DEVELOPMENT AND IMPLEMENTATION OF A DOMAIN MODEL FOR PERSISTENCE AND CODE GENERATION FOR CONSISTENT MATERIAL FLOW SIMULATIONS WITH MILAN

Paul Jahr, Lars Schiemann, Volker Wohlgemuth

HTW Berlin
University of Applied Science,
Industrial Environmental Informatics Unit
Wilhelminenhofstr. 75A, 12459 Berlin, Germany

paul.jahr@student.htw-berlin.de
lars.schiemann@htw-berlin.de
volker.wohlgemuth@htw-berlin.de

Abstract

This paper gives an introduction to the domain model of the material flow simulator MILAN, showing the advantages of domain driven design. MILAN incorporates a component based event discrete infrastructure with material flow analysis functionality. It is built on the open source plugin-based rich client platform EMPINIA. MILAN extends the framework towards the specific field of material flow simulation. The concept of material flow simulation combines the approach of production-oriented, job-related simulation and material flow analysis for the application field production systems. EMPINIA offers a Domain Specific Language (DSL) to define a domain model. This approach simplifies the complexity of designing the domain logic providing best practice and good tested code. The domain, for which the material flow simulator MILAN was developed, consists of model, experiments, simulation entities and material flow definitions. Additional simulation entities and their relations can be developed and added to MILAN respectively existing components can be customized much more easily utilizing this mechanism.

Keywords: Event Discrete Simulation, Production Systems, Plugin-based Components, EMPINIA Framework, Material Flow Simulation

Presenting Author's biography

Paul Jahr is a master student and Lars Schiemann is a master of science of industrial environmental informatics at HTW Berlin. Both are core developers of the EMPINIA framework since 2006 and are professionals in Microsoft .Net. Together they designed and developed most parts of the event discrete simulation tool MILAN.

1 Introduction

The material flow simulator MILAN was developed during the EMPORER project which was funded by the German Ministry for Education and Research (BMBF) from 2006 to 2009. The goal of this project was 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.) (cf. [7] and [10]). One outcome is the EMPINIA framework which is freely available under [11]. EMPINIA supports rapid application development with a highly extensible RCP, like Eclipse (cf. [12]), based on the Microsoft .NET technology.

The main function of the EMPINIA framework is to offer a plugin mechanism where developers can add any kind of extension. An extension is always added to an existing extension point. Extensions themselves can be further extended if they define their own extension points. EMPINIA provides modules for designing an application with graphical user interface (GUI) and other standard software components. This enables application developers to concentrate on their specific problem domain instead of doing repetitive software developing tasks, such as persistence, logging, application and user settings, message notification and user interface design.

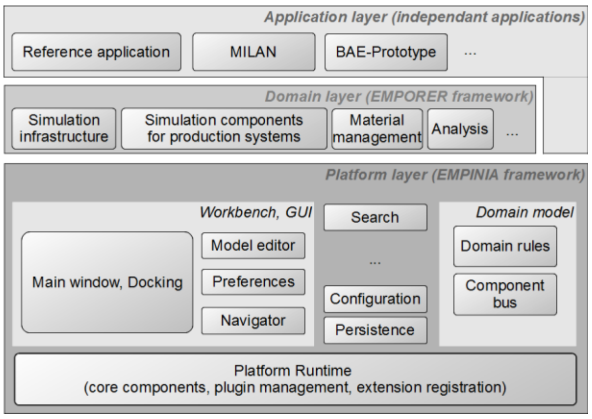
Application layer (independant applications)
Reference application
MILAN
BAE-Prototype
...
Domain layer (EMPORER framework)
Simulation infrastructure
Simulation components for production systems
Material management
Analysis
...
Platform layer (EMPINIA framework)
Workbench, GUI
Main window, Docking
Model editor
Preferences
Navigator
Search
...
Configuration
Persistence
Domain model
Domain rules
Component bus
Platform Runtime
(core components, plugin management, extension registration)

Fig. 1 EMPINIA and EMPORER architecture

EMIS are complex applications which combine many different fields of interest. Developing an EMIS is difficult, expensive and needs interdisciplinary knowledge (cf. [8]). The approach of the EMPORER project was to reduce this effort by also creating a set of extensions, which address common needs of many EMIS problems. The result is the EMPORER framework/toolkit, which offers independent parts like material management, simulation capabilities and data analysis functionalities (cf. Fig. 1).

