APPLICATIONS AND EXTENSIONS OF THE UNIFIED MODELLING LANGUAGE UML 2 FOR DISCRETE EVENT SIMULATION

NICOLAS KNAAK, BERND PAGE

Department of Informatics, University of Hamburg
Vogt-Kölln-Straße 30, 22527 Hamburg, Germany
E-mail: [knaak, page]@informatik.uni-hamburg.de

Abstract: Due to the close relation between object-oriented modelling and the domain of discrete event simulation (DES) the Unified Modelling Language (UML) has in the past been frequently applied for simulation modelling. However, most of this work is restricted to particular application domains or diagram types and based on the UML 1.x versions that had some drawbacks concerning dynamic modelling and formal precision. In this paper we provide a short overview of the new UML version 2.0 and sum up its benefits for discrete event simulation. We evaluate related work concerning the use of UML in DES and state some general directives of how to apply several diagram types. The treatment concentrates on UML 2 activity diagrams and relates them to the well-known DES modelling world views. Finally we provide an outlook on our further work that is especially concerned with generating simulation code from UML diagrams for our Java-based simulation framework DESMO-J.

Keywords: Object-Oriented Simulation, Unified Modelling Language, UML 2, Discrete-Event Simulation

1 INTRODUCTION

As Paul Fishwick pointed out in a panel discussion at the 2003 Winter Simulation Conference, *"modelling is one of the primary components of simulation"* [Barton et al., 2003]. One of the most crucial steps during the course of a simulation study is to build a *conceptual model* from system descriptions, observations, data, hypotheses, and a-priori knowledge according to a problem definition. The conceptual model is required to be *transparent, valid, understandable, and preferably easy to transform into a computer implementation*.

In the history of discrete event simulation (DES) many different notations and formalisms like conventional flow charts, event graphs [Schruben, 1983] or Petri nets (e.g. [Bause and Kritzinger, 1996]) have been proposed and applied for conceptual modelling. In an object-oriented discrete simulation context it seems obvious to make use of today's standard graphical notation for object-oriented modelling, the Unified Modelling Language (UML).

Using UML stands to reason even more, since the current version 2.0 contains significant improvements concerning *dynamic behaviour modelling* as a key aspect in discrete simulation. *UML behaviour diagrams* are more common in practical and educational applications than formalisms like Petri nets, but they are nevertheless given a quite concise semantic in version 2.0.

The idea of using *UML as a modelling language in simulation* is not new. However, existing work often either concentrates on particular application domains such as network performance analysis [De Wet and Kritzinger, 2004] or presents specific extensions and applications of certain diagram types (e.g. UML 1.x activity diagrams in [Öchslein et al. 2001]), often without regard to the UML standard and its inherent extension mechanisms.

In this contribution we propose how to *generally apply and extend several UML diagram types for DES modelling*. In particular, we show relations between the elements of UML 2 activity diagrams and the dominant world views of DES, i.e. event-scheduling, process-interaction, activity scanning, and transaction-oriented simulation (see e.g. [Page and Kreutzer, 2005, pp.98-132]).

The paper is organized as follows: In Section 2 we provide a short overview of the new UML 2.0 and sum up important differences from the preliminary versions. We introduce the *modelling language's different diagram types*, its main design principles and briefly discuss *meta-model and extension mechanisms*.

In Section 3 we review related work concerning the use of UML in simulation that is mostly based on the UML 1.x versions. Section 4 contains some general directives of how several UML diagram types can be employed for DES.

Section 5 concentrates on *UML 2 activity diagrams* and relates them to the well-known DES modelling world views. Section 6 concludes the paper and provides an outlook on our further work that is especially concerned with generating simulation code from UML diagrams for our Java-based simulation framework DESMO-J [Lechler and Page, 1999; Page and Kreutzer, 2005, Ch. 10]. The UML 2 specific facts presented in the text are mostly based on the German textbooks "UML 2 Glasklar" by [Jeckle et al., 2002] and "Softwareentwicklung mit UML 2" by [Born et al., 2004].

2 DIAGRAM TYPES AND DESIGN PRINCIPLES OF THE UML

According to the *UML reference manual*, the *Unified Modelling Language is "a general-purpose visual modeling language that is used to specify, visualize, construct, and document the artifacts of a software system"* [Rumbaugh et al., 1999, p. 3]. As [Jeckle et al., 2002, p. 10] point out, it is *not* "complete, not a programming language, not a formal language, not specialized to an application area and [...] first of all not a method or software process".

The term "unified" refers to the fact that the UML is a unification of various popular object oriented (OO) modelling techniques from the beginning 1990s. This era, that is sometimes referred to as the "method wars" (see e.g. [Born et al., 2004, p. 13]), was ended when the authors of the most widely used OO techniques, James Rumbaugh, Ivar Jacobson and Grady Booch, "began to adopt ideas from each other's methods" [Booch et al., 1999, p. xix] by the mid 1990s. As a result of these efforts the Unified Modelling Language version 0.9 was presented in 1996 [Jeckle et al., 2002, p. 12].

The development was carried on under the patronage of the Object Management Group (OMG) up to the version 1.5. Due to several drawbacks of the UML 1.x versions a strongly revised UML 2.0 was proposed and adopted as the official UML version in 2005.[1]

As the major version step indicates, the UML 2.0 contains a large number of modifications compared to its predecessors. The new version intends to improve the language structure mainly in the following aspects: The *semantic precision* of the graphical notation is enhanced, thus making *direct execution of models* possible, and the complex language specification was redesigned to become smaller and *more concise* [Jeckle et al., 2004].

In the context of simulation it is of special importance that the UML 2.0 contains numerous improvements concerning *dynamic behaviour modelling*: It introduces two new behaviour diagram types (*timing diagrams* and *interaction overview diagrams*), strongly enhances the expressiveness and formal semantics of the existing types (activity diagrams, interaction diagrams, and, to a lesser extent, statecharts), and finally integrates the *notion of time* [Jeckle et al., 2004]. Though the UML is still not a formal language, there are ambitions to make UML models executable in the context of the *Model Driven Architecture* (MDA; see [Born et al., 2004, pp. 273]).

