Visualizing the Dynamic Behavior of ProC/B Models

Peter Kemper and Carsten Tepper[*]
Universität Dortmund

Abstract

Process interaction is a well known paradigm to formulate simulation models. In practice, complex models of interacting processes require substantial effort in the verification and validation phase of a simulation study. A simulator trace can document all activities performed in a simulation run. In this paper, we present two visualization aids to investigate the dynamic behavior of a simulation model based on traces. The first approach animates the graphical model description to visualize a trace in the same language as used to specify the model itself. The second approach employs a variant of message sequence charts or sequence diagrams to give an orthogonal illustration of interacting processes that generate the dynamic behavior. Both approaches have been implemented and evaluated. We consider a particular process-oriented modeling formalism, the ProC/B notation, and its corresponding tool, the ProC/B Toolset, that have been developed for the modeling and analysis of logistics networks.

1 Introduction

A modeling notation that resembles the lingo and perception of the way practitioners typically consider an application problem is more straight forward to use and more likely to achieve user acceptance in practice. The ProC/B Toolset [1] is a modeling and simulation environment whose process-oriented modeling notation follows a modern but common way of considering logistics networks. From a conceptual point of view, the ProC/B notation yields simulation models that are service networks with active (like queueing servers) and passive resources (like storage areas) where a dynamic number of tasks/processes are generated according to a given set of (possibly complex) behavior patterns. Discrete event simulation [7] is used to evaluate those models. However, we experienced that models often do not perform as intended, and bugs have to be identified and removed. Non-trivial problems arise from interaction of processes and simultaneous resource allocation. Those problems often only document themselves in unexpected values of performance measures conducted from simulation runs. As most simulators, the ProC/B simulator is able to create a trace to document what happened. However, such simulation traces yield often too much data such that error detection becomes a laborious and painful process.
In this paper, we propose two visualization aids that help to analyze dynamic behavior by help of traces. The first approach animates behavior by highlighting activities in the corresponding original model specification. The second approach uses a variant of message sequence charts (MSCs) to put the focus on interactions among processes and resources of

[*]Informatik IV, Universität Dortmund, D-44221 Dortmund, Germany. This research is supported by Deutsche Forschungsgemeinschaft, Sonderforschungsbereich 559 'Modelling of Large Logistics Networks'.

a model. Both approaches have their strengths and limitations, however in combination they increase productivity in the validation and debugging of simulation models. Clearly, most simulation environments provide supporting features to debug simulation models. Concepts are mainly analogous to what is known for debugging software in general. For instance, Arena [6] provides features to track back syntax errors detected in the compilation and linking process that generates the simulator code from a graphic model description. In addition in Arena, the simulator code can be performed step-by-step or automatically up to a breakpoint, with intermediate results presented and the active pieces of code being animated. This is similar to what a debugger does and what our animation approach does as well. On the other hand, visualizing communicating processes by MSC-like graphics is natural and common in parallel programming, for instance the XPVM environment [5] visualizes behavior of parallel programms in MSCs-like graphics called space-time view.
So, neither animation of simulation models nor MSCs are novel ideas as such. We see the contribution of this paper in that we employ both for debugging simulation models, in the implementation and integration into a tool and the way they have been designed to handle the problem of largeness in terms of the size of models and the lengths of traces. In both cases, largeness demands structure and abstraction to help a user in gaining insight into the behavior of a model. While the direct animation has to follow the hierarchical structure of a model as imposed by the ProC/B formalism, MSCs support a compositional structure by grouping interacting processes and resources. Note that both concepts are in general applicable to simulation models with interacting processes; its discussion with respect to the ProC/B formalism can be seen as an exemplification.
The paper is structured as follows. Section 2 provides a brief overview of the ProC/B formalism and presents a running example. Section 3 shows how visualizing the dynamic behavior can help to understand the reasons for a deadlock in a model. The ProC/B Toolset supports an interactive exploration of the dynamic behavior where decision on the next event is made by a user and it also supports automatic trace generation by a simulator and subsequent animation of that given trace. In addition, a trace can be visualized as an MSC. In section 4, we explain how this visualization is realized and propose operations to enhance clarity of the represented data. After that we evaluate both concepts and conclude.

2 ProC/B formalism

The ProC/B formalism [1] is a modeling language that is especially designed to the needs of logistics networks. It is the common specification language of the collaborative research center "Modeling of Large Logistics Networks" (SFB 559)[1] at the university of Dortmund. The formalism integrates two notions of hierarchy. Since systems are often build in a hierarchical manner from subsystems whose functionality is combined and integrated into a more complex functionality provided by the overall system, the ProC/B formalism captures those structures by functional units (FUs) that can include sub-FUs. Behavior is expressed in process chains (PCs), i.e., process patterns made up of activities. FUs provide services which may be called from their environments, and whose detailed behavior is defined internally within its FU. PCs may make use of services provided by FUs to perform their own activities. The hierarchical description has sources and sinks at its top level to interact with an environment and has predefined standard functional units to close the hierarchy at any bottom level. Those standard FUs are servers to model time consumption at shared

