*Proceedings of the 2009 Winter Simulation Conference*
*M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin and R. G. Ingalls, eds.*

# BUILDING INSIGHTFUL SIMULATION MODELS USING FORMAL APPROACHES – A CASE STUDY ON PETRI NETS

Durk-Jouke van der Zee

Faculty of Economics & Business
University of Groningen
Groningen, The Netherlands

## ABSTRACT

In recent years development of formal approaches for modeling and simulation of manufacturing systems received significant attention. Approaches building on alternative Petri Nets formalisms show essential strengths in accurately capturing both a system's static structure and its dynamics, availability of mathematical analysis methods, and graphical representation. However, models of realistic systems are often perceived as too large and complex to understand by project stakeholders. This hinders their participation in modeling, and solution finding, and may influence their perception of model credibility. In this article we address this issue by considering a structured approach for embodying high-level manufacturing concepts. The approach aims at creating more insightful simulation models by building on sound and explicit conceptualization, i.e., the choice of manufacturing concepts, and clear rules for their formalization, i.e., their mapping on elementary model components. We adopted the Petri Nets based tool ExSpect$^{TM}$ to illustrate and evaluate our approach.

## 1 INTRODUCTION

Formal approaches building on alternative Petri Nets formalisms have proved to be successful tools for modeling and simulation of manufacturing systems (DiCesare et al. 1993, Zeigler 1990, Jensen 1997). Their basic strengths in accurately describing system's static structure and its dynamics, availability of mathematical analysis methods, and graphical nature underpin this success. However, many authors suggest that there is an even greater potential for their practical use, if they would allow for a more active stakeholder understanding and participation. So far, this is often hindered by model size and complexity (Kamper 1991, Barretto et al. 1999, Barber et al. 2003, Ryan and Heavey 2006).

Basically, stakeholders' understanding of simulation model elements and their workings starts from their recognition of high level manufacturing concepts, like, for example, machines, buffers, planners, goods etc. Choice of concepts and their coded and visual representation follows from the analyst's creativity. His creativity is bounded and guided by implicit or explicit guidelines, that is, good modeling practices and principles (Pidd 1999), domain-related insights (Valentin and Verbraeck 2005) and the choice of elementary model components (Kreutzer 1986).

Typically, formal approaches start from a few low-level components for defining models. Hence, guaranteeing stakeholder understanding puts high demands on analyst's skills in sound conceptualization and structured formalization. Conceptualization refers to the choice of entities, their activities, and relationships considered characteristic for a domain, whereas formalization concerns the mapping of the respective system elements on model components. Here we will address guidance for the analyst in both activities, assuming the use of formal approaches for manufacturing simulation.

Robinson (2008) distinguishes between three basic approaches in guiding the analyst in conceptual modeling: principles of modeling, methods of simplification, and modeling frameworks. Principles of modeling advocate an evolutionary development of models – start small and simple, and adapt and extend the model incrementally. Methods of simplification work the other way around by suggesting ways for model pruning. While both approaches offer relevant assistance for conceptual modeling, they do not a-priori address the creation of the conceptual model, i.e., the identification of elementary model components appealing to a domain. Modeling frameworks distinguish themselves from the aforementioned approaches by specifying a procedure for detailing a model in terms of its elements, their attributes and their relationships.

In this article we extend our previous work on a modeling framework for manufacturing simulation by linking it with the use of ExSpect$^{TM}$, a Petri Nets based tool (ExSpect$^{TM}$ 1999). In doing so we aim to illustrate and evaluate how the notion of a well-defined domain related modeling framework, and rules for mapping its outcomes on model components, may make a difference for model understanding.

The remainder of the article is organized as follows. In Section 2 we shortly discuss our modeling framework for manufacturing simulation and its underpinnings, i.e., engineering principles. Next, we motivate our choice for ExSpect$^{TM}$, discuss its logic, and specify rules for mapping elements of the modeling framework on ExSpect$^{TM}$ basic components (Section 3). In Sections 4 and 5 we illustrate and evaluate the use of the modeling framework, and the mapping rules by a case example. Finally, in Section 6, we summarize main conclusions.

## 2 CONCEPTUALIZATION - A MODELING FRAMEWORK FOR MANUFACTURING SIMULATION

In this section we discuss a modeling framework for manufacturing simulation. First we categorize engineering principles underpinning the framework. Next, we consider its core, i.e., a reference architecture for manufacturing systems. Finally, we introduce a method for its use. For more details, see Van der Zee and Van der Vorst (2005), Van der Zee et al. (2007).

### 2.1 Engineering Principles

Decomposition and classification are cognitive approaches suggested as being fundamental in the process of identifying the phenomena to be modeled (Burton-Jones and Meso 2006). Basically, decomposition starts from a top-down approach towards modeling, while classification works the other way around. Essentially, classification boils down to an abstraction of similarities among individual things for supporting cognitive economy, inference, and communication, see, for example, Smith and Medin (1981).

Let us now consider the engineering principles underlying the modeling framework for manufacturing simulation, see Table 1. Principles I-III are rather well-known. They follow from the notion of a system boundary (I), basic system logic (II), and queueing systems (III). According to Lefrancois and Montreuil (1994) a distinction between intelligent and non-intelligent entities (IV) permits a more natural and richer presentation and implementation of systems modeled. In such a context, intelligent beings are modeled as agents. Agents are used to implement decision rules inherent to manufacturing system planning and control. Examples include routines for scheduling, dispatching and releasing jobs for a machine or department. Principles I – IV tend to be valid for a much wider category than just manufacturing systems.

Table 1: Decomposition principles underlying the modeling framework for manufacturing simulation (Van der Zee 2007)

| | Decomposition principles |
|---|---|
| I | External and internal entities |
| II | Movable and non-movable entities |
| III | Queues and servers |
| IV | Intelligent and non-intelligent entities |
| V | Infrastructure, flows and jobs |
| VI | Modality: physical, information and control elements |
| VII | Dynamics: executing jobs |

Principles V-VII are somewhat more specific for the manufacturing field. Manufacturing systems are built up of infrastructural elements like workstations, information systems and managers. Flows refer to the objects being exchanged and transformed by this infrastructure as a net effect of jobs being executed (V). The separation of physical, information and control elements (VI) is assumed to facilitate a higher degree of model reusability and a more "natural" model building environment (Mize et al. 1992). Van der Zee and Van der Vorst (2005) suggest that all activities in the manufacturing system, including decision making, should have a common denominator: the job (VII). The job concept is meant to bring two important advantages. Firstly, the use of this common denominator for all activities will provide a clear and natural mechanism for event scheduling, where events are related to the start and the completion of jobs. Secondly, an explicit notion and allocation of company activities (compare V), increases visibility and traceability of decision variables.