2 The Material Flow Simulator MILAN

MILAN was implemented as a prototype for an EMIS application based on EMPINIA. It combines the ecological material flow analysis perspective with economical job-oriented simulations of a certain under investigation, e.g. a production plant. It includes the discrete event simulation infrastructure with material-accounting and -management functionality and so facilitates optimizations from economical and material flow based perspective at the same time. Thus, it respects economical impacts on ecological changes and vice versa (cf. [8] and [9]).

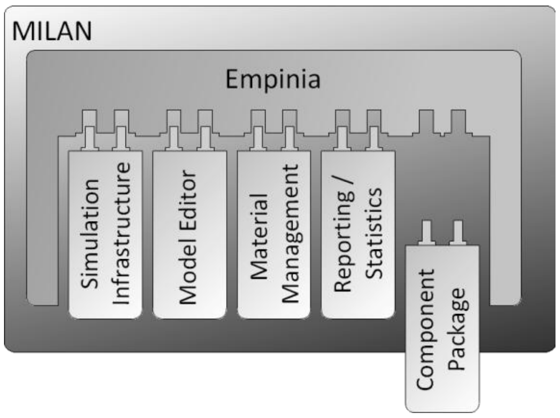

Fig. 2 MILAN as a set of EMPINIA components

MILAN is build upon a set of specialized extensions of EMPINIA. Besides the components which come with EMPINIA there are some EMIS specific extensions taken from the EMPORER EMIS toolkit and combined into MILAN (cf. [2], [5] and [6]). The architecture is shown in Fig. 2 and described in the following part.

2.1 Infrastructure

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 discrete time steps.
- The simulation components have access to many **stochastic distributions**, e.g. Normal, Bernoulli, and Exponential. 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.

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.

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. The bookkeeping is realized using accounting rules, 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.

For MILAN it was necessary to provide libraries of simulation components (e.g. for production systems: machines, transporters, storage) to give the ability to simulate a common production system with MILAN. Simulation entities can be added to MILAN as building blocks via the EMPINIA plugin mechanism to build a user-specific model. These components can either be generally applicable or might be used for a very specialized purpose. Specialized entities are developed for the production sector (e.g. semi-conductor sector with coater, stepper and dispatcher; cf. [8]) or they represent a concrete production machine of a certain company with its specific parameters. In contrast general components are highly abstracted and are applicable for many production systems. The parameterization of these entities is more abstract but closer to the specific problem then a common simulation language or standard simulation software. An abstract set of components for production systems comes with MILAN and is ready for modeling production systems (cf. [2] and [6]). The components were influenced by [3].

While a simulation runs, the internal behavior of such a simulation entity can take over different states. These states represent a passive or active activity. At the current development stage, there are several states/activities available, which can be used in entities, like 'Failure', 'Maintenance' and 'Setup' for all kind of machinery. The states are implemented as plugins and attached to the entity extensions, so that new states can easily be added and reused for the development of new or more complex simulation components. Today, the following parts are available.

- **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 breakdown (so the will be unavailable for a certain amount of time), can be maintained or can have a setup duration.
- **Conveyors** are one way to model the transport of products between simulation entities.
- **Transporter systems** define the second way of moving products between entities. They use a configurable set of transport units (e.g. trucks) to transport products. 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 entities that store products until they are needed by downstream simulation parts.
- **Synchronization points** can be used to combine parallel production chains or divide one stream of products into different chains using specific transformation definitions.
- **Entry points** define one kind of system boundary of a model. Entry points are creating products according to stochastic distributions 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.

2.2 Simulation with MILAN

The execution of a material flow simulation requires the creation 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 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 analysis perspective this can be the same as a balancing period in a static material flow analysis model.