[1]see http://www.sfb559.uni-dortmund.de/index.php?lang=eng

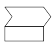
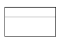
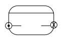
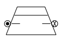
ProC/B element	Symbol	Semantic for Dynamic Behavior
`Source`	$\odot$	process creation
`Sink`	$\otimes$	process termination
`Process Chain Element` (PCE)		activity in a process chain, semantics: (stochastic) delay, service call
`Sequence`	$\longrightarrow$	sequential order of activities, from left to right
`AND-Connector`		synchronization of incoming activities (left) with joint start of outgoing activities (right)
`OR-Connector`		probab. /bool. branching, selects single activity (right), triggered by termination of any activity from left
`PC-Connector`		synchronization of processes (left side) continuation or creation of processes (right side)
`Functional Unit` (FU)		structuring element, a form of container, provides services, contains resources
`Server`		familiar queueing station
`Counter`		familiar passive resource (semaphore etc.)

Table 1: Overview: Core ProC/B modeling elements: graphical denotation and semantics

resources due to service and waiting time and counters to model space consumption. The latter may induce delays due to blocking effects for service calls, e.g., a counter keeps track of the number of goods on stock and a service call to remove goods will block if the stock is empty. Table 1 includes core ProC/B modeling elements.

We illustrate the approach by an example. The model reflects a small fraction of what needs to be done by a car manufacturer in the production of a car. It considers ordering, delivery and installation of door modules to the auto body. A supplier provides the door modules and trucks carry them by help of transport frames from the supplier to the manufacturer. A limited number of trucks and frames circulate between supplier and manufacturer. The manufacturer has door modules on stock and reorders replacements whenever door modules are used. The upper part of Fig. 1 shows three process chains. The top one, 'OrderTyp1' describes processing cars with 2 doors. The description is read from left to right and starts with a source node that specifies that processes of that kind are generated individually every k time units, where k is a random number with a negative exponential distribution with rate $\lambda = 0.5$. The first PCE models some preproduction steps by a single activity with an exponentially distributed delay. The following PC-connector describes that two things happen in parallel, the instance of a process itself continues with activity 'getDoorModule' and a new process 'OrderDoorTyp1' is created whose first activity is 'OrderTyp1DoorModule'. Instances of processes proceed in an asynchronous manner till they terminate at a sink node at the right side. A direct interaction among instances of processes is possible by PC-connectors which allow for synchronization, creation and termination, and indirectly by service calls to servers and counters. Note that an activity may be a simple delay as for the first element 'PreProduction' or a service call to a FU, a server

