utilized in a use-case specific environment and tested in an industrial test bed scenario.

6. Conclusions and outlook

This document represents the requirements, technical concept and prototypical implementation of a generic SE with a simulation tool wrapper and integration into industrial architecture. The generic SE allows users to execute a configured simulation for their factory, and can be accessed as a service via the standard interface of the industrial middleware. Internally, the SE contains a data model of simulation based elements describing components that can be instantiated to describe a particular simulation. This data model data can be configured for the target usage to either include standard DES elements or user-developed ABS models.

The data model represents the complete simulation experiment configuration ranging from expected performance evaluation KPIs over control logic definition to topology generation and initialization with current resource status from shop floor. This will be used for automatic model generation and execution as well as for result transfer for other modules within the industrial architecture.

To enable the SE to utilize industrial simulation tools, which do not natively support PERFoRM-compliant languages, a translation module provides a process for converting PML into tool-legible languages (and in reverse for outputs).

The generic SE was designed in consideration of functional requirements derived from the four use cases covering a range of industrial applications. These four use cases cover a wide range of simulation tasks, from small batch sizes on the order of 1 to larger batch sizes on the order of thousands. The variety of products to be included also spans multiple orders of magnitude. The boundary conditions regarding the questions that the four use case would like to utilize simulation to answer also vary from simple feasibility evaluation to risk quantification. Thus the generic character of SE fulfils the wide range of tasks covered by the use cases. The concept even allows a flexible extension regarding additional functionalities and modules by serving the wrapper methodology.

As such, the SE as provided in this document describes an enterprise solution that can be successfully applied in a generic context. Within work packages 7-10, this specification and the software framework can be customized and configured to operate for the specific targeted use case conditions.

7. References

[1] I. Grigoryev, *AnyLogic 7 in three days: A quick course in simulation modeling*, 2016.

[2] AnyLogic, *Über The AnyLogic Company.* [Online] Available: http://www.anylogic.de/about-us. Accessed on: Jan. 02 2017.

[3] AnyLogic, *Application Areas.* [Online] Available: http://www.anylogic.com/application-areas. Accessed on: Jan. 02 2017.

[4] AnyLogic, *What kind of license do I need if I .?* [Online] Available: http://www.anylogic.com/editions-comparison. Accessed on: Jan. 02 2017.

[5] A. Borshchev and A. Filippov, Eds., From System Dynamics and Discrete Event to Practical Agent Based Modeling: Reasons, Techniques, Tools, 2004

[6] D. Pawlaszczyk, Skalierbare agentenbasierte Simulation: Werkzeuge und Techniken zur verteilten Ausführung agentenbasierter Modelle: Techn. Univ., Diss.–Ilmenau, 2009. Ilmenau: Univ.-Verl., 2009.

[7] A. Borshchev, The big book of simulation modeling: Multimethod modeling with AnyLogic 6. Chicago: AnyLogic North America, 2013.

[8] J. Barbosa and P. Leitao, "Simulation of multi-agent manufacturing systems using agent-based modelling platforms", in 2011 9th IEEE International Conference on Industrial Informatics (INDIN), 2011, pp. 477–482.

[9] P.Vrba, "Simulation in agent-based control systems: Mast case study", International Journal of Manufacturing Technology and Management, vol. 8, no. 1/2/3, p. 175, 2006.

[10] N. R. Jennings et al., Autonomous Agents and Multi-Agent Systems, vol. 1, no. 1, pp. 7–38, 1998

[11] S. J. Russell and P. Norvig, Artificial intelligence: A modern approach. Upper Saddle River: Prentice Hall, 1995.

[12] M. J. Wooldridge, An introduction to multiagent systems, Reprint. Chichester: Wi- ley, 2008.

[13] M. J. Wooldridge, Ed., Intelligent agents: ECAI-94 Workshop on Agent Theories, Architectures, and Languages, Amsterdam, The Netherlands, August 8 - 9, 1994 ; proceedings, 2. printing. Berlin: Springer, 1995, vol. 890.

[14] S. Kirn, "Kooperierende intelligente softwareagenten", Wirtschaftsinformatik, vol. 44, no. 1, pp. 53–63, 2002.

[15] M. Paolucci and R. Sacile, Agent-based manufacturing and control systems: New agile manufacturing solutions for achieving peak performance. Boca Raton, Fla: CRC Press, 2005.

[16] M. Schönemann et al., “Simulation of matrix-structured manufacturing systems”, Journal of Manufacturing Systems, no. 37, pp. 104–112, 2015.

[17] Schieritz, Nadine and Größler, Andreas 2003. “Emergent Structures in Supply Chains - A Study Integrating Agent-Based and System Dynamics Modeling”. Proceedings of the 36th Annual Hawaii International Conference on System Sciences : 6 - 9 January 2003, Big Island, Hawaii

[18] Deliverable D2.4, " Industrial Manufacturing Middleware: Specification, prototype implementation and validation" PERFoRM Project, 2017.

[19] Simulation of systems in materials handling. VDI Guideline 3633, 2014.

[20] Deliverable D2.3, "Specification of the Generic Interfaces for Machinery, Control Systems and Data Backbone" PERFoRM Project, 2017.

[21] Deliverable D2.3, "Siemens description and requirements of architectures for retrofitting production equipment" PERFoRM Project, 2016.

[22] Deliverable D6.2, "Self-Adaptive Highly Modular and Flexible Assembly Demonstrator Design and Set-Up," PERFoRM Project, 2017.

[23] G. Gordon, “A General Purpose System Simulation Program”, Proc. EJCC, Vol. 20, 1961, pp. 87-104.

[24] A.M. Law, W.D. Kelton, “Simulation Modeling and Analysis”, 1991

[25] J.W. Forrester, “Industrail Dynamics – A Major Breakthrough for Decision Makers”, 1958

[26] J.W. Forrester, “Industrail Dynamics”, 1961