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. 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 from the material flow perspective and the job-based simulation perspective at the same time.

3 Domain Model

Domain entities are software objects. But in difference to “normal” software objects in an object-oriented manner they are not used to build the whole application infrastructure. They only represent distinct parts of the problem domain, e.g. for simulation: entity, model, experiment and their parameterizations.

3.1 Architecture and implementation

The domain layer provides a central and consistent access to the entities from all over the EMPINIA platform. All changes of domain entities (add, remove, update) are first validated via user defined domain rules and can also be rolled back on demand before they are persisted. In the simulation context this means you can change the modeling capabilities by defining a domain specifying entities and rules. This rules can be used to validate inputs or restrict the remove- and adding-process of new simulation component instances respecting their dependencies. Modifications on the components interaction capabilities can be done by rewriting the specific rule definition or add a new one, compile the plugin containing this rule and replace the old plugin with the new one. After a restart, the application will be able to apply the new rules.

Additional Domain entities and rules can be added like every extension in EMPINIA without touching the compiled MILAN software itself. Newly created features are validated, initialized and their functionality is provided in the user interface at runtime of the application after the containing package was added. Extending EMPINIA can be realized using .NET class attributes or XML-declarations.

To create the domain model EMPINIA supports developers in many ways. As mentioned before, applications based on the EMPINIA framework can be extended by creating so called extensions. An extension can either be any kind of configuration or new program code (written in one of the many .NET programming languages). The simulation infrastructure provides a set of extension points. As mentioned in 2.1 the infrastructure itself, as well as the event-discrete simulation domain are extensions to the basic framework functionality. To be able to define production system components in this domain, all currently required entities had to be built as EMPINIA extensions. Most of the more complex object-oriented classes are composed of a set of reusable basic building blocks that for example, define different states an entity can be in during a simulation run. This part of the creation of domain entities requires very specific knowledge in the domain of event-discrete simulation and had to be programmed manually. Exactly this task can (and should) be the focus of developers of simulation components.

3.2 Domain Code Generation

Besides their logic (behavior) many of the properties of the simulation components (structure, state) needs to be persistable (e.g. to be able to safe a model instance with all its containing entities and their initial setup or to save results of a simulation run). Often code in this area (especially for properties with simple, single-valued types) has a very similar structure and can be generated by a code generator. Generally, a code generator is a program that can transform some type of structure definition into source code written in one or more specific programming languages (e.g. recurring implementation details such as classes, interfaces, properties). EMPINIA provides a code generator, which is able to produce a variety of code artifacts needed during the development of domain entities. It can generate huge parts of code to access and observe the domain entities properties conveniently. Because a code generator always generates the same kind of structures based on its available generation abilities (templates), all of its newly generated artifacts automatically gain quality if the underlying templates get improved. The generator can be extended with additional functionality like the generation of domain object hierarchies, visual representations in graphical editors or dialogs and many more. Because the generation process itself is very simple (string concatenation of templates with some variables) it is executed in a fraction of the time which would be necessary to write the code by hand. This also greatly reduces the occurrence of errors because mostly the cause can be found in one of the templates.

```
<entity
  name="ExampleEntity"
  baseType="Empinia.Persistence.DomainEntity"
  namespace="Htw.Simulation.Discrete"
  accessRestriction="public">

  <attribute
    name="DomainType"
    parameters="ExampleEntityExtensionId,    _
         Htw.Simulation.Discrete.Bundle.API,_
                    typeof(IExampleEntity),_
                     typeof(ExampleEntity),_
           Name = &quot;ExampleEntity&quot;"
    target="Class" />

  <property
    name="Name"
    accessRestriction="public">
    <type
      name="System.String" />
  </property>

  <property
    name="AnotherProperty"
    accessRestriction="public">
    <type
      name="System.Int32" />
  </property>


  ...
</entity>
```

List. 1 Simplified example for a domain code generation definition for parts of a domain entity

In List. 1 an excerpt of the definition of a simple example domain entity written in XML is shown. It defines a domain entity called "ExampleEntity". The class should inherit from a base class with common domain entity behavior (not mandatory). It should consist of two properties ("Name" and "AnotherProperty"). Furthermore the class should be decorated with a class attribute that declares the class explicitly as an EMPINIA extension to the domain type extension point.

