Development of simulation components for material flow simulation of production systems based on the plugin architecture framework EMPINIA

P. Jahr, L. Schiemann, V. Wohlgemuth
HTW Berlin, University of Applied Science, Department of Engineering II
Industrial Environmental Informatics Unit
Wilhelminenhofstr. 75A, D-12459 Berlin, Germany
paul.jahr@student.htw-berlin.de
lars.schiemann@student.htw-berlin.de
volker.wohlgemuth@htw-berlin.de

Abstract
This paper gives an overview on the design and development of simulation components for production systems using the open-source software framework EMPINIA. These components (e.g. machines, transport devices, storages) are created in the course of the EMPORER project, which is funded by the German Ministry for Education and Research (BMBF). The goal of this project is to provide an open-source framework for a fast, easy and lightweight development of applications in the field of environmental management information systems (EMIS), including a detailed support for the solution of certain domain specific problems (e.g. material flow analysis, simulation, handling of hazardous materials etc.). As a part of the EMPORER project an EMPINIA-based material flow simulator called MILAN is developed. It incorporates a discrete event simulation infrastructure with material accounting and management functionality. The goal of this development lies on extensibility of existing simulation and material accounting functionality. Thus new simulation functions and components should easily be added, exchanged or recombined with minimal effort. The concept of material flow simulation combines the approach of production-oriented, order-related simulation and material flow analysis for the application field production systems. To use simulation components for material flow simulation purposes, these components must support both perspectives. This paper covers the structure of the developed production system components in detail as well as the EMPINIA architecture and the material flow simulator MILAN.

1. Motivation
The rational use of goods, such as the production, consumption and distribution is widely known as economic activity. Its improvement is directly connected to the in- and output relations and consists of the attempt to get more returns while investing lesser resources. (cf. Wöhe et al. 2008). This process is also called optimization and it is target-oriented (e.g. optimizing the costs, quality, efficiency or effectiveness). Optimizations can be achieved using an operations research approach (cf. Domschke et al. 2008). The more complex and detailed a system is the less probable is the achievement of an optimized result within the boundaries of an average time frame using methods from operations research (e.g. the problem of “Traveling Sales Man”). In these cases, the usage of simulation methods can be an adequate approach. The result determined by simulations is possibly not the optimum. However approximated results can be deduced. Simulations can help to analyze and understand the behavior of a system and therefore allow the user to play with a model of a real world system to find a result that is relative close to the optimum.

One field of interest which is not a common part of economical simulation efforts yet is the simulation and optimization of material flows (cf. Wohlgemuth 2005). In a production system there are raw materials converted to salable goods with the help of energy, auxiliary materials or other supplies. Furthermore some unintended products like waste and emissions are produced. These materials pass through different stages of the value chain. Raw materials are delivered, processed by machines or workers, combined to products, packed and finally shipped to customers. Working items are stored, transported, sold, used and sooner or later consumed or rather taken out of order and disposed. During
the modeling phase of production systems these processes are modeled using separated components, which can be labeled as building blocks. These components represent the elements (e.g. machines) a product has to pass within his production life cycle. These workstations, the working items, workers and any other constituent part of the system become entities of the corresponding model (as long as they are relevant for the goals a certain simulation study is focusing on). They are designed to be parameterized by the modeler in the course of simulation experiments. Most of the currently used methods to simulate production systems are discrete event-based and process-based simulations (cf. Page et al. 2005).

Typical simulation software in the field of optimization of productions does not offer options to consider environmental and economical goals together based on one consistent model (e.g. high adherence to delivery dates, short throughput time, high workload, low stocks). Energy and material consumptions per production process are normally not representable with low effort (cf. Wohlgemuth 2005). To face this problem, the concept of a discrete event-based material flow simulator was developed, which integrates the material flow perspective with the job-orientated simulation approach for production system. Thus a simulation run can compile results that are relevant for the economic behavior as well as results that describe the impacts of the production system under examination to the environment (cf. Wohlgemuth 2005, Wohlgemuth et al. 2006). This enables the assistance of operative decisions with strategic and tactical considerations for both economical and environmental aspects (see Figure 1).