UML 2.0 contains a total of 13 diagram types to visualise different aspects of object-oriented modelling. According to [Jeckle et al., 2002, p. 16] the UML diagrams can be broadly divided into three classes:

- *Structural diagrams* model the static structure of a system. Among them are *class diagrams*, *object diagrams*, *package diagrams*, *component diagrams*, *composition structure diagrams*, and *deployment diagrams*.
- *Behaviour diagrams* serve to display the dynamic behaviour of objects or components at different levels of detail. This class of diagrams includes *use case diagrams*, *activity diagrams*, *statechart diagrams*, and several *interaction diagram* types.
- *Interaction diagrams* are special behaviour diagrams that focus on the interactions going on between two or more objects in a system. Interaction diagrams can be divided into *sequence diagrams* and *timing diagrams* that emphasise the temporal order of interaction events on the one hand and *communication diagrams* that highlight the general structure of the cooperation between partners in an interaction on the other hand [Jeckle et al., 2002, p. 391]. A new type of interaction diagram is the *interaction overview diagram*. These diagrams represent a mixture between activity diagrams and interaction diagrams showing the causal and temporal interplay between different interaction scenarios [Jeckle et al. 2002, p. 419].

Without going into details of the respective diagram types, it becomes obvious that the UML is a complex and voluminous visual language incorporating a large number of different modelling techniques. Since complexity and partial inconsistency have been major criticisms of the 1.x versions [Jeckle et al., 2002, p. 13], the language definition of the UML 2.0 adheres to some general principles that [Born et al., 2004, p. 10] identify as

- hierarchical and object oriented language design,
- language definition by meta-modelling,
- separation between vocabulary of concept and notation, and accordingly model and diagram.

[1] See `http://www.uml.org`

The notion of meta-modelling refers to the fact that the concepts and notations of the UML are themselves defined in an object-oriented model that is expressed in terms of the UML [Born et al., 2004, p. 12]. This object-oriented language definition makes extensions of the UML quite easy. As a software framework builds the basis for domain-specific software-applications, *UML concepts and diagrams can be customized and extended for special fields like simulation*.

Such extensions are either stated as extensions of the metamodel itself, or by using a lightweight mechanism called *stereotyping* [Born et al., 2004, p. 245]. According to [Jeckle et al., 2002, p. 95] a stereotype is *"a class in the metamodel that is able to further specify other classes [...] by extension"*. The definition of stereotypes is expressed in UML notation using an *extension arrow* with a filled arrowhead [Jeckle et al., 2002, p. 93].

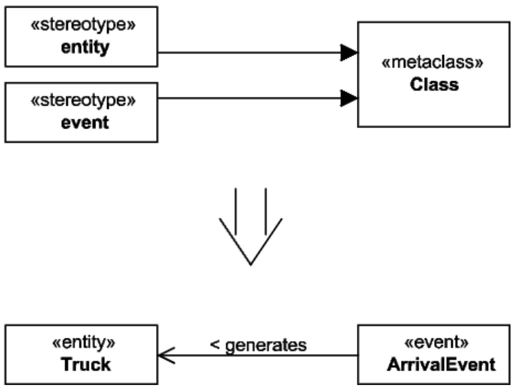


Figure 1 Simulation specific extensions of the UML using the stereotype mechanism (from [Page and Kreutzer, 2005, p. 65]).

Figure 1 shows how stereotypes are used to implement simulation specific UML extensions. To represent entity types in DES models, the metaclass *Class* is extended with a stereotype «`entity`». We state a further specification following [Spaniol and Hoff, 1995, p. 9] by claiming that objects of classes tagged with this stereotype are "able to actively move forward in simulation time".

Now entity types in class diagrams are marked by attaching the term «`entity`» in angle brackets to the respective model elements. Event classes and further extensions in the following sections are specified in a similar manner.

3 RELATED WORK

Traditionally there is a close link between object oriented modelling and the domain of DES. First of all this becomes obvious in the fact that the pioneering object-oriented programming language SIMULA [Birtwistle, 1979] was originally designed for DES. Though conceptual object-oriented modelling in the simulation domain is quite frequently based on UML, we found relatively few examples in the literature on how to systematically apply the different UML diagram types during the course of a simulation study.

[Richter and März, 2000] report on an attempt to apply UML and the Rational Unified Process to the design of simulation models. Their example is not a discrete event model but a simulator for evolutionary algorithms. During the different development phases they apply class diagrams to document the static structure of their simulation software, sequence diagrams to depict interactions between the simulator's components, and statecharts to model the component's behaviour. It should be noted that in this work *UML is used only to document the simulation software* and not the domain model.

[Arief and Speirs, 2000] present a UML tool that is able to generate simulation code for the process-oriented DES library *JavaSim* from class and sequence diagrams. Other simulation world views or diagram types are not supported, but the tool incorporates random variables and simulation statistics. An approach by [Marzolla and Balsamo, 2004] concentrates on the performance evaluation of UML models by mapping them to the process-oriented simulation library *libcppsim*. They use deployment, use-case, and activity diagrams for modelling and focus on the parametrisation and observation of UML models according to the OMG's *UML profile for schedulability, performance and time* [OMG, 2005].

[Öchslein et al., 2001] apply modified UML 1.x activity diagrams to agent-based simulation modelling. They incorporate a large number of modelling elements like object nodes and send-/receive-signal actions and define their own extensions for timed states and "emergency-rules" anticipating some UML 2.0 elements. However, the extended notation can only be handled and executed by their development tool SeSAm. [Köhler et al., 2000] use class, statechart, collaboration, and so called story diagrams to model and simulate production systems with their UML case tool Fujaba.

[De Wet and Kritzinger, 2004] employ UML 2.0 component and statechart models for the performance analysis of network systems. The formal semantics of these UML diagrams is enhanced with inscriptions in the *Specification and Description Language* (SDL) that is common in communication systems engineering. The UML 2.0 compatible case-tool Tau Telelogic is used as an editor. There is a code generator based on the Velocity template engine that generates simulation programs for the process-oriented *SimmCast* framework.

UML is also applied in textbooks on DES. [Garrido, 2001] uses mainly UML 1.x class, statechart, and collaboration diagrams for process-oriented modelling. Some extensions for resource modelling are defined without regard to the UML stereotype mechanism. In [Page et al., 2000] UML 1.x class, activity, statechart, and sequence diagrams are applied for the documentation of simulation models and -software. There are some ad-hoc extensions for modelling time consumption in the process-oriented world view, but their semantics are not very concise.

