Tool Support for Distributed Management of Simulation Models and Evaluation Data

Marc Störzel and Ursula Wellen
Department of Computer Science
University of Dortmund
Baroper Str. 301, 44227 Dortmund
Germany
E-mail: {mstoerzel,wellen}@ls10.cs.uni-dortmund.de

KEYWORDS
distributed data management, evaluation data, simulation, Process Landscaping.

ABSTRACT

This article presents a software system supporting the evaluation of simulation data generated for the analysis of communication between locally distributed processes. The corresponding process modeling language is based on Reference Nets, a special type of high level Petri Nets, which allows the modeling of communication between different net instances with synchronous communication channels. For the simulation of this type of Petri Net we make use of and extend the software system Renew, a Java-based Petri net tool.
As simulation often generates mass data, our approach is to store simulation results separately from data describing structural features of process models. We use Renew's graphical editor and simulation component as a basis, implement the feature of storing simulation data by using a database server and extend the tool further with an evaluation component. We discuss the advantages of separation of the different data types and present a framework developed for the reintegration of these data streams. With an example, we show how algorithms for the evaluation of simulation data can benefit from this approach.

INTRODUCTION

In order to analyze the simulation data of process models for optimization purposes it is not sufficient to store the simulation results in a database and to compare different cycles. It is necessary to extend this data with knowledge about structural features of the considered processes, such as the amount of process models and their parameterization. This second type of data is often already stored separately from simulation data, e.g. as part of the modeling data. It can be extended with further information about the underlying process models to analyze better simulation results. Therefore, a set of process models, called process landscape (Gruhn and Wellen 2001), represented by Petri Nets, is simulated with varying parameterizations mapping concrete locational distributions of the processes, where the Petri Nets structure stays mostly unchanged (Störzel 2001). In this paper we discuss a software architecture supporting this approach by separating structural data from simulation data and reintegrating both data streams for evaluation purposes.

The underlying process modeling method we use for the development of distributed process models, called Process Landscaping (Gruhn and Wellen 2000), makes use of Petri Nets allowing the modeling of communication interfaces between nets in a sufficient and comfortable way. Reference Nets (Kummer 19999), a special type of timed colored Petri Nets (Jensen 1992), support the modeling of communication between different net instances with synchronous channels, where communication systems can be modeled as separated nets. Renew (Kummer and Wienberg 2000) is a software tool consisting of a graphical editor for the modeling of Reference Nets and a simulation component, who's graphical output may be switched off for performance reasons. We took Renew as a basis and extended it with

- an XML-based approach for the parameterization of experiments as a set of simulation cycles,
- project-oriented management of the hierarchical structure of both, process model and simulation data, where a project consists of one or more experiments,
- database-driven recording of mass simulation data, and
- preparation and stochastically evaluation of simulation cycles.

We have chosen the method of discrete event-driven simulation (Law and Kelton 1991) for our analyses because we focus on the communication between process models where each receiving or sending of a message is modeled as an event. Even when hybrid simulation techniques are engaged (e. g. (Donzelli and Lazeolla 2001)) the communication itself is modeled in a discrete manner. In Petri Nets, each firing of a transition is handled as an event which may enable other transitions, and this may result in further events. Analyzing the nets statically is not applicable because of the models' complexity and restrictions for timed transitions (Beckmann 1997). With respect to stochastic characteristics of simulation cycles we clustered several cycles with unchanged parameters to an experiment and experiments with varying parameterization to projects.
There are several studies comparable with our approach e.g. in the area of climate research (Lautenschlager 1995). In this area we can also find separation of data storing. But in difference to our approach Lautenschlager stores data concerning the model's structure within a relational database management system (RDBMS), and the measured climate data (about 60 TByte each year) is stored as binary large objects (BLOBs), i.e. unstructured bytestreams.

This comparison also shows that the approach of separating model data and measured (simulation) data is not restricted to Petri nets. Petri nets are often used to analyse dynamic behaviour based on static structures, like it is the case for our process landscape. But they are not the only suitable notation resp. specification language for modeling and simulation purposes.
For example Tolujew et al. (Merkuryeva et al. 1998), Merkuryeva, Merkuryev and Tolujew 2000) start with modeling flow charts which are translated into a computer simulation program using SIMAN language block-diagrams. The authors analyse logistic processes at a container terminal. Similar to our approach, they also use special evaluation components after the simulation run.
The software architecture for our simulation and evaluation purposes is described in more detail in the following section. It presents a global view of the different components supporting the process of parameterization, simulation and evaluation of a process landscape. Section 3 describes our concept of distributed data management in more detail and explains the motivation for this approach. Both types of data storage are handled in a separate subsection. The process of reintegration is shown by an example algorithm in a third subsection. Finally, in section 4 the advantages of this approach are summarized and an overview to our future research is presented.