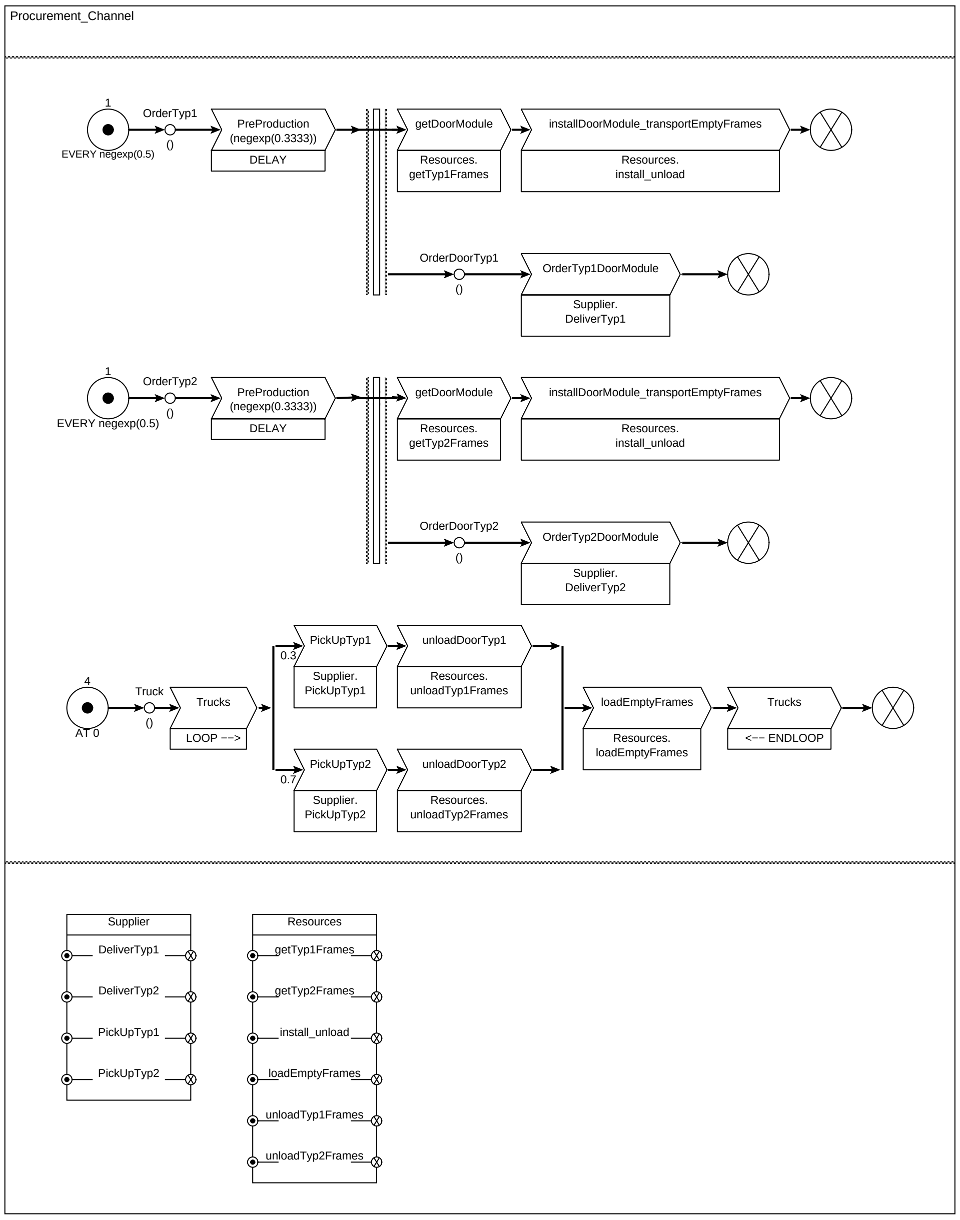


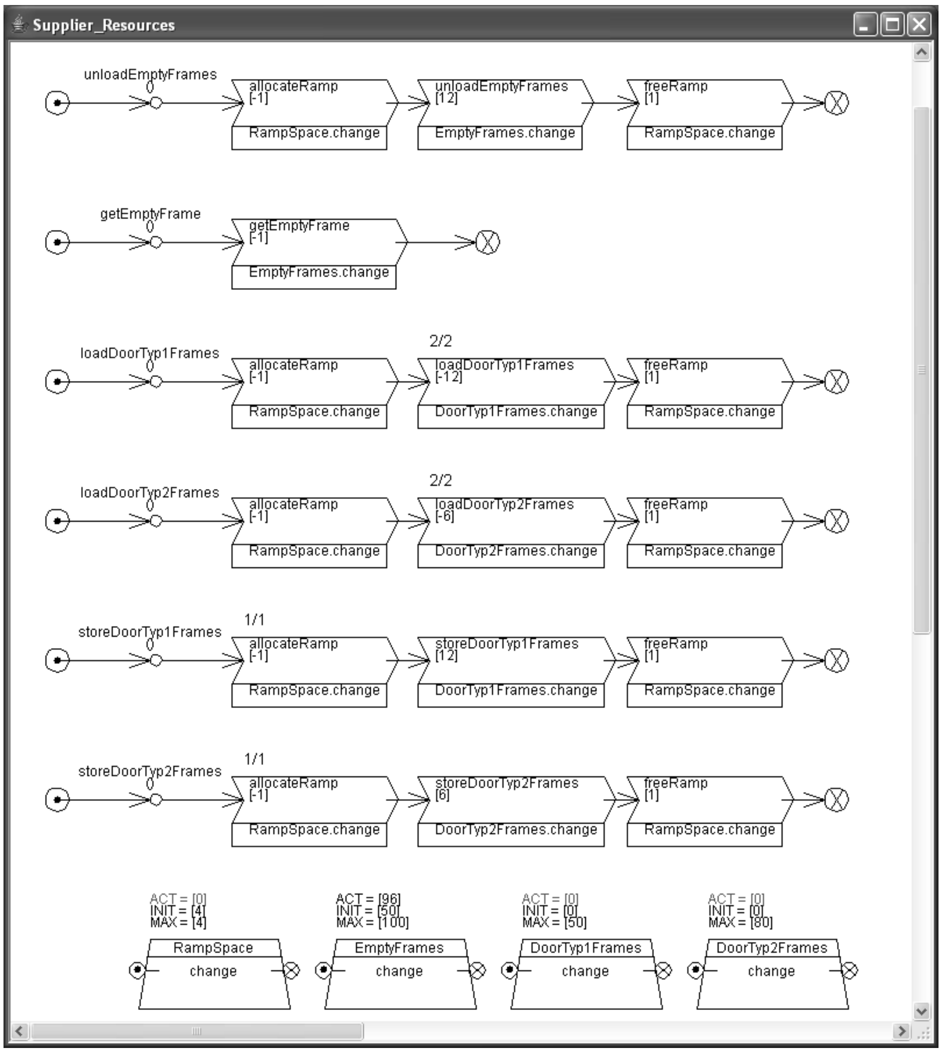
Figure 1: ProC/B model ‘Procurement_Channel’

or a counter. For instance ‘getDoorModule’, the activity after the PC-connector, calls a service ‘Resources.getTyp1Frames’ from FU ‘Resources’. The detailed description of that particular service is given in the same notation in the description of the corresponding FU.