The declaration of such structures is a subset of the domain code generation DSL. Given an input like that, the code generator transforms it into a set of source code artifacts depending on the requested templates using the defined structure. To choose the generated functionality the code generator can be configured using another subset of the DSL which refers to the selection of artifact types. This can be relatively simple things like classes, interfaces, methods and properties, but also whole class structures, configuration sets and even tests. As long as a working template is available and the input is correct, the generator will produce compliable source code and valid configuration files.

3.3 Domain Model Persistence

Almost every professional application obviously needs a mechanism to save the state of the users work (e.g. a document, table, record and so on). With a domain model, saving the state of domain entities can easily be done using an own software layer (persistence layer). Advantages of this approach are the concealment of the complexity of persistence implementations (e.g. relational databases, binary or text files) and the possibility to change the complete implementation if it does not fit the needs of the developer anymore. The domain model used in this application provides an automatic persistence mechanism to handle such functionality. Properties of a domain entity, which should be persisted, are defined using a persistence-specific DSL. A persistence mapper transforms the configuration into commands to the underlying persistence backend. It saves, creates, reads and modifies domain entity instances in a consistent way. The data container is variable (file, database, webservice etc.). The current implemented persistence foundation uses the widely used open source object-relational mapper NHibernate (cf. [13]) which converts runtime objects in memory into records in a relational database and vice versa. Its configuration is done in a similar manner to the domain DSL described above. Again definitions are written in XML, using a specific set of elements that describe relations between classes and their attributes to tables and columns. An example of such a mapping definition can be seen in List. 2.

```
<hibernate-mapping
 assembly="Htw.Simulation.Discrete"
 namespace="Htw.Simulation.Discrete"
 xmlns="urn:nhibernate-mapping-2.2">

 <class

 name="Htw.Simulation.Discrete.ExampleEntity"
  table="EXAMPLE_ENTITIES"
  ...>

  <id
   name="Id"
   column=" EXAMPLE_ENTITY_ID">
   <generator
    class="native" />
  </id>

  <version
   name="ObjectVersion"
   column="NHVersion"
   type="Int32" />

  <property
   name="Name"
   type="System.String"
   length="255" />

  <property
   name="AnotherProperty"
   type="System.Int32" />

  ...

 </class>
</hibernate-mapping>
```

List. 2 Simplified example for an object-relational mapping definition (NHibernate 2.2)

The example is related to the previously used sample of a domain entity ("ExampleEntity", cf. List. 1). It shows the mapping of the class to a table called "EXAMPLE_ENTITIES". In the lower part both properties are declared. The other two declarations are persistence specific. The element "id" defines a column that contains the primary key of the table which can be used in conjunction with other entities persistent data (stored runtime instance relations). The attribute "native" tells the underlying database management system should use its own build-in algorithm to generate unique values for primary keys. The "version" element creates a special table column that contains a version number for the record to track changes made to the stored entity instance.

The example only shows a very small subset of the possible mapping possibilities of NHibernate 2.2. It is obvious, that the shown code could easily also be generated during the domain code generation. The generation of persistence DSL artifacts will probably implemented into the domain code generation DSL very soon.

4 Vision

The code generation can be used for handling repetitive work and reduce it for a developer. In EMPINIA many components are used by domain entities. Such usage can be generated in the future. This concerns for example visual representation of the domain component in editors and dialogs of the user interface of the EMPINIA-based application.

Designing a domain model via user interface editors would be a nice feature to avoid mistakes in writing XML-declarations. The bypass over XML can then be dropped. A possible realization for example could be a Microsoft Visual Studio add-in which works like the integrated class diagram editor in Visual Studio.

The persistence mechanism of EMPINIA is currently restricted to relational databases accessed by using NHibernate. In some cases a more lightweight implementation would be preferable e.g. ASCII-serialization like XML or JASON. The opportunity to switch between these mechanisms will increase the field of operation of EMPINIA.