### 2.2 Reference Architecture for Manufacturing Systems

In this section we discuss a reference architecture for manufacturing simulation. We will relate our choice of model components to the framing, i.e., combining, of decomposition principles - as identified by their Roman numbering. We adopt an object oriented notation, cf. Booch (1994).

### 2.2.1 Classes and Hierarchies

The reference architecture distinguishes between three main object classes: *agents*, *flow items* and *jobs* (V), see Figure 1. *Agents* represent the infrastructural, *non-movable*, elements of a manufacturing system such as workstations, information systems and managers (II). They are assumed to be intelligent to a certain extent (IV). Their decision-making capabilities relate to transformations of goods or data. System boundaries are reflected by the notion of *internal* and *external* agents (I). For internal agents such as machines, warehouses, AGV systems, planners etc. it is distinguished between processors and storages, by considering the nature of their dominant type of resources (III). For external agents, the framework separates suppliers from customers (III).

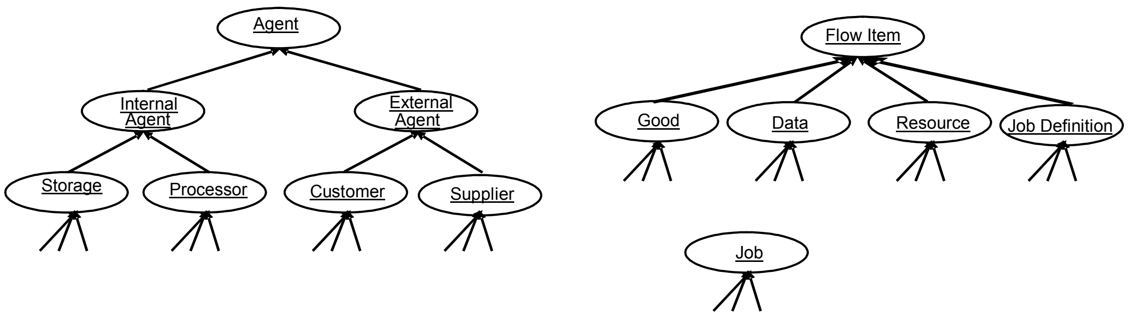

Figure 1: Reference architecture for manufacturing simulation – main object classes

Four types of movable (flow) items (II) are included in the modeling framework: goods (like, for example, materials, parts, semi-finished products), resources (like, for example, workers, tools, vehicles), data (like, for example, monitoring data, demand figures) and job definitions (VI). Job definitions model the flow of control, i.e., messages steering the movement of goods, resources, and data. As such they initiate and inform agent activities, by carrying relevant information related to these activities, such as, for example, their input, processing conditions and the agents to whom the resulting output should be sent.

In a manufacturing system agents and flows are linked by jobs, i.e., business activities (V). Moreover, we assume that each such activity relates to a job. Typically, job execution links flow items and agent' resources.

### 2.2.2 Class Definitions - Agents

The structure for agents follows from their role as intelligent entities (IV). The definition of a structure for an internal agent was inspired by the atomic model as defined by Zeigler (1990), see Figure 2a.

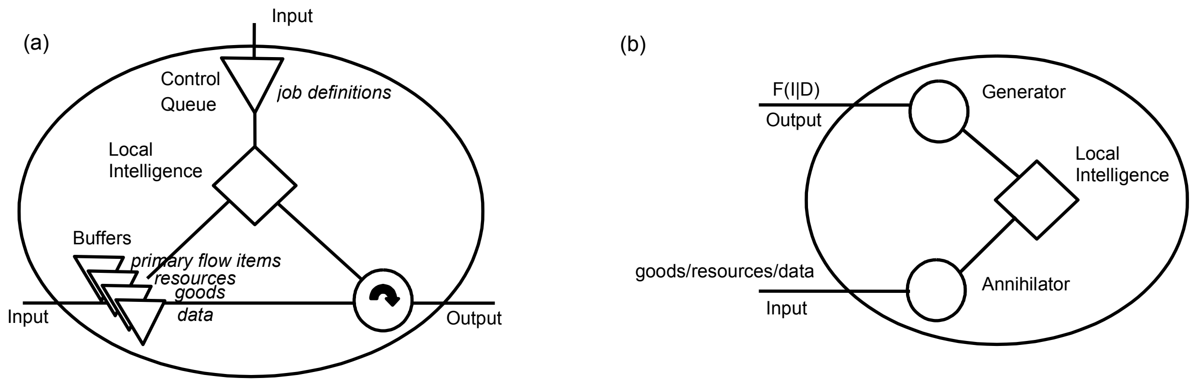

Figure 2: Agent definitions – internal agent (a); external agent (b)

The state of an agent relates to its attributes and their values. Attributes concern buffers and transformers (III). Buffers model the temporary storage of those flow items which are the prime subject of a future job or which have a facilitative role in job execution (resources, information). Note how job definitions for an agent are stored in the control queue. The transformer reflects a set of jobs in execution and contains the flow items that are related to these jobs.

Above basic elements of an internal agent are discussed. For agent functions we will distinguish between input and output operations, and the local intelligence (IV). The initiation of a job is enabled by rules comprised in the local intelligence (IV). As a first rule in initiating a job, the job definition with the highest priority in the control queue is investigated. The respective job may start if demands on its inputs, capacity, and processing conditions are met.

The notion of local intelligence applies to all agents, including work stations and planners. Where intelligence for work stations may be restricted to elementary rules for timing and release of jobs, decision logic for controllers may be comprehensive. Decision logic for controllers may refer to a wide range of rules that support, for example, capacity planning, material planning, scheduling and dispatching.