For example, Fig. 2 shows 6 services offered by a FU 'Supplier_Resources' that contains 4 counter objects. Back to our example, any activity that performs a service call lasts until the call returns from that FU. The second process chain in Fig. 1 describes how cars with 4 doors are handled and finally process chain 'Trucks' describes the cyclic behavior of trucks. The value 4 at its source node indicates that 4 instances are created simultaneously and the corresponding point of time is given below, i.e., at time 0. Its first and last activity describe a loop. In between those activities, any instance of a truck process performs a 0.3 vs 0.7 random choice to follow the upper or lower branch at the OR-connector. This models a transport of door modules for cars with 2 doors vs 4 doors. Trucks pick up door modules at the supplier's site by activities that call services 'PickUpTyp1', resp. 'PickUpTyp2' and subsequently transport and unload at the manufacturer's site (services 'unloadTyp1Frames', resp. 'unloadTyp2Frames' of FU 'Resources'). A subsequent OR-connector describes that trucks what ever branch they come from subsequently perform 'loadEmptyFrames', which models that empty frames are transported from the manufacturer to the supplier. Hence the 'ENDLOOP' activity describes that the process should iterate and proceed at the aforegoing 'LOOP' activity.
For a given ProC/B model, the well-defined semantics makes it a discrete event system (DES) [4] such that we can perform analysis techniques known for DES. In particular, simulation is well appreciated by logistics experts. The ProC/B Toolset employs the HIT simulator [3] for simulative analysis of ProC/B models. However, simulation provides useful quantitative results, i.e., estimates performance measures, only if the given model is correct from a functional point of view. Functional deficiencies are not necessarily trivial to detect. For instance, Bause and Beilner discuss a model of a Freight Village in [2] that reaches a deadlock after several months of model time. Such faulty behavior can be but need not be detected by simulation, however its absence cannot be guaranteed.
As presented so far, simulation of the example model yields performance values that are alarming, since the throughput becomes zero after some time. To identify the reason for that behavior, we configure the simulator to write a trace into a file that documents each event performed in the simulation. Clearly, this full documentation provides sufficient information to detect the reason of that behavior; however the sheer amount of data makes it difficult to identify. This confronts a visualization aid with two challenges, namely to cope with the amount of data and to minimize the intellectual effort for a human being to identify the cause of any given behavior. In the next sections, we propose two different ways to visualize traces of a ProC/B model.

3 Animating the Dynamic Behavior of a ProC/B model

Since a modeler is familiar with the model and the notation used to describe the model, any form of animation of the same notation seems a straight forward and natural visualization. In ProC/B models, the current state of a model is given by accounting for which process is in which PCE. Despite detailed information on values of individual attributes and variables, we count the number of processes per PCE and per resource and visualize that number next to its corresponding ProC/B element. Given a representation of the current state, there are two ways to proceed. One can either automatically identify possible activities and let the user decide interactively with which element to proceed. This yields an interactive step-by-step procedure by which a user can investigate any potential dynamic behavior of the model. This technique mimicks what a debugger provides for the interactive analysis of program code. In the ProC/B Toolset, an interactive animation is currently supported and in addition, it also allows to proceed backwards in the dynamic behavior. Since timing constraints are not considered for manually, interactively created traces, the modeler might generate traces that may interfere with timing distributions assigned to events. However, manually investigating the dynamic behavior of a model clearly increases the modeler's understanding. Such traces can be stored for documentation and further analysis. Note that traces may come from many sources, e.g., simulators, model checkers that produce witnesses, but also measurement data taken from real systems that allows to generate traces.



Supplier_Resources
unloadEmptyFrames
0
allocateRamp [-1] RampSpace.change
unloadEmptyFrames [12] EmptyFrames.change
freeRamp [1] RampSpace.change
getEmptyFrame
0
getEmptyFrame [-1] EmptyFrames.change
loadDoorTyp1Frames
0
2/2
allocateRamp [-1] RampSpace.change
loadDoorTyp1Frames [-12] DoorTyp1Frames.change
freeRamp [1] RampSpace.change
loadDoorTyp2Frames
0
2/2
allocateRamp [-1] RampSpace.change
loadDoorTyp2Frames [-6] DoorTyp2Frames.change
freeRamp [1] RampSpace.change
storeDoorTyp1Frames
0
1/1
allocateRamp [-1] RampSpace.change
storeDoorTyp1Frames [12] DoorTyp1Frames.change
freeRamp [1] RampSpace.change
storeDoorTyp2Frames
0
1/1
allocateRamp [-1] RampSpace.change
storeDoorTyp2Frames [6] DoorTyp2Frames.change
freeRamp [1] RampSpace.change
ACT = [0] INIT = [4] MAX = [4]
RampSpace change
ACT = [96] INIT = [50] MAX = [100]
EmptyFrames change
ACT = [0] INIT = [0] MAX = [50]
DoorTyp1Frames change
ACT = [0] INIT = [0] MAX = [80]
DoorTyp2Frames change