4 APPLICATIONS OF UML 2 IN DISCRETE EVENT SIMULATION

Due to the *enhanced precision and expressiveness of the dynamic diagram types* the new UML 2.0 is a promising modelling language for DES. Based on the literature review and our experience in simulation teaching and practice, we now present a summary of the benefits of UML 2 and its particular diagram types for DES.

In the first place UML is suited for the DES domain because of the event-based communication model underlying all behaviour diagrams (see [Jeckle et al., 2002, pp. 172]). Due to the widespread use of the modelling language in industry and teaching, simulation model designers, implementers and users benefit from UML diagrams as a common and simulation-software independent basis for documenting, visualizing and understanding the model structure [Richter and März, 2000, p. 2]. The different UML diagrams provide multiple views focusing on either *structural, behavioural, or interaction-related aspects* of the same model.

The quite concise semi-formal semantics of UML 2 behaviour diagrams with relations to formal models like automata and Petri nets also provide support for the task of model validation and verification. The generation of executable code from static and dynamic UML models is an important means to narrow the gap between conceptual and computer models in simulation [Klügl, 2001, p. 80]. A current drawback of adopting UML 2.0 for model driven design[2] is the fact that there are only few case-tools at hand supporting the new version.

In our experience the different UML diagram types are best applied for the following purposes in simulation modelling:

- *Structural Diagrams*: *Class and object diagrams* are useful to depict entities and relations of the system under study during the conceptual modelling phase (see Figure 2). Stereotypes can be employed to relate classes to notions from discrete simulation like e.g. processes or events. *Package and class diagrams* are also suitable to display the structure of simulation software during implementation. *Component, composition structure, and deployment diagrams* might render themselves useful in component-based and distributed simulation. An example is shown in [De Wet and Kritzinger, 2004].
- *Behaviour Diagrams*: *Use case diagrams* can be applied to provide a broad overview of relations between actors and activities in the target system that builds the basis for more detailed event or process descriptions. Behaviour modelling of individual entities and events is done with *statecharts and activity diagrams*. We use statecharts predominantly in an early modelling phase to build an *abstract conceptual model* of the target system's state transitions (see the example in Figure 3). This state model is then refined to one or more *activity diagrams* that are closer to the respective event- or process-oriented simulation style (see Section 5). Due to their Petri-net like token semantic activity diagrams also support the validation of process models either by manual "token-game" simulations or by mapping to executable code.
- *Interaction diagrams*: Since temporal aspects of interactions are of special importance in simulation, we preferably use *sequence and timing diagrams* in simulation modelling. An example is shown in Figure 4. Both diagram types might also be applied in validation to *visualize simulation traces* (see e.g. [Systä, 2000; Cabac et al. 2006]).

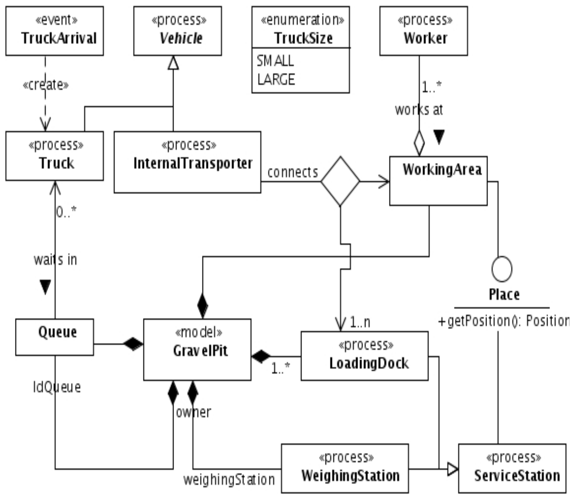


Figure 2: A class diagram of a "Gravel Pit" model (adopted with modifications from [Birtwistle, 1979]) from our simulation course that serves as a continuing example in this paper.

[2] The basic idea behind the notion of model-driven design is that "the model is the implementation" [Selic, 2003; De Wet and Kritzinger, 2004].

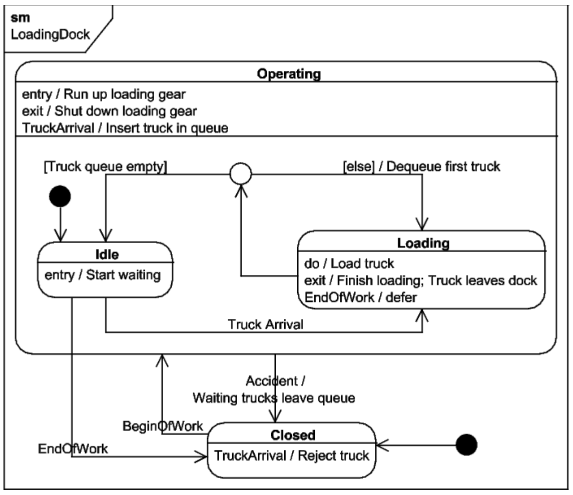

sm
LoadingDock
Operating
entry / Run up loading gear
exit / Shut down loading gear
TruckArrival / Insert truck in queue
[Truck queue empty]
[else] / Dequeue first truck
Idle
entry / Start waiting
Loading
do / Load truck
exit / Finish loading; Truck leaves dock
EndOfWork / defer
Truck Arrival
Accident /
Waiting trucks leave queue
BeginOfWork
EndOfWork
Closed
TruckArrival / Reject truck

Figure 3 The “Gravel Pit” model's loading dock modelled as a hierarchical state machine (from [Page and Kreutzer, 2005, p. 74]). The “emergency exit” behaviour shown here is a common pattern in hierarchical state machines that can e.g. be found in [Köhler, 1999, p. 38].