Besides the element local intelligence (IV), which is also found for internal agents, generators and annihilators are distinguished for external agents, see Figure 3b (customer). Generators represent "sources" of flow items, while annihilators model "sinks" in which flow items disappear (I). Local intelligence may be used to link generators and annihilators.

### 2.2.3 Relationships between Agents

Agents communicate with other agents by exchanging alternative types of flow items, being the net result of job execution (V, VI). In this section we will consider two specializations of the basic type of relationship in somewhat more detail (Figure 3a,b):

- The relationship between an internal agent and its controller.
- Relationships between external and internal agents.

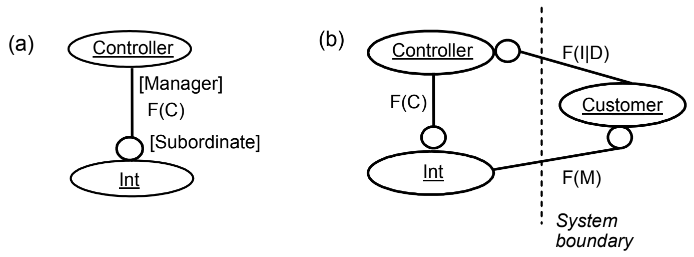

Figure 3: Agent relationships – control (a), and customers (b)

Control (Figure 3a) is assumed to be effectuated by the sending of job definitions from a controller object to an internal agent, denoted as Int (VI). Each agent (subordinate) refers to exactly one controller (manager) from which it receives its job definitions, denoted as F(C). Reversely, a subordinate can send information about its status to its controller. Mechanisms like hierarchical control and coordinated control are embedded in this class structure, by allowing higher level controllers to steer activities of lower level controllers.

For external agents we distinguish between customers and suppliers (I). Typically, a customer issues an order by sending a demand signal (F(I|D)) to an internal agent of the type controller (Figure 3b). The controller in its turn specifies a job definition (F(C)) for an internal agent (Int) who is responsible for the deliverance of the requested items (F(M)), where M refers to the modality. In the case of a supplier roles have changed: the controller sends an order to a supplier, who has to take care of delivery of the requested items.

### 2.2.4 Dynamics Structure – Agents Executing Jobs

In line with our job-oriented view we assume the execution of jobs by agents as the driving force of business dynamics (VII). Job execution is related to a procedural three-phase description (Pidd 1998). In the A Phase it is determined which job is completed next. Once this job has been found, time is advanced to the corresponding moment in time. Subsequently, the job is "completed" (in the B-Phase), i.e., the resulting output is sent from the agent that carries out the job to other agents. In the C-Phase it is tested if the flow items that are received by these agents enable the initiation of new jobs (conditional activities). The three phases are repeatedly run through until the (simulation) time is up.

## 2.3 Method for Use

Elementary steps for application of the modeling framework concern (1) the determination of a system boundary, (2) the definition of entities, i.e., agents, flow items, and jobs following a top-down refinement process, and (3) detailing entities "bottom up". Details on the respective steps may be found in Van der Zee (2007).

## 3 FORMALIZATION: EXSPECT$^{TM}$ – LOGIC AND MAPPING RULES

In this section we discuss essential characteristics of ExSpect$^{TM}$, and indicate their support for embodying high-level manufacturing concepts. Next, we specify rules for mapping high-level manufacturing concepts on ExSpect$^{TM}$ logic.

### 3.1 EXSPECT$^{TM}$ – An Informal Introduction

In past decades much research has been done on developing Petri Nets formalisms, and its application for various fields of interest (Jensen 2009). For a first introduction on Petri Nets, see, for example, Murata (1989), David and Alla (1994), and Jensen (1997). Essential strengths of Petri Nets are in their descriptive power, allowing to capture both static and dynamic characteristics for a system, availability of mathematical methods, next to simulation, and their graphical representation. Here we address use of Petri Nets models for simulation modeling and analysis.

Starting point for all research on Petri Nets is the seminal work by Petri (1962), who defined the classic Petri Nets, cf. Figure 4. Classic Petri Nets represent systems by means of a *net structure*, i.e. a bipartite directed graph, for specifying the static part of the system, and a *marking* which reflects the state of the system. The net structure consists of two kinds of nodes: *places* and *transitions*. Places and transitions are connected by means of *directed arcs*. Arcs are only allowed to connect a place with a transition, or a transition with a place. The first type of place is regarded as an input place for a transition, while the latter is regarded as an output place. The marking or state of a net corresponds to the assignment of one or more *tokens* to each place.

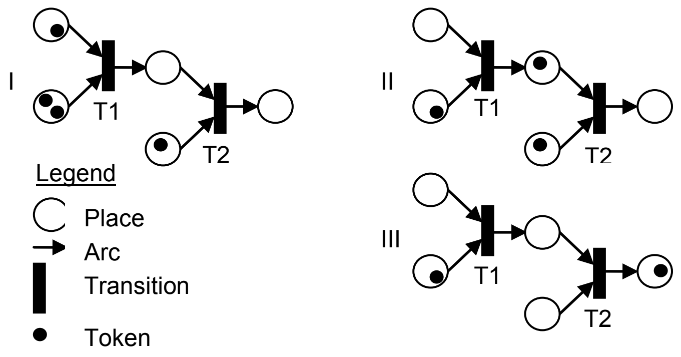

Figure 4: Classic Petri Nets – Notation and case example

State changes are modeled by Petri Nets through the *firing* of a transition. Firing of a transition boils down to the consumption of a specified number of tokens from all its input places and the production of tokens for all of its output places. A transition is only allowed to fire (enabled) if there are sufficient tokens available in all of its respective input places. The firing of a transition is considered indivisible or atomic. An illustration of the firing mechanism and the associated state changes are displayed in Figure 4 (I, II, III). Note how firing transition T1, allows transition T2 to fire next. The net structure in Figure 4 may be used to model simple processes, like secretaries' duties in putting a letter in an envelope (T1), and putting a stamp on the envelope (T2), or a simple assembly operation involving two machines (T1, T2).

ExSpect$^{TM}$ extends classic Petri Nets through the notions of *color*, *time*, and *hierarchy* (Van der Aalst 1992, 1994). Let us shortly consider the respective extensions, and imply their relevance for our modeling framework. Note how our choice for ExSpect$^{TM}$ as a research vehicle is motivated by the way it facilitates an easy mapping of high-level manufacturing concepts - next to the presence of an established tool.