Figure 2: Animated FU 'Supplier_Resources' - Partial deadlock situation

So alternatively and for a given trace, a user can follow that trace step by step. By help of that feature, we can animate and observe a trace generated by the simulator for a simulation run that shows zero throughput for the given model configuration. The recorded example trace contains more than 6900 steps for a simulated time interval of 1000 time units. In each step, the graphic representation of the ProC/B model is annotated with state information and PCEs are highlighted in red for blocked processes and in green for processes ready to proceed. Fig. 2 shows the animated figure for FU ‘Supplier_Resources‘ (but without colors for readability). That FU encapsulates 6 services which access 4 counters namely ‘RampSpace’, which describes the number of available ramps for trucks to unload empty frames and load door modules mounted on frames, ‘EmptyFrames’, which gives the number of empty frames currently on stock at the supplier’s site, ‘DoorTyp1Frames’ and ‘DoorTyp2Frames’ hold the number of frames that carry door modules of type 1, resp. 2. Values of counters have a finite domain, for instance ‘RampSpace’ takes values in $\{0, 1, \ldots, 4\}$. 0 is a predefined minimum, the maximum (max=4), the initial value (init=4) and the current value (act=0) in the current state of the trace are given above the counter in Fig. 2. In addition, the animation adds numerical values to the graphical model representation, e.g., the activity ‘ loadDoorTyp1Frames’ in service ‘loadDoorTyp1Frames’ carries value $2/2$ above. $a = 2/b = 2$ indicates that a processes are currently blocked at that activity, $b = 2$ processes reached that activity in total. The animation highlights blocked activities with $a > 0$ in red.
By help of this representation, we can identify the reason of the faulty behavior. Two processes of type ‘loadDoorTyp1Frames’ and two of type ‘loadDoorTyp2Frames’ have allocated all 4 available ramps. ‘RampSpace’ Act=0 indicates that there are no free ramps left. Those processes are blocked since their service calls to reduce counters ‘DoorTyp1Frames’ by 12, resp. ‘DoorTyp2Frames’ by 6 are blocked since the current values of those counters are 0. The processes that could resolve the situation by increasing the counter values are ‘storeDoorTyp1Frames’, or resp. ‘storeDoorTyp2Frames’, which are currently blocked themselves due to a lack of ramps. It is a classical deadlock with a circular waiting situation and simultaneous resource allocation; it is a partial deadlock due to source nodes of ‘OrderTyp1‘ and ‘OrderTyp2‘ that keep creating processes in Fig. 1. This implies that one needs to follow many steps from the beginning, possibly switching frequently through a set of FUs (especially for large models with a rich hierarchy and many FUs) to reach the state where the situation becomes visible. The higher the number of FUs, the more difficult it gets to retain an overview of the overall state that is distributed among a set of processes and a set of submodels / FUs. Alternatively, one can trace back from the last state of the model. However, this can be also cumbersome if source nodes create more and more processes that perform non-trivial activities but do not really cause the underlying problem as it is the case for ‘OrderTyp1‘ and ‘OrderTyp2‘ in our example. We remark, that the hierarchy of the model has been reengineered to allow for a concise graphical representation by 2 figures that allow us to explain how the deadlock occurs. In the original structure, the effect localized in Fig. 2 is distributed over two submodels.
Structure and abstraction is required to help a user gain insight in the model behavior. Animating the ProC/B model itself implies that the animation must follow the given hierarchical structure based on refinement of activities and containment of FUs. This may be but need not be the best way to structure a model of interacting processes for the visualization of its behavior. In the following section, we describe an alternative representation that employs Message Sequence Charts (MSCs) to follow a different point of view.

4 Visualizing Behavior by Message Sequence Charts

When tracking down particularities of the dynamic behavior of complex models, we believe it is natural that one wants to see how individual instances of processes proceed and how they interact with each other and available resources. MSCs [8] (or sequence diagrams in UML as a related formalism) set the focus on processes and their interactions and thus have the potential to visualize a ProC/B trace with respect to the afore going expectation. Formally, an MSC defines a labeled directed acyclic graph.