STRUCTURE OF TOOL SUPPORT

The basis for our work is the software tool Renew. For science and research purposes its Java-based source code is available for free. It consists of a comfortable modeling tool and a performant simulation component. In version 1.4, which is the version we started with, a coupling to a database system was part of the architecture's specification but not yet implemented. We implemented this interface and added a further component for a project-oriented management of data and an evaluation component. The resulting software architecture is depicted in figure 1, where the additional parts are marked gray, namely the evaluation component, the data base and the report component.
The process of analyzing the complete model of all communicating processes, called process landscape, can be divided into 5 steps:

1. modeling process models as a set of Petri Nets
2. modeling further Petri Nets representing communication interfaces between the different process models
3. parameterization of all Petri Nets with respect to dynamic communication aspects
4. running several simulation cycles
5. evaluation of simulation data and analysis of results

This process is supported by our software tool. It supports the modeling, parameterization simulation and evaluation of a process landscape and creates a so-called project file. This file holds references to further files representing net descriptions. The editing of this file is not yet implemented within our software tool, but due to the fact that it is an XML file with a defined DTD it can easily be modified with any editor.

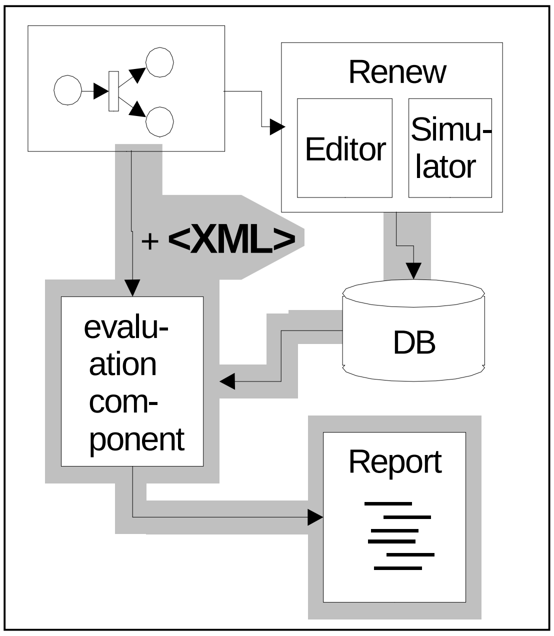


Figure 1: Integration of tools

With the graphical editor each net can be modified in a comfortable way. The proper simulation of the complete set of Petri Nets can be controlled by graphical output of each simulation run before a simulation for evaluation purposes is started. For the latter, the tool initiates a database connection and starts an adjustable count of simulation cycles. After the simulation data has been computed an evaluation can be initiated. To stochastically evaluate the simulation results, the evaluation component makes use of both the mass simulation data and the additional information stored in the project file.

DATA MANAGEMENT

In our approach two different types of data are stored separately. A project file stores information about the set of process models, their relation to each other and additional data concerning the properties to be evaluated. We chose XML to file this kind of information, because XML

- supports the building of a hierarchy of process models,
- allows adding structured information to net instances,
- can easily be supported by other software and
- is (more or less) human readable.

Simulation of a process landscape generates mass data as a second type of data. We use a database management as best technical solution of storing these data fast and efficiently. For this purpose, we set up a database structure which is mostly independent of any intended evaluation purpose.
Both the project file's and the database's structure are discussed in the following two subsections. The third subsection explains how the information stored in the

database is transformed into an object model corresponding to the project file. By example an algorithm is presented operating on this object model for evaluation purposes.