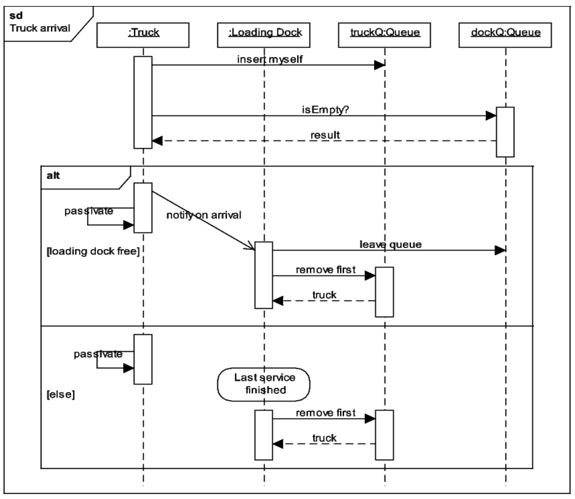


Figure 4 A UML 2 sequence diagram showing two alternative interaction courses on a truck arrival in the “Gravel Pit” model (from [Page and Kreutzer, 2005, p. 91]).

In the following we focus on UML activity diagrams and relate their modelling elements to the common DES world views as they are supported by our simulation framework DESMO-J (Discrete Event Simulation in Java, see [Page and Neufeld, 2003]). Though most elements can be used in their standard form, we also specify some extensions using the stereotype mechanism.

5 RELATING ACTIVITY DIAGRAMS TO THE WORLD VIEWS OF DES

According to [Jeckle et al., 2002, p. 199] *activity diagrams* are the *notation of choice for modelling processes*. While in software engineering the main application of this diagram type is the description of operations, use cases or business processes [Jeckle et al., 2002, p. 199], in DES activity diagrams are well suited for *modelling event routines in event-scheduling* and *life cycles of simulation processes in the process-interaction world view*. Due to features like concurrency, object flow, and message passing they are particularly appropriate to display the synchronization of two or more such processes using advanced modelling constructs related to the activity-scanning or transaction-oriented world view.

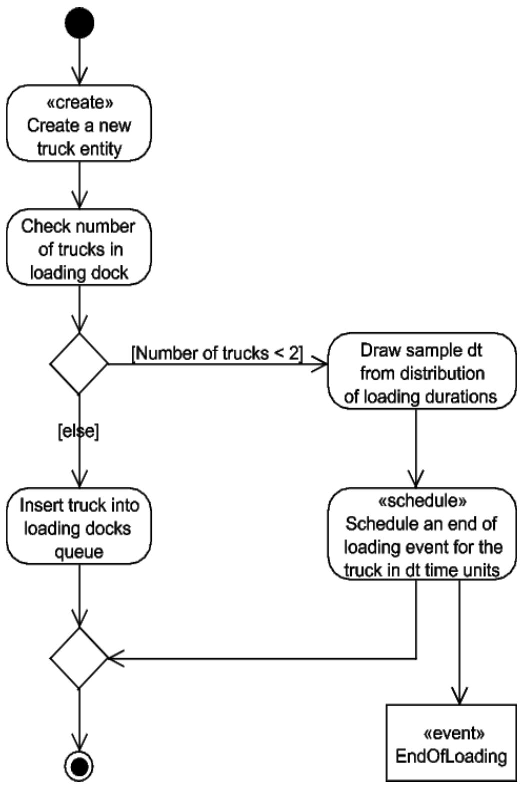

«create»
Create a new truck entity
Check number of trucks in loading dock
[Number of trucks < 2]
Draw sample dt from distribution of loading durations
[else]
Insert truck into loading docks queue
«schedule»
Schedule an end of loading event for the truck in dt time units
«event»
EndOfLoading

Figure 5 An example activity diagram describing the actions performed for the event "arrival of a truck in the gravel pit" .

Activity diagrams have experienced major changes in the transition from version 1.x to 2.0. In UML 1.x, they were a special case of statecharts emphasizing synchronous control flow instead of asynchronous event handling. In the new version, the statechart-like event handling semantic of activity diagrams has been replaced by a *Petri net-like token semantic* making this diagram type even more suitable for modelling concurrent processes.

The most straightforward application of activity diagrams in simulation is the description of event-routines in *event-scheduling simulation*. This world view is characterised by a *top down modelling perspective*. The model's dynamic is specified in event routines of certain event classes that *change the state of the model's entities at discrete points* in time [Page, et.al. 2000, p. 10].

While event graphs [Schruben, 1983] display causal relations between event classes, the *detailled description of event routines* can be done with activity diagrams used basically as flowcharts. Figure 5 shows a simple example drawn from the "Gravel Pit" simulation model. The event routine *Truck Arrival* is composed of elementary actions like creation and modification of entities or scheduling of further events, that we introduce stereotypes for. *Object nodes* (see below) might be employed to indicate *causal relations to other event classes*.

An execution of the event routine can be visualised as a token flow similar to Petri nets. A control token is initially located at the initial node. Each action node fires when there is a token present on all of its incoming edges, consumes the tokens, and produces new tokens on its outgoing edges.

Different from event-scheduling, the *process interaction world view* is characterised by a *bottom-up modelling perspective*, where the model dynamic is specified in terms of simulation processes and their life cycles [Page and Kreutzer, p. 114]. There is an alternation between active process phases where processes change the model state, and passive phases representing either wait states or *time consuming activities*. The main operations in process interaction are

- *passivation* of processes to enter an unconstrained wait-state,
- *activation* of waiting processes, and
- the *hold* operation to enter a time-constrained wait state.

Figure 6 shows two process classes from the "Gravel Pit" example, that synchronize via sending and reception of activation signals. When a truck arrives at the loading dock, it queues up in a *truck queue*. If an idle dock is available, the truck activates it to start loading. This can be denoted in UML by a send-signal action [Jeckle et al., 2002, p. 214] with the stereotype «`activate`». The truck then remains passive until loading is finished.

The passive state is indicated by a receive-signal action [Jeckle et al., 2002, p. 214] with the stereotype «`passive`». The notion of an action is somewhat misleading in this context, because the token flow in the activity diagram is explicitly delayed (thereby forcing the process to wait) until the reception of a matching signal.

As stated in [Jeckle et al., 2005, p. 215] *delays with predefined duration* are modelled using a *time signal reception node* depicted by an hour glass symbol. To conform to the process-interaction world view we tag it with the additional stereotype «`hold`». This node delays incoming tokens for a specified duration. One might also imagine that the modelled process schedules itself a time signal when the node is reached and then waits for its reception.