Definition 1 *An MSC M is given as a tuple $M = (V, <, P, M, K, T, N, m)$, where V is a finite set of events, $<\subseteq V \times V$ is an acyclic relation, P is a set of processes, M is a set of message names, $L : V \rightarrow P$ is a mapping that associates each event with a process, $K : V \rightarrow s, r, b, l$ is a mapping that describes the kind of each event as send, receive, broadcast or local, respectively, $N : V \rightarrow M$ maps every event to a name, $m = m_{sr} \cup m_b$ is a relation called matching with $m_{sr} \subseteq V \times V$ that pairs up send and receive events. Each send is paired up with exactly one receive and vice versa. Events v_1 and v_2 can be paired up with each other, only if $N(v_1) = N(v_2)$. $M_b \subseteq 2^V$ relates up to $|P|$ events that are broadcast events, i.e., that can be related if its elements are broadcast events.*

Figure 3 visualizes part of an example trace as an MSC. Each MSC process p is shown as a vertical line where events $v \in V$ with $L(v) = p$ are ordered from top to bottom according to the ordering relation $<$ defined for events. Send, receive and broadcast events result in horizontal lines that connect all processes that are involved. Sender and receivers are not distinguished for a broadcasting event. Fig. 3 shows that 'RampSpace' interacts with 'unloadEmptyFrames', 'loadDoorTyp1Frames', and 'loadDoorTyp2Frames', and that only 'unloadEmptyFrames' performs matching pairs of events that allocate and free ramps. The last four events connected to 'RampSpace' are all allocating events which results in the deadlock discussed before.
A trace of a ProC/B model contains the activities of instances of various process chains, their interactions and service calls to functional units, which allows to map a trace to an MSC in many ways. We use a mapping where each instance of a process chain as well as each service of a functional unit gives an individual process in the MSC. If a process chain calls a service of a functional unit, this results in a send event from the process for the instance of the process chain to that service process of the functional unit. If a service call terminates and returns, that service process of the functional unit does a send event with the process of the instance of the process chain as the receiving process. PCEs that do not formulate service calls become local events in an MSC process.
Obviously, the afore going construction results in a finite number of processes for any given finite trace. Nevertheless, the number of instances of process chains is expected to be high while the lifetime of a single instance is expected to be rather short compared to the length of the overall trace. Hence we seek to reduce the number of processes that are visualized by joining (merging) processes. As a first step, we join all instances of a single process chain into one process. In order to be able to distinguish among instances, we attach the identifiers (a numerical integer value) of each involved instance to its corresponding events. Identifiers are optionally visualized in the MSC with their corresponding events.
Traces tend to provide a lot of information and very detailed information on the dynamic behavior. Since there is no intelligent way to automatically identify the most appropriate level of aggregation for a human to understand the dynamic behavior, we must provide

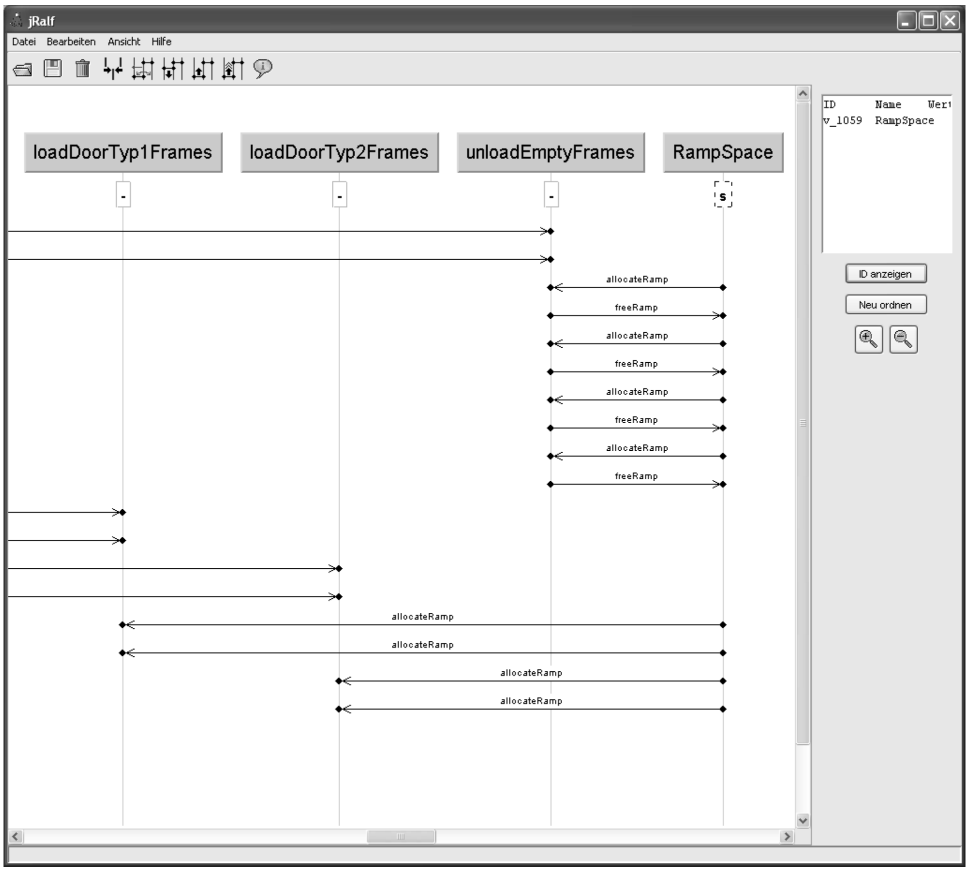


Figure 3: MSC of ProC/B trace

a number of operations to dynamically aggregate or refine the visualized amount of data taken from the trace. We propose the following set of operations.