XML-based Project File

Figure 2 depicts a typical project file. The root tag holds the project name and a hint where mass data is stored for simulation and evaluation, respectively. This information may be overridden by command line parameters or program settings (in this order) to support the exchange of project files between sites with different technical settings.

```
<project name=“DA“
  dbURL=“jdbc:mysql://localhost/test“
  driver=“org.gjt.mm.mysql.Driver“>

  <experiment name=“currentStatus“>
    <cluster name=“ProjectManagement“
        net=“PM.rnw“>
    </cluster>

    <cluster name=“ApplicationEngineering“
        net=“AE.rnw“>
        Some comment
    </cluster>

    <communicationSystem
        net=“../cbd/technical/PM_AE.rnw“
        costPerMinute=“42.0“
        capacity=“42000“>

        <channel
            initiator=“initiator_PM“
            from=“ProjectManagement“
            responder=“responder_AE“
            to=“ApplicationEngineering“>
        </channel>
    </communicationSystem>
  </experiment>
  <experiment name=“plannedSolution“>
      ....
  </experiment>
</project>
```

Figure 2: Sample project file

One project may consist of several experiments. Each experiment references to a set of Petri Nets, holds information about their linkage and may store additional data for some nets.
In general, Petri Nets are referenced by `cluster` tags. The mandatory attribute `net` gives reference to a file readable by Renew which holds the description of a Reference Net.

The `name` attribute may hold an unique id within an experiment. If there is no such id, the name is derived from the file name. A comment may be included in a `cluster` tag which makes the generated evaluation report much more readable. A second type of tag for referencing Petri Nets, also found in a project file, is more domain specific. With focus on the communication between processes the models handling the process of communication itself are of special interest. These models are Reference Nets, too. They are described by `communicationSystem` tags on the same level as `cluster` tags. These tags contain some additional data and may hold one or more `channel` tags. Each `channel` tag describes a linkage between any two `cluster` tags, referenced by their names. The linkage direction and the definition of the communication initiating and responding clusters also has to be specified.
In this subsection, we have shown how information about the mass data storage, the set of Petri Nets forming the process landscape to be simulated, structural information about net linkages and additional data are tied together in a project file. The next subsection describes the structure of the database and how additional data are stored therein.

Data Storage of Mass Simulation Data

We now discuss the database structure for logging simulation data by following the sequence of firing transitions in the example net shown by figure 3.

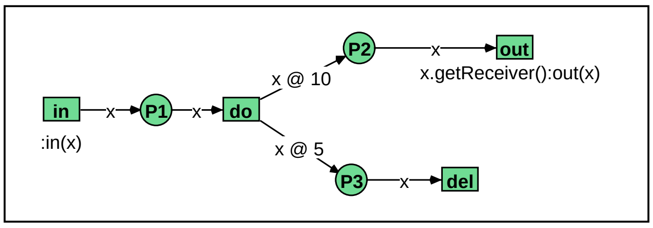


Figure 3: examplary net for illustrating database structure

The transition labeled **in** represents a part of a synchronous channel (here also named **in**). Synchronous channels are an extension of Reference Nets, where the firing of a starting transition also initiates the firing of a so-called downlink transition (Kummer 1998). The consumed token data can thereby be used by this downlink transition.
Table 1 shows the main database table, where the entries depict the process dynamic of the Petri Net in figure 3. The firing of transition **in** results in the action of putting a token on place **P1**. All putting or removing actions resulting from one transition’s firing have the same value of **transactionCount**. The firing results of transition **in** enable transition **do**, resulting in three further actions (see rows 2 – 4 in table 1). First the token placed on **P1** is consumed. Each token has a unique **tokenId**, valid only for a concrete marking situation of the net. The firing of transition **do** alters the net’s marking and produces two new tokens with ids “1” and “2”.

Table 1: Exemplary data for illustrating database structure

action	***transactionID***	***transactionCount***	***netID***	***placeName***	***tokenID***	***tokenData***	***clock***	***duration***
Putting	47110815	0	0	P1	1	Sender..	0	0
Removing	47110815	1	0	P1	1	Sender..	0	0
Putting	47110815	1	0	P3	1	Sender..	0	5
Putting	47110815	1	0	P2	2	Sender..	0	10
Removing	47110815	2	0	P3	1	Sender..	5	0
Removing	47110815	3	0	P2	2	Sender..	10	0

According to the arcs' inscription, the **tokenData** is copied from consumed to produced tokens. Using Reference Nets token data can be of any type of Java object. But the token placed on **P2** gets available after ten units of time (see last line of table 1), where the one on **P3** is available after five units of time respectively (see penultimate row in table 1). This is reflected by the data in the column entitled **clock**, which holds the value of a global clock when each action takes place (for simplicity we start with zero). All tokens necessary to enable this transition have to have arrived before the transition fires. Therefore, the time the transition fires, depends on the arrival time of the last token. To distinguish the time recorded in the column **clock** from each token's delay, another column called **duration** is introduced. It stores the token's delay assigned by the inscription of an outgoing arc of the producing transition.