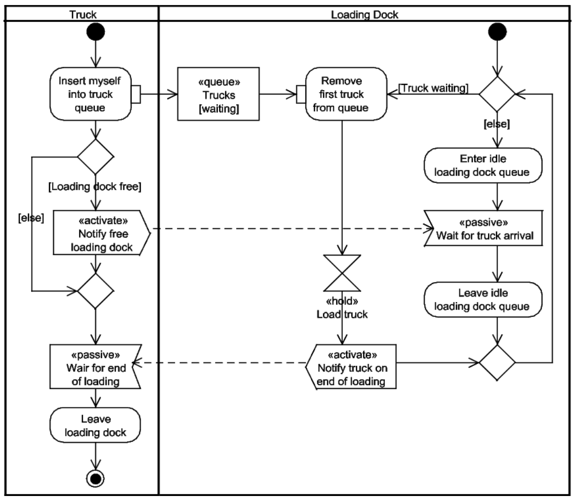

Truck
Loading Dock
Insert myself into truck queue
«queue» Trucks [waiting]
Remove first truck from queue
[Truck waiting]
[else]
Enter idle loading dock queue
[Loading dock free]
[else]
«activate» Notify free loading dock
«passive» Wait for truck arrival
«hold» Load truck
Leave idle loading dock queue
«passive» Wair for end of loading
«activate» Notify truck on end of loading
Leave loading dock

Figure 6 Synchronisation of trucks and loading docks in the "Gravel Pit" model via sending and reception of signals.

While in the example the service duration of the loading dock is stated implicitly by naming the corresponding time consuming activity (*load truck*) UML also provides keywords *after* and *when* for relative and absolute time specifications. In stochastic simulation it should be possible to specify such durations in terms of the underlying random distributions (e.g. *after(Exp(3.0)*) for exponentially distributed durations with a mean of 3 time units).

Note that the dashed arrows between the send and receive actions are not part of the standard UML notation, but only serve to clarify the direction of signal flow between the processes here. In larger simulation models it might not be sensible to display all communicating processes in a single diagram. In this case one would prefer to draw a detailed activity diagram for each process and use sequence or timing diagrams to specify exemplary interactions.

Another important requirement in process-interaction is the ability to model *preemption* and *interrupts* in queueing systems. Figure 7 shows an example of a customer process with limited waiting patience. After a certain waiting period the customer leaves the queue without being serviced. This can be represented using *interruptible activity regions* [Jeckle et al., 2002, pp. 241] in activity diagrams.
The interruptible activity region is depicted by a rounded rectangle with a dashed outline containing a

“blizzard”-shaped interrupt edge [Jeckle et al., 2002, pp. 242]. In the example, the customer process receives a *begin of service* signal that triggers the corresponding signal reception action at the beginning of service. This causes the interruptible activity region to be left via the interrupt edge. Different from a standard edge this step removes *all* tokens from the region. Thus no further actions within the region can be executed.

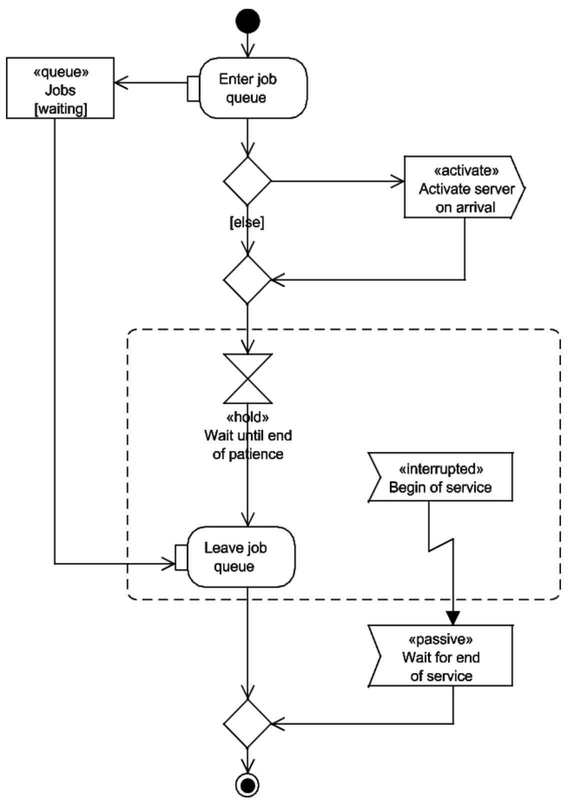


Figure 7 A customer with limited patience modelled with an interruptible activity region.

The ability of activity diagrams to display data flow is also interesting in the context of process-interaction. The main construct for *modelling data flow* are *object nodes* depicted by rectangles [Jeckle et al., 2002, pp. 218]. When the outgoing edge of an action node is connected to an object node, execution of the action produces a so called *data token* that contains the result object of the execution. The data token is *stored in the object node* and might serve as input to another action with its incoming edge connected to the object node.

A useful property of object nodes is their ability to *buffer incoming data tokens*. This allows them to be used as synchronisation constructs in queueing and resource models. Figure 8 shows the interaction between trucks and the gravel pit's loading dock as a simple *single server queueing system*. A truck generator process creates truck objects at a certain arrival rate and inserts them into a *queue modelled as an object node*. We use the stereotype «queue» to indicate that an object node has a queue semantic. The inscription names the type of entities to be stored in the queue. The optional inscription in square brackets is a condition constraining the state of all entities in the object node.

In the example, trucks are dequeued by the loading dock's *remove from queue* action. The semantic of the edge connecting this action node to the queue is similar to standard edges in activity diagrams.
When the queue node provides a data token (i.e. there is a truck waiting) and the standard edge above the action nodes provides a control token, the action node fires and removes one truck object from the queue. According to the UML 2 standard, there is a *note symbol* with the stereotype «selection» [Jeckle et al., 2002, p. 227] attached to the edge that indicates the selection strategy FIFO (or any other queueing strategy).
If no truck is present, the loading dock process waits for the arrival of the next truck. Note that queues used in process oriented simulation software (e.g. in the DESMO-J framework) often do not have these synchronization capabilities. In this case, one has to check the presence of objects in queues and synchronize processes explicitly via signals.