Whereas classic Petri Nets do not allow to further detail tokens, ExSpect$^{TM}$ allows for token attributes, or "colored" tokens. An important advantage of *colored* tokens is the possibility to reduce model size, by adding detail to tokens instead of creating more elaborate net structures – as for classic Petri Nets. Furthermore, from a manufacturing perspective token attributes assist in more natural modeling, as they allow modeling flow items like goods, data, etc. in realistic way, cf. Section 2.2.

ExSpect$^{TM}$ supports model structuring by allowing for *hierarchy*, i.e., representing systems as an aggregate of subsystems. Therefore a construct, called "system" is introduced. A system is an aggregate of places, transitions, and (possibly) subsystems. The notion of systems helps in creating model overview for both analyst and stakeholders. Within the context of manufacturing systems are helpful to identify and represent high-level concepts like machines, planners, buffers, etc. Recall, how they are addressed in our modeling framework as agents.

Logistic performance analysis for manufacturing systems implies the notion of *time*. Classic Petri Nets are not capable of handling quantitative time. ExSpect$^{TM}$ allows for the possibility of stochastic delays, which are associated with time stamps linked to tokens. Time stamps specify the moment the token becomes available. Linking model dynamics with tokens allows for a straightforward representations of jobs, as a (composite of multiple) flow items(s) being transformed until a pre-specified completion moment.

Figure 5 shows a small example net structure for ExSpect$^{TM}$. Note how ExSpect$^{TM}$ addresses transitions as processors, and places as channels. Token colors are mentioned in brackets. Time stamps are indicated by dashed lines, specifying the moment (t) a token becomes available. For example, the token representing a pair of green shoes, will be available at t=1. Firing of processor T1 is enabled at t=1, given the presence of both shoes and shoelaces. Next firing of processor T2 is possible if both a box are available, and the combined shoes and shoelaces. Note how the time stamp of the latter entity follows from the execution of processor T1. The system concept (dashed rectangle) may be used to represent the "shoe shop", instead of single operations. In turn this system may be used as a building block in a larger manufacturing system.

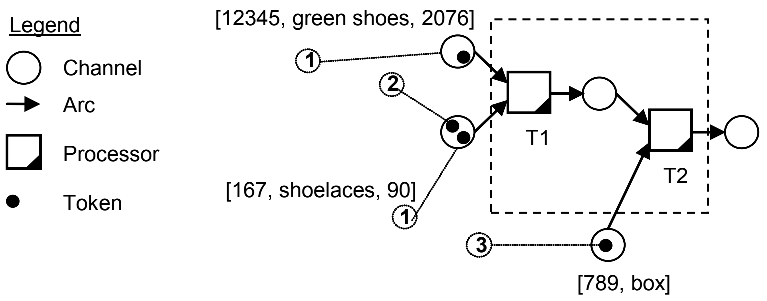

Figure 5: ExSpect$^{TM}$– Case example

## 3.2 ExSpect$^{TM}$ - Language

In this section we consider basic language constructs for ExSpect$^{TM}$, and link them to the basic elements of the underlying Petri Nets formalism, see Section 3.1. For more details and backgrounds see Van der Aalst (1992, 1994) and ExSpect$^{TM}$ (1999). ExSpect$^{TM}$ distinguishes between four kinds of definitions: *type definitions, function definitions, processor definitions, and system definitions*. Type definitions are used to specify token values. Each place has a type, which determines the values for the tokens it may contain. The type system of ExSpect$^{TM}$ considers primitive types, like, booleans, numerals, reals, and strings, and type constructors, allowing to specify lists, sets, records etc. For example, a car may be specified as:

```
car: [id: str, color: str, year: num]
```

This *type definition* may be used to specify the car's license plates, its color, and the year it was built. It builds on a type constructor, i.e., record, denoted as "[]", and two primitive types, i.e., num and str, referring to numerical and string values.

*Functions* answer to the need to specify values for tokens that are produced as a net effect of firing processors. Many well-known functions are available for ExSpect$^{TM}$ as a standard feature, concerning set-theoretical, logical, and numerical constants and functions. For example, the following function may be used to compute the surface of a rectangle:

```
surface [x: real, y: real]
:=
x*y
: real;
```

*Processor definitions* are used to describe transitions. Definitions consist of a header and a contents part. The header part specifies the processor's name, interaction structure, and parameters. The interaction list details input channels, output channels and stores. Note how stores refer to a special kind of place, which always contains exactly one token. The contents part consists of concurrent assignments of expressions to output channels and stores. Expressions may include the use of functions. For example, a machine operation may be modeled as:

```
proc machine [in   start: part,
              out  finish: part,
              val  t : real]
:=
finish <- operation(start) delay t;
```

Note how the duration of the operation (a function) is represented by adapting the time stamp for the respective part (token) by including a delay of duration t.

Similar to the processor definition a *system definition* consists of a header and a contents part. The header details value, function, processor, and system parameters, and interaction structure, see processor definition. The contents part refers to a listing of all its elements, i.e., channels, stores, processors, and channels. For example, consider a system which consists of a gas station and car wash:

```
sys car_service [in x:car, out y:car, t:real]
:=
channel cleaned_cars: init [id: 1, color: purple, year: 1990],
carwash (in x, out cleaned_cars),
gas station (in cleaned_cars, out y, val t);
```

Gas station and car wash are represented by two processors. Processors are linked through a common channel cleaned cars. The respective channel is initialized by defining one token of type car.

## 3.3 Guidelines for Mapping High-level Manufacturing Concepts on Basic Model Components of ExSpect$^{TM}$.

### 3.3.1 Object Classes - Flow Items, Agents and Jobs

Our reference architecture for manufacturing systems distinguishes between three main object classes, i.e., agents, flow items, and jobs, see Section 2.2. In Section 3.1 we linked the respective classes to ExSpect$^{TM}$ in an informal way. We found how *flow items* can be represented naturally by *tokens*. *Type definitions* allow to specify differences among flow items in a straightforward way, by associating them with token attributes.