After five units of time the token on **P3** is consumed and no other token is produced (see figure 3). So the removing action is the only one taking place while firing transition **del**. Five units of time later transition **out** fires. **out** is the uplink transition of the represented synchronous channel. The channel's corresponding downlink transition is dynamically set according to a token data's attribute, which evaluates from a call of the method **getReceiver** on the object represented by the token itself.

Two more columns of table 1 have to be explained: **netID** gives a unique id to each net instance. This is matched to the net's name and corresponding filename using a further table which is not discussed here. Another column is labeled **transactionID**. The value of this column remains unchanged throughout a complete simulation cycle and is matched to an experiment. This is needed to distinguish different simulation cycles of the same experiment.

The following subsection presents how information stored in the database and in the project file is combined for evaluation purposes.

Evaluation Component

The first step for evaluating simulation data is to bring together and synchronize both data streams. This is done by the evaluation component. It builds up a hierarchy of objects by processing the project file and then fills up this hierarchy by scanning the database. Afterwards, the resulting data model can be used as input for specific analysis algorithms.

We now discuss the structure of the model and its construction. Additionally, we explain how an example algorithm for the computing of communication costs deals with this basis.

As shown in figure 4, classes for a project, an experiment and a net form the upper three levels of the data structure. Building them up according to the project file is mainly straight forward, only the differentiation of nets representing clusters and nets representing communication systems is stored additionally in class **Net** by a boolean marker. Class **Run** represents a single simulation cycle. It is introduced on a separated level, because in some cases not all places of a net may be marked in a specific simulation cycle. Therefore, the set of places may vary between cycles, but stochastically analysis over all cycles has to consider all places.

Places and tokens are handled on the following two levels, parallel to the structure of action sequences and actions. Class **Action** references the same token data, whereas class **ActionSequence** stores the sequence of tokens' production or consumption in the order they arise during a simulation cycle. Compared with dealing only with place data, this allows an improved evaluation.

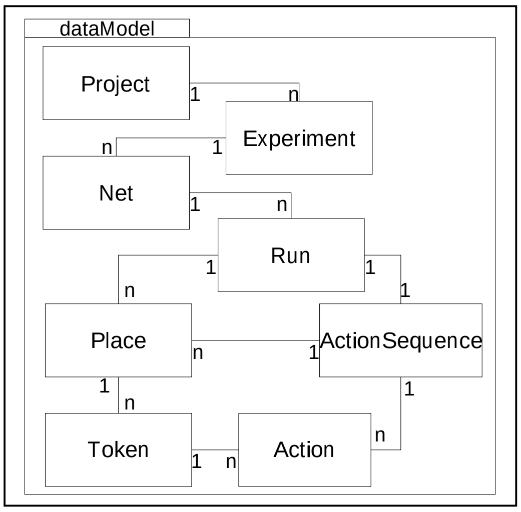
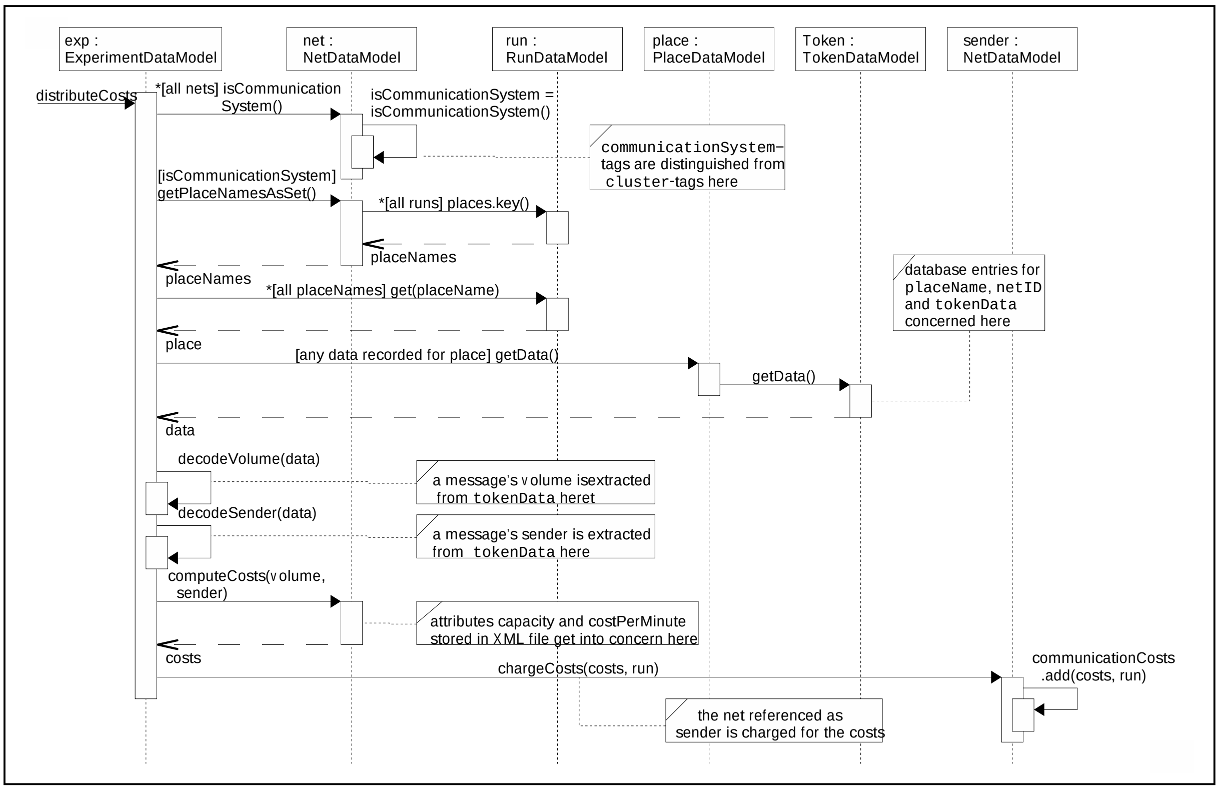