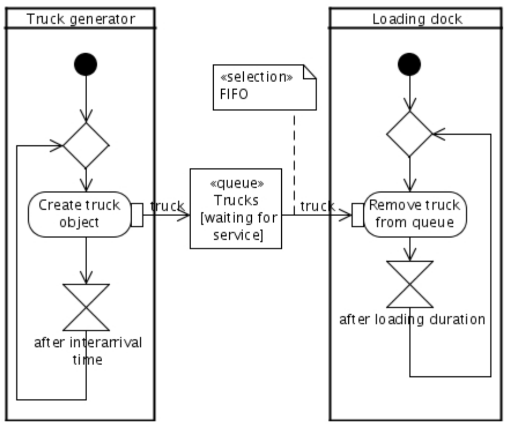
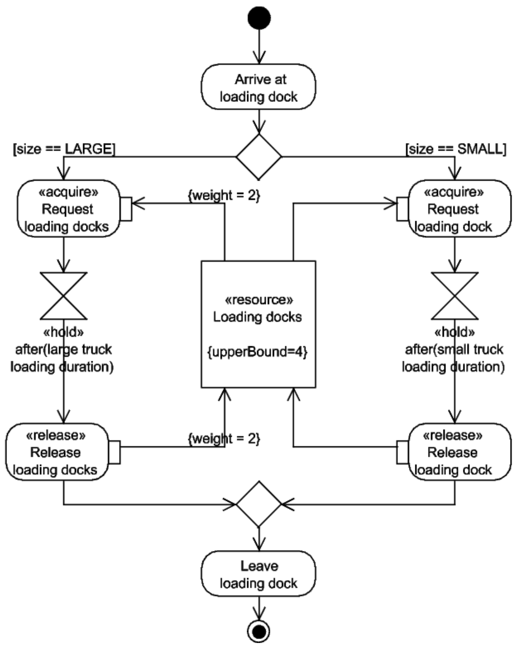


Figure 8 A queue modelled as an object node (from [Page and Kreutzer, 2005, p. 84]).

Object nodes also map directly to advanced synchronisation constructs used in the transaction oriented world view, that is especially suited for modelling production systems. Transaction-oriented models consist of static *blocks* (e.g. machines) and dynamic *transactions* (e.g. jobs) that change their state while flowing through a network of blocks. Such models can easily be reduced to process-oriented models by mapping processes to transactions and blocks to *resources with internal wait queues* [Page et al., 2000, p. 17; p. 98].
An example of transaction oriented modelling with UML 2 activity diagrams is shown in Figure 9. It displays the life cycle of a refined truck process

from the "Gravel Pit" model. Here trucks might have two different sizes LARGE and SMALL. To become loaded, large trucks require two of 4 available loading docks while small trucks require only one. The loading docks are represented as a *resource with a certain capacity*.

The resource semantic of the corresponding object node is indicated by the stereotype «`resource`». As before, the truck process waits in its life cycle when there are too few resources available for service. In UML 2 the *maximum capacity of an object node* can be specified using the attribute `upperBound` written in curly braces. Similarly, the number of tokens an edge consumes or produces is specified using the attribute `weight` where the default weight of 1 is not stated explicitly.



Arrive at loading dock
[size == LARGE]
[size == SMALL]
«acquire» Request loading docks
{weight = 2}
«acquire» Request loading dock
«resource» Loading docks
{upperBound=4}
«hold» after(large truck loading duration)
«hold» after(small truck loading duration)
«release» Release loading docks
{weight = 2}
«release» Release loading dock
Leave loading dock

Figure 9 Interaction between trucks and loading docks modelled with a resource (from [Page and Kreutzer, 2005, p. 85]).

In the activity scanning world view the model dynamic is described in a declarative way by specifying activities that are executed whenever their preconditions hold [Page et al., 2000, p. 17]. Nevertheless this world view can be reduced to process interaction by introducing conditional wait queues [Page et al., 2000, pp. 102]. Each condition is represented by a wait queue where processes wait in their life cycle until the expected condition holds. To improve performance of such simulations, it is advisable to explicitly re-check these conditions only on changes of the model state.

We represent conditional waiting in UML 2 activity diagrams by receive-signal actions with the stereotype «`waitOnCondition`». Re-checking a condition is requested by a send-signal action with stereotype «`checkCondition`». Figure 10 shows an example from the "Gravel Pit" model where large trucks arriving at the loading dock wait for large diggers to become available while small trucks are only served by small diggers (adopted with modifications from [Page et al., 2000, pp. 145]).

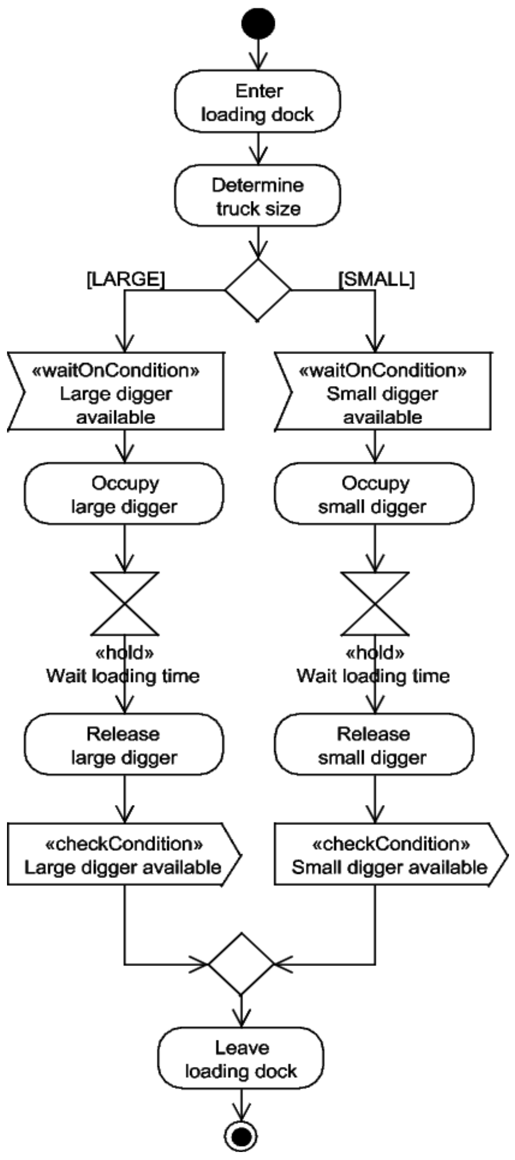


Figure 10 Interaction between trucks and loading docks modelled with conditional waiting.