We propose to specify *internal agents* as *systems*, according to the following generic format:

```
(* INTERFACE *)
sys internal_agent [in X, out Y, con Z, val id: A ... ]
:=
(* BUFFERS *)
store Input_Buffer     : B init ...
store Control_Queue    : C init ...
...
(* TRANSFORMERS *)
channel Transformer    : D init ...
...
(* INPUT OPERATIONS *)
input (in x, out Input Buffer| pre x.receiver = id)
inputc (in x, out Control_Queue | pre x.receiver = id)
...
(* LOCAL INTELLIGENCE FOR EXECUTION *)
local_intelligence (in Input_Buffer, Control_Queue, <BUFFERS>, out <TRANSFORMERS>
                    | pre check(Control_Queue, Input_Buffer, <BUFFERS>))
 (* OUTPUT OPERATIONS *)
output (in <TRANSFORMER>, out y)
...;
```

To support our discussion on the internal agent's definition we graphically depict a typical agent structure as a Petri Nets model, see Figure 6. The header of the system definition specifies input and output channels. They reflect the inflow and outflow of alternative types of flow items. We set no a-priori restrictions to their specification, except for the presence of an input channel linking the agent with its controller (cf. "con Z" in the header), also see Section 3.3.2. The body of the system definition models the agents' internal structure in a generic way. This format is meant to characterize a skeleton architecture, which specifies default elements of agent´s structure and their relationships. The architecture may be enriched and extended for practical use.

The skeleton architecture includes an input buffer (Input_Buffer), control queue (Control_Queue), transformer (Transformer), and operations, i.e., input, output, and local intelligence. The respective operations link input channels ("X" in system header) with Input_Buffer and Control_Queue (input, inputc), Transformer and output channels (output, cf. "Y" in system header), and Control Queue, Input_Buffer and Transformer (local_intelligence). Following ExSpect$^{TM}$ logic operations are modeled by processors, whereas buffers, and transformers are represented by stores and channels.

Practical use of the agent's system definition assumes the detailing of type definitions, processors and functions. Furthermore the above skeleton architecture allows for the inclusion of additional (facilitative) buffers (<BUFFERS>), and transformers (<TRANSFORMERS>) being dedicated to specific flow items. This may enhance model transparency. Note how the system definition of an external agent follows from pruning the system definition of the internal agent.

*Jobs* are associated with the activities described by the processor local_intelligence. Their dynamics is reflected by (sub)sets of flow items (tokens) contained in transformers. Respective sets refer to inputs required for a specific job, for example, goods, tools etc., as reflected by alternative types of tokens. Jobs are initiated as net result of executing (firing) of the local intelligence, see 3.3.3.

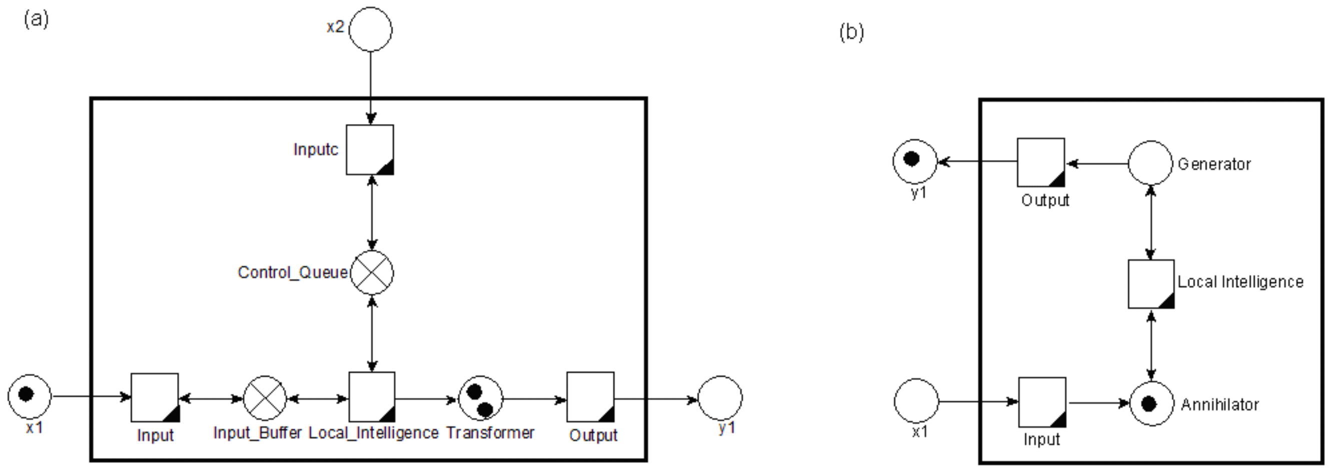

Figure 6: Petri Net models of agents – internal agent (a), and external agent (b)

### 3.3.2 Relationships between Agents

Agent relationships are modeled by linking them through a common channel, which connects their respective output and input channels. Control relationships are modeled by connecting each agent to a single controller, i.e., supervising agent, cf. Section 2.2.

### 3.3.3 Dynamics Structure – Agents Executing Jobs

In our modeling framework we relate dynamics to job execution. Job initiation is associated with the execution of the local intelligence. This boils down to the firing of the respective processor. Preconditions for firing are the availability of job definitions and its associated job inputs, see 3.3.1. Firing implies the setting of time stamps for tokens modeling flow items - being subject to a job. In turn, time stamps set the bound events of job completion.

## 4 CASE EXAMPLE – SUPPLY OF REFINED OILS

In this section we present a case example to illustrate guidance of the modeling framework, and mapping rules in creating more insightful Petri Net simulation models. The case example concerns the supply of refined oils for producing margarine. It is motivated by a project in industry (Schippers 1995), concerning a large manufacturer of margarine. The manufacturer decided to concentrate its refinery facilities for producing oils and fats at a single location. This decision forced a redesign of the supply system, involving the transportation of truckloads of oils and fats from the new refinery to the storage tanks at the margarine production facilities. Here we will consider the supply system for a single production facility.

### 4.1 System Description

The supply system is made up of transportation facilities, i.e., trucks, unloading stations, and a number of storage tanks. The refinery is located at about an half hour driving distance from the production facility. System complexity follows from restrictions on the use of resources, and the need to tightly tune supply operations, in order to (1) guarantee a high service level at the production lines, (2) avoid overstocks, i.e., truck loads which cannot be unloaded because storage tanks are not available, (3) reduce waiting costs at the unloading stations. Redesign of the supply system involves both its infrastructure, and its control.