Zooming and scrolling. The graphical representation of an MSC of a trace may easily exceed the available space of a display. Zooming and scrolling are conventional means to handle that problem, but without reduction of information.

Projections. Certain processes, events, or state information may be irrelevant for particular considerations, which in turn motivates projection operations that allow to remove a process, hide any individual event or hide all events with a certain name. These operations reduce the amount of visible information.

Highlighting and coloring. An operation that allows to assign a specific color to all events with a certain name or a state variables of a certain value is useful to make regular patterns visible for a human observer.

Merging of processes. Dynamically joining processes (with a corresponding split operation as inverse) into a single new process allows us to set focus on those processes that remain. This operation is important to reduce the number of visualized processes while retaining information on interactions among processes. In this way it differs from the projection operation. It allows each user to individually adapt the level of granularity with respect to the mapping of functional units and process chains to MSC processes.

Horizontal permutations of processes. Clearly, there are as many horizontal orders of processes as there are permutations of P. An automatic identification of those that are particularly useful for a human observer is impossible, some heuristics for ordering processes exist at best. A pragmatic way to solve this problem is to support manual reordering of processes, e.g., by simple drag and drop operations on the displayed process of the MSC.

Switching among partial and total order. Traces are typically generated such that the ordering relation $<$ on events is a total order, e.g., as generated by some discrete event simulator. However, if the simulation model consists of a set of asynchronous, interacting components (we avoid the term process at this point to avoid misunderstandings), a trace results from a partial order due to causal or logical dependencies among events as well as an order according to the point in time any event occurs. The given total order is present if the MSC is seen as a sequence of horizontal rows, where each row contains at most one local event or a pair of send and receive events or a set of broadcasting events that all match. If flow of information takes time, matching send/receive events may reside at different rows. However, one can for the sake of information aggregation, violate the total order and just stick to the partial order as given by the projection of events to all individual processes. The latter means in practice, that per row each process may show a local event or a send / receive or broadcast event. For the sake of clarity, a matching pair of send and receive, respectively matching broadcasting events should reside in the same row.

These operations are intended to support human perception and understanding of the dynamic behavior of a process oriented system that documents itself as a trace.

Implementation Issues. Note that visualization of traces by MSCs is rather independent from the ProC/B formalism. Consequently, we developed an XML file format for traces that consists of some header information that defines the set of processes P, state variables and names of events associated with each process as well as a static matching for any send, receive or broadcasting event. The trace as an ordered sequence of events (optionally an alternating ordered sequence of states and events) follows that header information [2].

The implementation provides read and write operation on traces in the XML format. A notable feature of the write operation is to write the events of a trace in an order that is only consistent with the partial order present in the representation and writes all local events ordered by processes up to the next send/receive or broadcasting event. If that trace is then used as input for an animation as discussed in Section 3, the positive effect is that there are no more switches among functional units and ProC/B submodels present than necessary which makes it easier for a human observer to follow the sequence of events. However, the derived trace may violate timing constraints present in the original ProC/B model.

5 Evaluation

In this section, we evaluate both concepts for a number of criteria.

Familiarity of notation. Assuming that a modeler knows a model well, a ProC/B visualization uses a known formalism for a known model. The MSC representation requires knowledge of a different notation, although MSCs or sequence diagrams are rather well

[2] Current work goes into make that file interface compatible to the MSC standard

known from software engineering.

State representation. The hierarchical decomposition of ProC/B implies a distributed representation of the current state of a model where each FU provides local state fragments of processes that currently use one of their services. As an undesired effect, a user needs to keep an overview over all FUs in mind which are distributed among canvases. This may easily exceed what one is able to memorize. In MSCs, the state of processes is distributed among the MSC processes and a global state is a cut which is a horizontal line in the simplest case, in general, it zig-zags through MSC processes without crossing send/receive or broadcasting events. Identification of cuts is relatively simple.

Monitoring individual processes. In the ProC/B animation, monitoring an individual process is straightforward only in interactive mode when a user decides on what event to follow. But even in that case, fork and join operations and the dynamic creation of subprocesses make it even difficult to follow a single process if it consists of a set of subprocesses that run in parallel. The ProC/B animation of a given trace - except for specially constructed cases - will switch frequently among all processes and will not show locality in case of a stochastic model. This makes it very hard to monitor individual processes. In the MSC representation, it is natural to follow the lifeline of a single process and consider its local events and interactions with other processes. In case of fork/join operations, events will remain in the same lifeline. In case of created subprocesses, there are more MSC processes, which however could be merged into a single process. If all instances of a PC are grouped into one MSC process as proposed in Section 4, individual processes are identified by corresponding labels of events. To monitor a particular process requires an appropriate highlighting or coloring of all events that belong to that process.