Figure 4: Data model for the analysis of simulation data

The described structure becomes filled with simulation data while scanning the database table introduced in the last section. There is only one instance from class **Project**, directly corresponding to the database. Experiments are mapped to values of **transactionID**. A value of zero for **transactionCount** indicates a new simulation cycle. Each putting or removing action is directly mapped to an **Action** object and added to the **ActionSequence** object. For each place a change in stock is recorded over time. Objects of class **Token** hold application specific token data serialized as a string. They reference to the places and nets they occur in. Additionally, the duration value assigned to the token objects is recorded.

Data discussed until now enables the analysis of communication aspects like the average path length taken of a specific token within a net, or the ratio of instantly enabled transitions. This is done in the sense that no token necessary for enabling a transition was available before firing took place with respect to the tokens' duration values. Algorithms for evaluating these communication issues are explained in (Störzel 2001) in more detail.

Figure 5 presents an UML-based sequence diagram for computing communication costs within a process landscape. This attribute is managed in the project file (see figure2). Initially, method **distributeCosts** of class **ExperimentDataModel** is called. For each net representing a communication system the set of all places for which a marking has been recorded in any run is computed. In an iteration over the elements of this set all communication costs are summed up separately for each run. This is done by extracting data **volume** and **sender** from the recorded token data. This information is combined with the parameterization of capacity and costs per minute of the considered net, stored in the project file. The net identified as sender is charged for the costs. Storing this data for each simulation run enables evaluation of stochastic data such as the minimum, maximum, average, standard deviation and the distribution itself. This information is output from the evaluation component and may be stored in XML format to support the idea of data exchange between different tools.


exp : ExperimentDataModel
net : NetDataModel
run : RunDataModel
place : PlaceDataModel
Token : TokenDataModel
sender : NetDataModel
distributeCosts
*[all nets] isCommunication System()
isCommunicationSystem = isCommunicationSystem()
communicationSystem-tags are distinguished from cluster-tags here
[isCommunicationSystem] getPlaceNamesAsSet()
*[all runs] places.key()
placeNames
placeNames
*[all placeNames] get(placeName)
database entries for placeName, netID and tokenData concerned here
place
[any data recorded for place] getData()
getData()
data
decodeVolume(data)
a message's volume isextracted from tokenData heret
decodeSender(data)
a message's sender is extracted from tokenData here
computeCosts(volume, sender)
attributes capacity and costPerMinute stored in XML file get into concern here
costs
chargeCosts(costs, run)
communicationCosts .add(costs, run)
the net referenced as sender is charged for the costs

Figures 5: Algorithm for computation of communication costs

RESULTS AND FUTURE RESEARCH

In this paper we presented our approach of storing mass simulation data separate from data concerning model structure and other model's properties to be evaluated. We have shown how the two data streams can be reintegrated into an appropriate data structure and how this may serve as a basis for evaluation algorithms. This approach has been validated to be successful and handy in different domains, such as interprocess communication analysis (Störzel 2001) and fuzzy timed specifications of multimedia presentations, where the latter is still in progress. We are confident of applying this approach successfully in broader areas of application.