## 4.2 Conceptualization

As a first step in modeling we considered the conceptual model of the supply system. Scope and level of detail are captured in our choice of agents and their attributes. An example of an agent definition is given in Figure 7. It represents the queue of trucks at the unloading stations. Jobs concern the moving of trucks towards unloading stations, as they follow from a decision of the queue manager. Operator data support local decision making by capturing agent status, and data on related agents. Note how the information on truck arrivals (cf. the Output operation) triggers decision making for the queue manager.

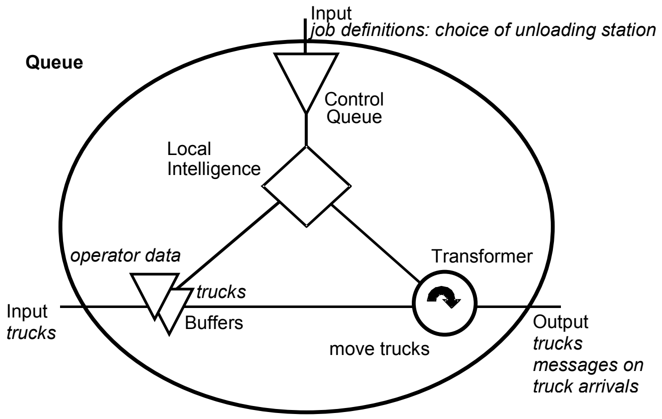

Figure 7: Class definition for agent Queue

## 4.3 Formalization

Figure 8 shows the Petri Nets simulation model we built starting from the conceptual model, and mapping rules as we proposed them in Section 3.3. The figure represents agents responsible for the physical flows of goods and resources (oil, trucks), i.e., OilRefinery, Queue, UnloadingStation1, UnloadingStation1.2, Tanks, and MargarineProduction, and their planning and control, i.e., SalesDepartment, PlanningOilDeliveries, QueueManager, LoadManager, and TankController. Here the sales department and MargarineProduction may be considered external agents. Note how adequate planning and control of the supply system assumes a detailed tuning of activities, as represented by the flows of data. For example, new deliveries of oil from the storage tanks to the production lines are only allowed at the moment production lines complete a job. The manager of the storage tanks (TankController) is informed on the respective job completions by messages (TankData).

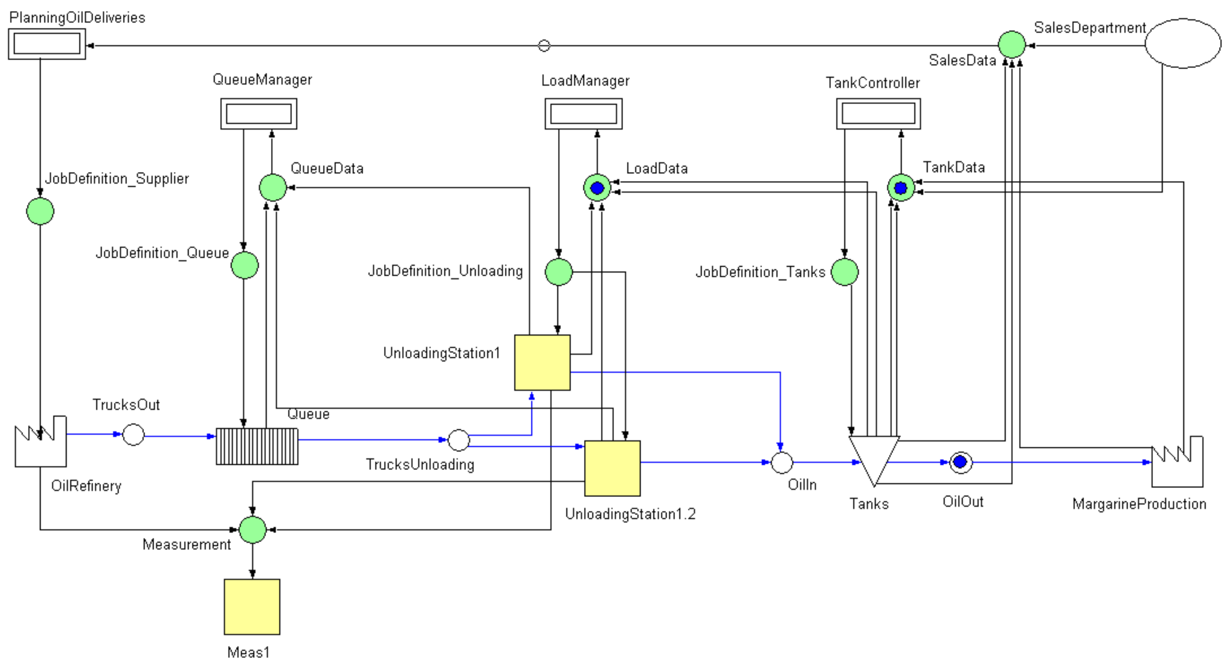

Figure 8: Petri Nets simulation model of the supply system

Each agent is defined according to the mapping rules proposed in Section 3.3. An example of a Petri Nets model of an agent (UnloadingStation1) is given in Figure 9. In line with Sections 2 and 3 the model distinguishes between input and output operations (Input_U, Inputc_U, Output_U), local intelligence (LocalIntelligence_U), buffers (InputBuffer, OperatorData), and transformers (Transformer). They are modeled following ExSpect$^{TM}$ logic by processors, stores, and channels respectively. Note how Signalout_meas facilitates performance measurements.