We finish this section with the presentation of a larger activity diagram showing a machine process from a complex production model. The diagram in Figure 11 combines many of the constructs described in this section; e.g. interruptible activity regions and queue object nodes. The model is built using a combination of event-scheduling and process-interaction modelling constructs. Unlike many other simulation frameworks, DESMO-J offers the possibility to implement such combined models [Page and Kreutzer, 2005, pp. 135-140]. Note the occurence of stereotypes related to process-interaction (e.g. «hold») and event-scheduling (e.g. «cancel») within the same activity diagram.

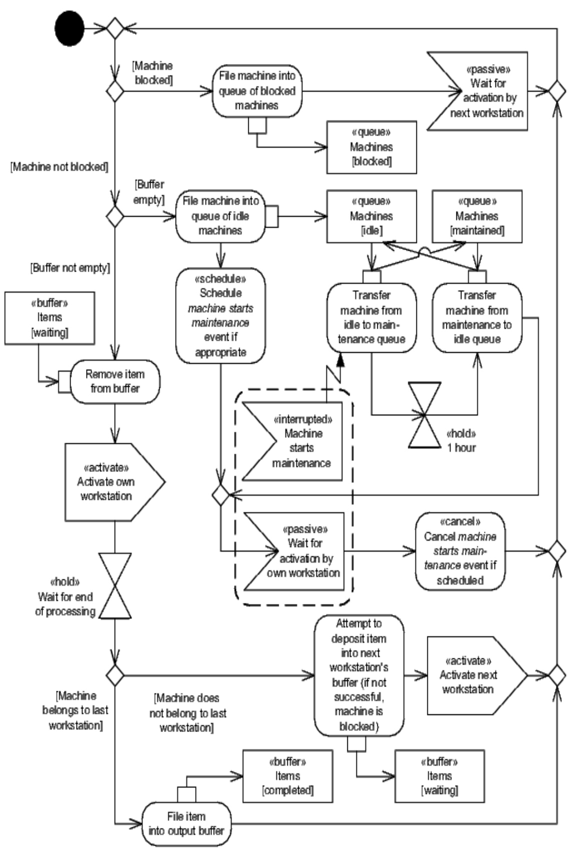

[Machine blocked]
File machine into queue of blocked machines
«passive» Wait for activation by next workstation
«queue» Machines [blocked]
[Machine not blocked]
[Buffer empty]
File machine into queue of idle machines
«queue» Machines [idle]
«queue» Machines [maintained]
[Buffer not empty]
«schedule» Schedule machine starts maintenance event if appropriate
Transfer machine from idle to maintenance queue
Transfer machine from maintenance to idle queue
«buffer» Items [waiting]
Remove item from buffer
«interrupted» Machine starts maintenance
«hold» 1 hour
«activate» Activate own workstation
«passive» Wait for activation by own workstation
«cancel» Cancel machine starts maintenance event if scheduled
«hold» Wait for end of processing
Attempt to deposit item into next workstation's buffer (if not successful, machine is blocked)
«activate» Activate next workstation
[Machine belongs to last workstation]
[Machine does not belong to last workstation]
«buffer» Items [completed]
«buffer» Items [waiting]
File item into output buffer

Figure 11 A larger example for simulation modelling with activity diagrams. The diagram shows a machine process from a complex production model that was built using a combined process-interaction and event-scheduling modelling style (from [Page and Kreutzer, 2005, p. 138]).

6 CONCLUSIONS AND FUTURE WORK

In this paper we have discussed the benefits of the new UML version 2.0 for the domain of discrete event simulation. Due to the enhanced conciseness and expressiveness of the behaviour diagrams, the new version has become an even more promising candidate for a standard modelling language in DES. Therefore it is necessary to systematically evaluate further for what purpose different UML diagram types can be applied best in simulation modelling. We include a proposal based on a literature review and on our practical and teaching experience in Section 4 of this paper.

In our opinion the enhanced UML 2.0 activity diagrams relate very well to the dominant world views of DES. While in event-scheduling, simple activity diagrams are used to specify event-routines, we argue that their main application domain is in process-interaction modelling based on the UML event model. Object nodes can be employed to represent synchronisation constructs in transaction-oriented or activity-scanning models. Though most elements of activity diagrams map directly to concepts from DES, some simulation-specific extensions (e.g. resources) were defined using the UML's stereotype mechanism.

In our further work we will evaluate the UML 2 notation in our simulation courses based on a simulation textbook which has been published recently [Page and Kreutzer, 2005]. Moreover we are developing a code generator that maps the activity diagram notation shown in Section 5 to code for our Java-based simulation framework DESMO-J [Lechler and Page, 1999]. An extended version of the light weight UML drawing tool *UMLet*[3] is used as a diagram editor that can also be integrated as a plugin into the popular *Eclipse*[4] development platform.

For code generation we employ a template-based approach using the *Velocity* template engine.[5] Template-based code generation provides the advantage that the destination language or framework can easily be changed due to a layered architecture in conformance with the viewpoints of the OMG's Model Driven Architecture [Born et al., 2004, p. 279]. Unlike [De Wet and Kritzinger, 2004] we use Java instead of SDL as inscription language, thus making activity diagrams more generally applicable at the cost of some formal precision.

At the conceptual level, a further evaluation of the OMG's *UML profile for schedulability, performance and time* [OMG, 2005] and its relation to the DES domain might be of interest (see also [Marzolla and Balsamo, 2004]). Work in this area could possibly lead to a standard UML profile for discrete event modelling in the future.

REFERENCES:

Arief L.B. and Speirs N.A. 2000, “A UML Tool for an Automatic Generation of Simulation Programs”. In *WOSP 2000*. Ontario, Canada.
Barton R.R. and Fishwick P.A. and Henriksen J.O. and Sargent R.G. and Twomey J.M. 2003, “Panel: Simulation: Past, present and future”. In *Proceed-*

[3] http://www.umlet.com
[4] http://www.eclipse.org
[5] http://jakarta.apache.org/velocity

ings of the 2003 Winter Simulation Conference, pp. 2044 - 2050.