Monitoring sets of processes. This is hardly possible in the ProC/B animation since a set of processes is expected to spread over a large number of FUs in a complex model, while only few FUs can be visualized in parallel due to the limited space of a monitor screen. In MSCs, monitoring sets of processes can be arranged by adjusting the horizontal order of processes and by merging processes.

Adjusting the level of abstraction. Trace animation with a ProC/B model has to live with the given hierarchical structure present in the model. Assuming that a given structure was chosen to avoid overly large PC descriptions per FU, then resolving the hierarchy towards a flat model would a) create non-trivial layout problems and b) space problems for a given monitor size. Visualizing individual instances of PCs seems not feasible for non-trivial models in ProC/B. For MSCs, this is natural. Furthermore, by joining or splitting processes and in combination with projections it is straightforward to adjust the level of abstraction to a given trace.

Adjusting the layout. Except for trivial operators like zooming or scrolling, the ProC/B notation does not provide operations to adjust the layout to a given trace. In MSCs, it is straightforward to arrange processes that are currently in the focus and that interact by a manually performed horizontal permutation in the layout. In addition, projection and highlighting allows to clarify particularities even further. The option to relax the ordering of events, allows for a more condense representation of the partial order embedded in the total order of events in a given trace.

Interactive generation of traces. An interactive generation is natural in the ProC/B notation. MSCs do not provide the details of a model specification, with state variables, attributes of individual instances of PCs etc. Although it is conceptually possible, we believe that the gap between the original model formalism and MSCs is too wide to make an inter-

active mode work well with MSCs of ProC/B models. The main obstacle we see is that a user does not oversee further implications of local choices made for the next event to occur.
Overall evaluation. MSCs are more flexible in adjusting the level of abstraction and they illustrate more clearly the way individual processes or sets of processes proceed. The MSC representation has more advantages than the ProC/B animation for the analysis of a given trace, while the ProC/B animation is more helpful to explore the dynamic behavior of a model in an interactive manner. The interactive exploration is productive in an early stage of validation to identify simple errors. If trouble shooting suffices to consider a given state - as it is often the case for safety properties - the ProC/B state visualization is as useful as an MSC that includes state information. If errors are caused by faulty behavior that results from the interaction among processes and resources, the MSC representation is clearly superior. In summary, both visualization have strengths and weaknesses and seem complementary. For instance, writing an MSC trace according to its partial order into a new totally ordered sequence of events that shows more locality implies that a subsequent animation in ProC/B does not switch among FUs more than necessary (obviously, the timing constraints are not preserved by that operation).

6 Conclusion

In this paper, we propose two different visualization aids to help a user gain insight in the dynamic behavior of simulation models that consist of processes with interaction and resource allocation. We consider the ProC/B formalism as a particular modeling notation that is used to model complex and large networks of logistic systems. One visualization aid is a direct animation of the ProC/B model, either in an interactive manner or by animation of a given trace that is obtained from a simulator. In addition, we discuss a variant of Message Sequence Charts (MSCs) to visualize how individual processes proceed and how they interact. MSCs are amenable to a number of operations to adjust the level of abstraction and to improve on the visual representation to make it easier for a human observer to understand the dynamic behavior of a given model. Both approaches have been implemented and are evaluated by help of an example taken from logistics in a manufacturing environment.
Thanks. We acknowledge and thank two groups of students at TU Dresden who evaluated the XML interface and implemented the MSC visualization tool jRalf.

References

[1] F. Bause, H. Beilner, et al. The ProC/B Toolset for the Modelling and Analysis of Process Chains. *In T. Field, et al (eds.) Computer Performance Evaluation*, Springer, LNCS 2324, 2002.

[2] H. Beilner and F. Bause. Intrinsic problems in simulation of logistics networks. In Proc. *11th European Simulation Symposium and Exhibition (ESS99), Erlangen (Germany)*, October 1999.

[3] H. Beilner, J. Mäter and C. Wysocki. The Hierarchical Evaluation Tool HIT. Short Papers and Tool Descriptions, 7th Int. Conf. Modelling Techniques and Tools, Vienna (Austria), 1994.

[4] C.G. Cassandras and S. Safortune. Introduction to Discrete Event Systems. Kluwer, 1999.

[5] G.A. Geist, J. Kohl and P. Papadopoulos. Visualization, Debugging, and Performance in PVM. Proc. Visualization and Debugging Workshop (Oct. 1994), reprinted in Simmons, M.L. et al. *Debugging and Performance Tuning for Parallel Computing Systems*, IEEE, pp. 65-77, 1996.

[6] W.D. Kelton, R.P. Sandowski and D.A. Sandowski. Simulation with Arena. McGraw-Hill, 2002.

[7] A.W. Law and W.D. Kelton. Simulation Modeling and Analysis. McGraw-Hill, 2000.

[8] Message Sequence Charts (MSC'96). ITU-T Recommendation Z.120, 1996.