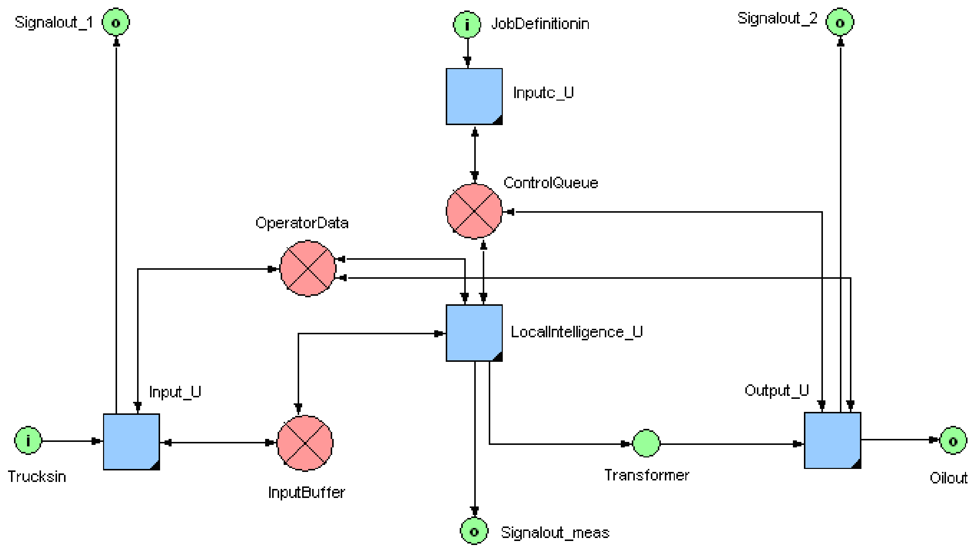

Figure 9: Petri Nets model for an agent (UnloadingStation1)

## 5 DISCUSSION – EVALUATION OF THE APPROACH

### 5.1 Scope

The modeling approach as discussed in this article starts from two assumptions: the use of a specific modeling framework for manufacturing simulation, and the choice for a specific, Petri Nets based tool. Direct implications are: no full coverage of the manufacturing field, and mapping rules which are dependent on the choice of tool and its fit with the modeling framework. For example, a view on manufacturing which starts from flow items, which display autonomous behavior in acquiring passive resources, may be captured less easy by the modeling framework. In this respect one may think of large scale transportation systems, or team-operated manufacturing cells. In principle, such 'misfits' may be repaired by extending or adapting the modeling framework.

ExSpect$^{TM}$ supports implementation of concepts from the modeling framework by its extensions relative to classic Petri Nets, i.e., colors, hierarchy and time. Their relevance follows from requirements on model size and realistic modeling, model overview, and field of application (manufacturing logistics) respectively. More specific support is related to the timing mechanism for ExSpect$^{TM}$, which relates time to tokens by means of time stamps. This facilitates an easy mapping of the job concept on ExSpect$^{TM}$ logic in terms of a composition of flow items, represented as tokens, being subject to time related transformations. Our choice for ExSpect$^{TM}$ does not imply, however, that other formal approaches could not be related to our modeling framework. Next to alternative Petri Nets based approaches one may think of DEVS (Zeigler 1990). Among others, it links very well with our agent concept, which was inspired by DEVS conception of an atomic model.

Next to simulation, ExSpect$^{TM}$ also supports mathematical methods for model analysis. See Van der Aalst (1992) for examples of methods and their applications. Their integration in our approach for more insightful simulation modeling is considered both a relevant and interesting issue for future research.

### 5.2 Guidance

Essential guidance from our modeling framework follows from its explicit view on manufacturing systems. Instead of having to create – an implicit – view on a manufacturing system of his own the analyst could start from a clear point of reference. Relevance of this explicit notion of elementary manufacturing concepts seems greater if formal approaches are used for model specification and coding – as they typically start modeling from low-level concepts. Further support of our approach is in a set of mapping rules, linking manufacturing concepts to ExSpect$^{TM}$ logic.

It is interesting to consider linking our modeling framework with the use of commercial software packages, such as, for example ARENA$^{TM}$, Em-Plant$^{TM}$, Enterprise Dynamics$^{TM}$, as an alternative for the approach proposed in this article. Whereas the use of commercial software packages may imply a "misfit" between the high-level concepts underlying their library, and the concepts in the modeling framework, low-level concepts as in Petri Nets may allow for a more tailored and founded (cf. engineering principles, see Section 2.1) representation of manufacturing concepts. Hence guidance of our approach may be more straightforward in the latter case.

### 5.3 Understanding

Corner stones for building understanding among stakeholders are model transparency and model completeness. Model transparency should result from the notion of a limited set of elementary manufacturing concepts offered by the modeling framework. This set should assist in building model structures that appeal to the imagination of all parties involved in the study. On the other hand, model completeness is related to the explicitation of relevant manufacturing objects and their workings. Essential support of our modeling framework in this respect may be related to the notion of engineering principles underlying the modeling framework, especially IV-VII, see Table 1. Adherence to these principles seems to be more relevant in case of formal modeling approaches. Interestingly, the opportunity and potential in doing so seems greater as in commercial simulation packages, which may be caught in their own logic, cf. Section 5.2.

### 5.4 Further Support

We consider the set of mapping rules for guiding the specification of manufacturing concepts as a first step in supporting the analyst. This leaves the possibility to further detail rules by, for example, distinguishing between alternative agent classes, or – beyond that – creating more permanent support by designing comprehensive libraries of building blocks. As clarified above, see Section 5.2, 5.3, we see great relevance and opportunity in doing so. Essential contributions from the modeling framework build on its underlying logic, i.e., engineering principles, whereas formal approaches offer the descriptive power. In our view such manufacturing logic may avoid some of the pragmatics often ruling model construction, and create more insight.

## 6 CONCLUDING REMARKS

In this article we addressed the practical use of formal approaches for simulation modeling. More in particular, we consider the use of Petri Nets. Petri Nets are known for their qualities in accurately modeling and representing real systems. However, Petri Net models for realistic cases are often found too large and complex to understand by non-experts. In this paper we propose a structured approach for improving model insight among stakeholders. More in particular, we consider the use of the Petri Nets based language called ExSpect$^{TM}$ for manufacturing simulation.

Key elements of the proposed approach are a modeling framework for manufacturing simulation, and a set of rules for linking it to the use of ExSpect$^{TM}$. The modeling framework supplies an explicit reference architecture for manufacturing systems and a method for its application. The architecture adopts a set of engineering principles, in defining manufacturing elements in terms of agent classes, and their interactions. This allows the analyst to start from a clear and appealing point of reference, instead of having to create a view of his own. In this way the perceived "gap" between the high-level manufacturing concepts and the low-levelness of Petri Nets components may be bridged. Mapping rules support linking manufacturing concepts to ExSpect$^{TM}$ logic in a transparent way.