To set the concept of MILAN on a stable and new basis and to use the advantages of modern software paradigms (e.g. reuse, platform independence, modularity, individual product design by the user) the described approach is currently further developed in the course of the EMPORER\(^{4}\) project. The new development of the material flow simulator MILAN is based on the open-source plugin framework EMPINIA\(^{5}\) (comparable to the Java framework Eclipse\(^{6}\)). EMPINIA, which is also developed in the course of the EMPORER project, is designed for the development of complex domain-specific applications especially in the field of environmental management information systems (EMIS) (cf. Wohlgemuth et al. 2008). It is a component-orientated extensible application framework based on Micro-

---

\(^{4}\) http://www.emporer.net

\(^{5}\) http://www.empinia.org

\(^{6}\) http://www.eclipse.org
softs.NET technology with the purpose to support and simplify the development of complex software systems.8

For MILAN it is necessary to provide libraries of simulation components (e.g. for production systems: machines, transporters, system boundaries) which enable the modeler to represent and simulate his system adequately. These components can be added to an application i.e. as building blocks via a plugin mechanism and thus can be used to build a user-specific model. This may lead to an easy development of user-specific components with low dependencies and an attachment to a modeling tool box for a certain application field, which is not possible with other simulation tools (cf. Page et al. 2000 & 2005). These components can either be generally applicable or might be used for very specialized purpose. Specialized entities are developed for a whole production sector (e.g. semi-conductor sector with coater, stepper and dispatcher) (cf. Wohlgemuth et al. 2004 & 2005) or they represent a production component of a certain company with its specific parameters. In contrast general components are highly abstracted and are applicable for many production systems (cf. Jahr et al. 2009). The goal of this project was the development and implementation of such general entities for MILAN.

In this sense this paper describes the components, a.k.a. plugins, which build MILAN besides EMPINIA and explains the development and structure of general production system entities. It further presents how the discrete event simulation for material flows as well as the modeling within the MILAN software works.

2. MILAN

The material flow simulator MILAN is build upon a set of highly specialized extensions for the plugin framework EMPINIA. Besides the components which come with EMPINIA there are many plugins taken from the EMIS toolbox and combined with MILAN. These are shown in Fehler! Verweisquelle konnte nicht gefunden werden. Furthermore it is shortly explained how these plugins are stucked together.

The simulation capabilities of the MILAN software consist of the following elements:
• The simulation core consists of the central simulation service, interfaces and abstract base classes for models, experiments and model entities. These are used in each kind of simulation. The simulation service provides models and experiments in a way that other software parts can use them. The simulation core gives models and their entities access to the functionality of a domain model service. A domain model defines the domain of an EMPINIA-based application, its elements and their relations as well as rules that apply to this domain. MILAN consists of the domain 'simulation' with elements like 'model' and 'entity'. Among other important functionalities the domain service provides possibilities to persist its elements. That is the reason why this service is used in MILAN to save and load formerly created models.
• A bundle for discrete event simulation extends the simulation core with classes specific to the discrete event simulation approach. These classes are using an EMPINIA extension that enables the development of logical graphs in order to combine entities of a model to a network diagram. The basic generic experiment component is extended with an event list and a scheduler, which are used to simulate time in discrete steps.
• The simulation components have access to many stochastic distributions (e.g. Normal, Bernoulli). They are used to generate streams of random numbers, for example to schedule an event which follows a certain arrival probability. Additional to these existing distributions user-defined distributions can also be added via plugins.

---

7 http://msdn.microsoft.com/de-de/netframework/default.aspx
8 More information about the design of applications with EMPINIA can be found in this book in the article: Conceptual Design and Implementation of a Tools Platform for the development of EMIS based on the open-source plugin-framework Empinia.
The material flow capabilities of the MILAN software consist of the following elements:

- A **material management** component allows creating, deleting and organizing materials as well as bill of materials. A user interface view for this kind of data is added through which the modeler can show and manipulate these items.
- The **material accounting** system accesses the material management to add or remove a custom amount of materials of a certain type. By its means it is possible to show, save and manage material and energy bookkeeping resulting from the simulation. Accounting rules define these bookkeeping which can be added by the user to all kind of discrete events in combination with relevant model components. Definition and structure of rules are defined with help of the extension mechanism as extensions. Afterwards these rules can be used and configured. Following the observer design pattern which is used by the material accounting system the simulation components do not 'know' anything about material flows and the bookkeeping of material input and output data. Thus a developer of simulation model components does not necessarily have to consider these mechanisms. Indeed the simulation operates independently of any material bookkeeping issues. The material flow system just listens to the simulations progress and reacts to occurring events if one of its rules applies.