Bause F. and Kritzinger P. 1996, *"Stochastic Petri Nets - An Introduction to the Theory"*. Vieweg, Braunschweig.

Birtwistle G.M. 1979, *"DEMOS, a System for Discrete Event Modelling on Simula"*. Macmillan, London.

Booch G. and Rumbaugh J. and Jacobson I. 1999, *"The Unified Modeling Language User Guide"*. Addison-Wesley, Reading (MA).

Born M. and Holz E. and Kath O. 2004, *"Softwareentwicklung mit UML 2 - Die neuen Entwurfstechniken UML 2, MOF 2 und MDA"*. Addison-Wesley, München.

Cabac L. and Knaak N. and Moldt D. and Rölke H. 2006, "Analysis of Multi-Agent Interactions with Process Mining Techniques". Accepted for the *4th German Conference on Multiagent System Technologies (MATES 2006)*, Erfurt, September 19-21 2006.

De Wet N. and Kritzinger P. 2004, "Using UML Models for the Performance Analysis of Network Systems". In *Proceedings of the Workshop on Integrated-reliability with Telecommunications and UML Languages (WITUL)*. Rennes, Brittany, France.

Garrido J.M. 2001, "*Object-Oriented Discrete-Event Simulation with Java - A Practical Introduction"*. Kluwer Academic / Plenum, New York.

Jeckle M. and Rupp C. and Hahn J. and Zengler B. and Queins S. 2002, "*UML 2 glasklar"*. Carl Hanser Verlag, München.

Jeckle M. and Rupp C. and Hahn J. and Zengler B. and Queins S. 2004, "UML 2.0: Evolution oder Degeneration". In *Objekt Spektrum*, pp. 12-19.

Klügl F. 2001 *"Multiagentensimulation - Konzepte, Werkzeuge, Anwendung"*. Addison-Wesley, München.

Köhler H.J. 1999, "*Code-Generierung für UML Kollaborations-, Sequenz- und Statechart-Diagramme"*. Master's thesis, University of Paderborn.

Köhler H.J. and Nickel U. and Niere J. and Zündorf A. 2000, "Integrating UML Diagrams for Production Control Systems". In *Proceedings of the 22nd International Conference on Software Engineering (ICSE)*, ACM Press, Limerick, Ireland, pp. 241-251.

Lechler T. and Page B. 1999, "DESMO-J - An Object-oriented Discrete Simulation Framework in Java". In *Simulation in Industry 99 - 11th European Simulation Symposium 1999 in Erlangen (Germany)*, SCS, Delft (Netherlands), pp. 46-50.

Marzolla M. and Balsamo S. 2004, *"UML-Psi: The UML Performance Simulator"*. Universita Ca Foscari di Venezia, Dipartmento di Informatica.

Object Management Group 2005, "*UML Profile for Schedulability, Performance and Time v1.1"*. Full Specification formal/05-01-02, OMG.

Öchslein C. and Klügl F. and Herrler R. and Puppe F. 2001, "UML for Behaviour-Oriented Multi-Agent Simulations". In *Proceedings of the CEEMAS*, Springer, Berlin, pp. 217-226.

Page B. and Lechler T. and Claassen S. 2000, *"Objektorientierte Simulation in Java mit dem Framework DESMO-J"*. Libri Books on Demand, Hamburg.

Page B. and Neufeld E. 2003, "Extending an object oriented Discrete Event Simulation Framework in Java for Habour Logistics". In *Proc. International Workshop on Harbour, Maritime & Multimodal Logistics Modelling and Simulation – HMS 2003*, Riga, Latvia, pp. 79-85.

Page B. and Kreutzer W. 2005, "*The Java Simulation Handbook. Simulating Discrete Event Systems with UML and Java*". Shaker Publ., Aachen.

Richter H. and März L. 2000, "Toward a Standard Process: The Use Of UML for Designing Simulation Models". In *Proceedings of the 2000 Winter Simulation Conference*, pp. 394-398.

Rumbaugh J. and Jacobson I. and Booch G. 1999, "*The Unified Modeling Language Reference Manual"*. Addison-Wesley, Reading (MA).

Schruben L. 1983, "Simulation Modeling with Event Graphs". In *Communications of the ACM*, 13, pp. 265-275.

Selic B. 2003, "Brass Bubbles: An overview of UML 2.0 and MDA". In *Object Technology Slovakia*.

Spaniol O. and Hoff S. 1995, *"Ereignisorientierte Simulation - Konzepte und Systemrealisierung"*. Internationale Thompson Publ., Bonn.

Systä T. 2000, "Understanding the Behavior of Java Programs". In *Proceedings of the 7th Working Conference on Reverse Engineering (WCRE 2000)*, Brisbane, Australia, pp. 214-223.

BIOGRAPHIES:

Dipl.-Inform. Nicolas Knaak obtained his diploma degree (MSc) in Computer Science from the University of Hamburg, Germany in 2002. Since then he has been working as a scientific assistant and PhD candidate in the simulation group led by Prof. Page at the University of Hamburg. He has taught courses on Java programming, algorithms and data structures, and computer simulation. From 2001 to 2003 he worked on a funded research project dealing with the agent-based simulation of city-courier services. His current research interests are UML modelling as well as the verification and validation of Multi-agent systems. At the moment he is preparing his PhD thesis on the validation of Multi-agent systems by means of simulation and process-mining techniques.

Prof. Dr.-Ing. Bernd Page received a PhD in Applied Computer Science from the Technical University of Berlin (Germany) and a Masters degree from Stanford University (USA). Since his university appointment he has been involved in research, teaching, and applications mainly in computer simulation and environmental informatics. Professor Page has published widely, resulting in a large number of papers for national and international conferences and several books. Professor Page was a cofounder and has for many years been the chairman of the working group ``Informatics for Environmental Protection" in the German Computer Society (GI). More recently Professor Page has intensified his interests in modelling logistic systems, an area where discrete event simulation tools such as the DESMO-J framework, which has been under development by his research group for many years, can be applied very effectively. His current research is guided by the idea of combining economic and environmental aspects of logistics models (i.e. “ecologistics”) and centers on the design of suitable tools for such models' construction and use. Professor Page has held a large number of research and development grants for public and industrial projects.