Relating the use of the modeling framework to ExSpect$^{TM}$ was found rather easy and straightforward. Petri Nets' extensions, i.e., colors, hierarchy and time, as embedded in ExSpect$^{TM}$, allow for an easy mapping of high-level manufacturing concepts captured in the modeling framework to ExSpect$^{TM}$ logic. Especially, the job concept, i.e., a manufacturing operation, is well supported by the ExSpect$^{TM}$ time mechanism, which links progress of time to movable objects – being part of a job – in terms of time stamps for tokens. Adapting the approach for other frameworks, like DEVS, may be a fruitful avenue for further research. For example, the agent concept in our modeling framework builds on the atomic model in DEVS.

Low-levelness of formal approaches may cause both a greater need and opportunity for the analyst in realizing model transparency and model completeness. The opportunity follows from the descriptive power of formal approaches, which allows embedding modeling frameworks to a greater extent, whereas many commercial simulation tools may be caught in their own (pragmatic) logic. This suggests a great potential for improving stakeholder understanding.

## ACKNOWLEDGEMENTS

The authors thank three anonymous referees for their helpful suggestions.

## REFERENCES

Barber, K.D., Dewhurst, F.W., Burns, R.L.D.H., and J.B.B. Rogers. 2003. Business-process modelling and simulation for manufacturing management - A practical way forward. *Business Process Management Journal* 9(4): 527-542.

Barretto, M. R. P., Chwif, L., and R.J. Paul. 1999. Combining the Best of the two: An Activity Cycle Diagram/ Condution Specification Approach. In *Proceedings of the UKSIM 1999*, 93-98. United Kingdom Simulation Society.

Booch, G. 1994. *Object-oriented Analysis and Design with applications*. Redwood City: Benjamin Cummings.

Burton-Jones, A., and P.N. Meso. 2006. An Empirical Test of Decomposition Principles in Object-Oriented Analysis. *Information Systems Research* 17(1): 38–60.

David, R., and H. Alla. 1994. Petri Nets for the Modeling of Dynamic Systems - A Survey. *Automatica* 30(2): 175-202.

DiCesare, F., Harhalakis, H., Proth, J.M., Silva, M., and F.B. Vernadat. 1993. *Practice of Petri Nets in Manufacturing*. London: Chapman & Hall.

ExSpect. 1999. User Manual. Diemen: Deloitte & Touche Bakkenist. Available via <http://www.exspect.com/> [accessed March 30, 2009].

Jensen, K. 1997. Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts, In *Monographs in Theoretical Computer Science*. 2nd ed.. 2nd corr. printing. Berlin: Springer-Verlag.

Jensen, K. 2009. Petri Nets World - online services for the international Petri Nets community. Available via <http://www.informatik.uni-hamburg.de/TGI/PetriNets/> [accessed March 30, 2009]

Kamper, S. 1991. On the Appropriateness of Petri Nets in Model Building and Simulation. *Systems Analysis Modelling Simulation* 8(9): 689-714.

Kreutzer, W. 1986. System Simulation – Programming Styles and Languages. Sydney: Addison-Wesley.

Lefrancois P., and B. Montreuil. 1994. An object-oriented knowledge representation for intelligent control of manufacturing workstations. *IIE Transactions* 26(1): 11-26.

Mize, J.H., H.C. Bhuskute, D.B. Pratt, and M. Kamath. 1992. Modelling of Integrated Manufacturing Systems Using an Object-Oriented Approach. *IIE Transactions* 24(3): 14-26.

Murata, T. 1989. Petri Nets : Properties, Analysis and Applications. *Proceedings of the IEEE* 77(4): 541-580.

Petri, C.A. 1962. Kommunikation mit Automaten. PhD thesis, Institut für instrumentelle Mathematik, Bonn, Germany.

Pidd, M. 1998. *Computer Simulation in Management Science*. 4th ed. Chichester: Wiley.

Pidd, M. 1999. *Tools for Thinking – Modelling in Management Science*. 2nd ed. Wiley: Chichester.

Robinson, S. 2008. Conceptual modelling for simulation Part I: definition and requirements. *Journal of the Operational Research Society* 59(3): 278-290.

Ryan J., and C. Heavey. 2006. Process modeling for simulation. *Computers in Industry* 57(1): 437-450.

Schippers, J.P. 1995. Een flexibel simulatiemodel ten behoeve van de evaluatie van diverse besturingsalgoritmen voor de supply chain van een margarinefabriek. Technical Report. University of Twente, The Netherlands (in Dutch).

Smith, E., and D. Medin. 1981. *Categories and Concepts.* Cambridge: Cambridge University Press.

Valentin E.C., and A. Verbraeck. 2005. Requirements for Domain Specific Discrete Event Simulation Environments. In: *Proceedings of the 2005 Winter Simulation Conference,* eds. M.E. Kuhl, N.M. Steiger, F.B. Armstrong and J.A. Joines, 654-663. Piscateway, New Jersey: IEEE.

Van der Aalst, W.M.P. 1992. Timed Coloured Petri Nets and their Application to Logistics. PhD thesis, University of Eindhoven, The Netherlands.

Van der Aalst, W.M.P. 1994. Putting high-level Petri Nets to work in industry. *Computers in Industry* 25: 45-54.

Van der Zee, D.J. 2007. Developing participative simulation models - framing decomposition principles for joint understanding. *Journal of Simulation* 1(3): 187-202.

Van der Zee, D.J., and J.G.A.J. van der Vorst. 2005. A modeling framework for supply chain simulation – Opportunities for improved decision-making. *Decision Sciences* 36(1): 65-95.

Zeigler, B.P. 1990. *Object-Oriented Simulation with Hierarchical, Modular Models, Intelligent Agents and Endomorphic Systems*. London: Academic Press.

## AUTHOR BIOGRAPHY

**DURK-JOUKE VAN DER ZEE** is associate professor of Production Systems Design at the University of Groningen, The Netherlands. He received his M.Sc. and Ph.D. in Industrial Engineering at the University of Twente, The Netherlands. His research interests include simulation methodology and applications, shop floor control systems and design and use of flexible manufacturing systems. His email is <d.j.van.der.zee@rug.nl>.