The common features of the MILAN software consist of the following elements:

- Graphical manipulation of building blocks leads to a faster development of a model. The **graph editor** can be used to manipulate and create models. The editor itself can work in different domains. Domain specific functionality and the graphical representation have to be defined by plugin developers enabling the editor to handle new domains and their components, which are also using plugin definitions.
- Manipulating model parameter for the simulation and material flow perspective is done by means of **property editors** enabling a simple and consistent way of setting values for all types of properties. For the production system domain there are standard editors implemented. These allow the change of component specific parameters like setting distributions, accounting rules, queue lengths or capacities etc.
• No analysis can be done without results. These are shown in reports which can be designed with the help of the **reporting** system. The data for the reports is aggregated during simulation runs by a system of observers that listen to changes in the material accounting and simulation entities.

![Figure 3: Screenshot of MILAN application based on EMPINIA](image)

An example of the graphical interface that users will get with MILAN is shown in Figure 3. You can see the simulation explorer with all the model components on the left side, below an overview map of the current model; in the middle the model editor with all available model entities in the toolbar, below a list of events which are currently scheduled; the experiment control for setting experiment parameter on the right side above a property editor for parameter setting of a model component (currently a workstation is selected).

The execution of a material flow simulation requires the development of a model that represents the system under investigation. Up to now this requires two models, one for the material flow analysis and another one for the simulation-related aspects. The material flow simulator MILAN however is able to integrate both specific views into one model. It retains the common model structures and adds the different sets of parameters. These parameters like sets for material accounting or probability distribution streams can be added subsequently to the build a model structure.

The modeling is done using a graphical network consisting of nodes and edges. The nodes describe important model elements, where products are handled or stay for processing for a certain period of time. Edges work as logical connections between these elements and are also intended to show the process flow direction.

Once the modeling is complete the parameterization of the particular stations can be started. Depending on a specific model element these parameters can be very different from each other. Workstations for example have a processing-time, conveyors a loading and transportation time. In cases where no suitable standard component to represent the actual system can be found, new components can be added using the extension mechanism. This enables developers to create their own components or to use component libraries from other parties. The modeling of material and energy flows is done with the help of appropriate rules. According to these rules a certain set of materials is accounted depending
on the occurrence of previously assigned events. At the end of the modeling phase the duration of the simulation can be chosen. From material flow perspective this can be the same as a balancing period.

As soon as the right model structure, parameters and times are configured the material flow simulation is ready for simulation experiments. Experimentation can be done by variation of parameters. This enables the user to experiment from the material flow perspective and the order-based simulation perspective at the same time using only one model and one application. Once a simulation ends its data can be visualized in a reporting view (e.g. balance sheets and simulation results). The user is now able to analyze if a simulation run is better than another.

3. Simulation components for production systems

Based on the simulation infrastructure and the discrete event simulation approach the next step of the development of MILAN was the implementation of a component package (in EMPINIA terms, a 'bundle') for the simulation of production systems. The included components extend the existing infrastructure in different areas.

It was necessary to introduce an extended type of model, which covers the specific requirements of the production system simulation. The production model structure is organized as a directed graph. The nodes of the graph represent stationary elements of the production system (e.g. workstations, conveyors, storages). The edges define the order of the elements and therefore define the way products can travel through the model. During a simulation the products are created inside the system and travel along the edges. Inside the stationary elements, products can be transformed, removed and new ones can be created. Processes represented through stationary elements are modeled using samples of stochastic distributions for the duration of each process. When a process is finished the processed products are sent to the next available stationary element. The ability to send and receive products is common to all stationary elements.

While the simulation is running the internal behavior of a stationary element can lead to a state where it is temporarily unable to accept incoming products. At the current development stage, there are several states available, which can be used in stationary elements, like 'Failure', 'Maintenance' and 'Setup'. The number of these states and the activities that change them depend on the implementation of the stationary element component. The activities themselves are implemented as extensions and attached to the component extensions, so that new activities can easily be added and used for the development of new or more complex simulation components.

At the moment there is a basic set of stationary elements implemented, which are offering a very broad set of capabilities for modeling and simulating production systems. The selection of these elements has been influenced by the work of Kosturiak et al. 1995, Wohlgemuth et al 2004 & 2005 and Banks et al. 2005.