The design of simulation entities is connected with high development effort. To lower this work the activity/state extensions in combination with code generation can be equipped to designing a component with the visual graph editor of MILAN. This can be realized for example like in LABView (cf. [14]).

5 References

[1] Busse, T.; Denz, N.; Page, B. (2008): A plugin-based framework for domain models and persistence in environment management information systems. In: Möller, A.; Page, B.; Schreiber, M. (Eds.) (2008): EnviroInfo 2008 -

Sustainable Development and Risk Management - Proceedings of the 22nd International Conference on Environmental Informatics for Environmental Protection. Lüneburg. Shaker Verlag, Aachen, S. 593-602

[2] Jahr, P.; Schiemann, L.; Mäusbacher, M.; Panic, D.; Schnackenbeck, T.; Wohlgemuth, V. (2009): Development of simulation components for material flow simulation of production systems based on the plugin architecture framework EMPINIA. In: Wohlgemuth, V.; Page, B.; Voigt, K. (Eds.): EnviroInfo 2009 - Environmental Informatics and Industrial Environmental Protection – Concepts, Methods and Tools. Proceedings of the 23nd International Conference Environmental Informatics, Volume 2, HTW Berlin. Shaker Verlag, Aachen, p. 161-169

[3] Košturiak, J.; Gregor, M. (1995): Simulation von Produktionssystemen. Springer Verlag, Wien.

[4] Page, B.; Kreutzer, W. (2005): The Java Simulation Handbook: Simulating Discrete Event System with UML and Java. Shaker Verlag, Aachen.

[5] Panic, D.; Schnackenbeck T.; Wohlgemuth V. (2008): Erweiterung eines Open Source Rahmenwerkes um Simulationsfunktionalität für betriebliche Umweltinformationssysteme. In: Wittmann, J., Wohlgemuth, V. (Eds.): Simulation in den Umwelt- und Geowissenschaften. Shaker Verlag, Aachen, p. 127-137

[6] Schiemann, L (2009): Implementierung eines Stoffstromsimulators für Produktionssysteme auf Basis des Open Source Rahmenwerkes EMPINIA. Masterthesis, HTW Berlin.

[7] Schnackenbeck, T.; Mäusbacher, M.; Panic, D.; Wohlgemuth, V. (2009): Conceptual Design and Implementation of a Tools Platform for the development of EMIS based on the open-source plugin-framework EMPINIA. In: Wohlgemuth, V.; Page, B.; Voigt, K. (Eds.): EnviroInfo 2009 - Environmental Informatics and Industrial Environmental Protection – Concepts, Methods and Tools. Proceedings of the 23nd International Conference Environmental Informatics, Volume 2, HTW Berlin. Shaker Verlag, Aachen, p. 149-154.

[8] Wohlgemuth, V. (2005): Komponentenbasierte Unterstützung von Methoden der Modellbildung und Simulation im Einsatzkontext des Umweltschutzes. Shaker Verlag, Aachen.

[9] Wohlgemuth, V.; Kreutzer, W.; Page, B. (2006): Combining discrete event simulation and material flow analysis in a component-based approach to industrial environmental protection. In: Environmental Modelling & Software. Elsevier, New York, p. 1607-1617

[10] Wohlgemuth, V.; Schnackenbeck, T.; Panic, D.; Barling, R.-L. (2008): Development of an Open Source Software Framework as a Basis for Implementing Plugin-Based Environmental Management Information Systems (EMIS). In: Möller, A.; Page, B.; Schreiber, M. (Eds.): EnviroInfo 2008 - Environmental Informatics for Environmental Protection. Proceedings of the 22nd International Conference Environmental Informatics - Informatics for Environmental Protection, Sustainable Development and Risk Management, September 10-12, 2008, Leuphana University Lüneburg. Shaker Verlag, Aachen, p.584-592

[11] http://www.empinia.org

[12] http://www.eclipse.org

[13] http://www.nforge.org

[14] http://www.ni.com/labview