Besides software support, we also want to extend our simulation and evaluation approach to additional process landscape features like the autonomy of different process locations (Gruhn and Wellen 2002).

Another point is that Renew also supports the feature of importing and exporting Reference Net descriptions in an XML format. This enables intregration of Renew and the extensions described here into a seamless chain of tools for integrated process planning and optimization support. A diploma thesis at the University of Dortmund (Brockmann 2002) has already implemented an integration of a document-based process planning tool (Palermo 2001) and the extended software system of this approach.

REFERENCES

Beckmann, M. 1997. "Simulationsverfahren für spezielle farbige stochastische Petri-Netze". Diplomarbeit at the Department of Computer Science, Technical University of Berlin, 1997, in German

Brockmann, C. 2002. "Werkzeuggestützte Modellierung von Prozesslandschaften", Diplomarbeit at the University of Dortmund, Department of Computer Science, Software Technology, April 2001, in German.

Donzelli, P. and G. Iazeolla. 2001. "A hybrid software process simulation model". *Software Process – Improvement and Practice*, Vol. 6, No. 2 (June), 97-110.

Gruhn, V. and U. Wellen. 2000. "Structuring Complex Software Processes by "Process Landscaping"". In *Proceedings of the 7th European Workshop on Software Process Technology, EWSPT 2000* (Kaprun, Austria, Feb). Reidar Conradi (ed.), 138-149, Springer Verlag, appeared as Lecture Notes in Computer Science No. 1780.

Gruhn, V. and U. Wellen. 2001. "Analyzing a Process Landscape by Simulation" *The Journal of Systems and Software* 59, 333-342.

Gruhn, V. and U. Wellen. 2002 "Autonomies in a Software Process Landscape", Internal Research Report No. 120, University of Dortmund, Department of Computer Science, Software Technology.

Jensen, K. 1997. *Coloured Petri Nets – Basic Concepts, Analysis Methods and Practical Use*, Volume 1, second edition, Springer Verlag.

Kummer, O. 1998. “Simulating synchronous channels and net instances”. In *5th Workshop Algorithmen und Werkzeuge für Petri-Netze* J. Desel; P. Kemper and E. Oberweis (eds.). Research Report No. 664, University of Dortmund, Department of Computer Science, Oct. 1998.

Kummer, O. and F. Wienberg. 2000. “Renew - The Reference Net Workshop”. *Petri Net Newsletter* No. 56, 12-16.

Law, A.M. and W.D. Kelton. 1991 *Simulation, Modeling & Analysis*, second edition, McGraw-Hill.

Lautenschlager, M. 1995. “Data Handling in the Climate Model Archive at DKRZ”. In *Proceedings of the 9th International Symposium on Computer Science for Environmental Protection, Space and Time in Environmental Information Systems*, H. Kremers and W. Pillmann (eds.), 287-294, Metropolis Verlag, Marburg, Germany.

Merkuryeva, Y., J. Tolujew Y., E. Blümel, L. Novitsky, E. Ginters, E. Viktorova, G. Merkuryev and J. Pronis. 1998. “A Modelling and Simulation Methodology for Managing the Riga Harbour Container Terminal”. *Simulation* 71, No. 2 (Aug), 84-95

Merkuryeva, G., Y. Merkuryev and J. Tolujew. 2000 “Computer Simulation and Metamodelling of Logistics Processes at a Container Terminal”. Published at http://www.ici.ro/ici/revista/sic2000_1/art06.html (2002)

Project Group Palermo. 2001. "Endbericht der Projektgruppe Palermo" Research Report No. 109, University of Dortmund, Department of Computer Science, Software Technology, March 2001, in German.

Störzel, M. 2001. “Simulation verteilter Prozesslandschaften”, Diplomarbeit at the University of Dortmund, Department of Computer Science, Software Technology, October 2001, in German.