The current set of available stationary elements contains:

- **Workstations** represent all types of machines, which perform tasks on products using a certain amount of simulated time. They can be configured to transform a set of products into another set using specific rules that define the amount of incoming and outgoing products. Workstations can break down (so the will be unavailable for a certain amount of time), can be maintained or can have a setup duration after they were activated or out of order.

- **Conveyors** are one way to model the transport of products between stationary elements. The transport duration is the same for each product as long as no disturbance (failure, maintenance, all successors unavailable) occurs. These break down durations are added to the processing time. There are two different conveyor implementations – one which accumulates products at its end and one which completely stops the movement of products when blocked (downstream elements are unavailable).

- **Transporter systems** define the second way of moving products between stationary elements. They use a configurable set of transport units (e.g. trucks) to transport products. Each transport process can take a different duration to finish. The transporters are so called resources that are defined inside the model using a resource pool. The resources of such a pool can be shared by several transporter systems.
Buffers are stationary elements that store products until they are needed by downstream stationary elements. They have a configurable capacity and become unavailable if this capacity is attained.

Synchronization points are stationary elements that can be used to combine parallel production chains or divide one stream of products into different chains using the same transformation rules that the workstation component uses. They use an internal queue to store products until a transformation rule can be satisfied.

Entry points define one kind of system boundary of a model. Entry points are creating products according to stochastic distributions (arrival times) and are feeding these products into the model.

Exit points mark the other kind of system boundary where products are removed after they completed their way through the production model. Both system boundaries will later be extended to act as points of intersection to a super ordinate model, if the containing model is itself a submodel and therefore has to act like a model component itself.

To use one of the mentioned classes according to EMPINIA's extension mechanism, the class is appended with a .NET Attribute ('Extension'). In this way the EMPINIA-runtime environment can perceive it as extension of the simulations components. An example of the usage of the attribute in the C# programming language is shown in Figure . The identifier (Id) of the extension is communicated to the attribute via a parameter. Thus the Ids of the incorporating bundles as well as the extension points and the type are given to it.

```csharp
[Extension("Id of the extension", "Id of the containing bundle",

"Id of the extension point", typeof (Workstation))]

public class Workstation: StationaryElement, Iworkstation
{
...
}
```

Figure 4: Extension configuration of a stationary element (workstation)

Another option for this kind of definition is the possibility to write the extensions in xml-code. This can be done within the configuration/definition file of the bundles (Bundle.xml). Each extension point has an XML-schemata to validate the descriptions of the extensions.

To be properly integrated into the simulation infrastructure the class has to implement various interfaces. Using a component via the interfaces enables the framework to use the component in different layers of the simulation domain. To be used in the production system domain for example, the component needs to be a simulation entity and therefore needs to implement the specific entity functions. It also has to support its usage in a net model which can be assured by implementing functions of a net node interface.

As stated above the components representing stationary elements use a base class that models their common basic behavior. For their respective special possibilities they use a set of support classes. For example, these support classes extend the stationary elements to be able to fail during processing, to be able to be maintained or similar behaviors. Each support class models a specific state in which a stationary element can change during a simulation. Supports handle the creation of their related events (in most cases a start and end event, that turn the represented state on and off). When a component turns into the supported and state how long it remains in this state has to be defined in the owning component. The intention behind the partition of the simulation components into these support classes is the reuse of frequently needed functionality whose usage only differs in the way it is activated and deactivated.

157
4. Conclusions and visions

During the EMPORER project the plugin architecture EMPINIA was already developed. As a further result of the EMPORER project the material flow simulator MILAN was developed which is based on the EMPINIA software framework. New components for specific application areas are still under development. Another important result of the EMPORER project is the implementation of very abstract simulation entities for the analysis of production systems. These entities enable users to model and simulate a broad set of production systems. Because of their modularity and the plugin mechanisms of EMPINIA it is very easy to add more specialized entities to the production system’s domain and to use them for a material flow simulation.

At the moment the production components are verified by performing a simulation study in a company that produces solar panels. The problems, results and experiences of this validation will be used to improve and enhance the components, the simulation infrastructure and MILAN as a simulation tool, itself.

Acknowledgement

This research project is funded by the German Federal Ministry of Education and Research (BMBF). The authors thank for the support.

References


