A Framework for Simulation Models of Service-Oriented Architectures

Falko Bause, Peter Buchholz, Jan Kriege, and Sebastian Vastag
Informatik IV, TU Dortmund
D-44221 Dortmund, Germany
{falko.bause,peter.buchholz,
jan.kriege,sebastian.vastag}@udo.edu

Author prepared version of a paper published in
LNCS 5119. Performance Evaluation: Metrics, Models and Benchmarks. Proc. of SPEC International Performance Evaluation Workshop (SIPEW 2008), Darmstadt, 2008.

http://dx.doi.org/10.1007/978-3-540-69814-2_14

A Framework for Simulation Models of Service-Oriented Architectures*

Falko Bause, Peter Buchholz, Jan Kriege, and Sebastian Vastag

Informatik IV, TU Dortmund

D-44221 Dortmund, Germany

{falko.bause,peter.buchholz,jan.kriege,sebastian.vastag}@udo.edu

Abstract

Service-Oriented Architectures (SOA) are one of the main paradigms for future software systems. Since these software systems are composed of a large number of different components it is non trivial to assure an adequate Quality of Service (QoS) of the overall system and performance analysis becomes an important issue. To consider performance issues early in the development process, a model based approach becomes necessary which has to be embedded into the development process of SOA to avoid overhead and assure consistency. In particular the specification of the software system should be used as a base for the resulting performance model. However, since common specification techniques for SOA are very high level, many details have to be added to come to an executable simulation model which is often needed for a detailed analysis of performance or dependability. This paper presents an approach which combines an extended version of process chains to describe the SOA components and some quantitative specifications at the higher levels. For the modelling of the detailed architecture and protocols the simulation tool *OMNeT++* is used. Both modelling levels are combined resulting in an executable simulation model for the whole architecture.

Keywords: Service-Oriented Architectures, Simulation, Process Chains, *OMNeT++*

1 Introduction

Service-Oriented Architectures (SOA) are one of the major paradigms to describe and realize complex software systems as they are required in todays IT-infrastructure. The description level of SOA is very high such that several details are hidden in the description and a loose coupling between different processes is assumed for a functioning system. Nevertheless, quantitative aspects like performance or dependability are major issues of SOA which are only partially addressed up to now. In general Service Level Agreements (SLAs) are negotiated between service provider (i.e. the SOA component) and the user which could be some business process or another component in a hierarchical SOA. This implies that the non-functional properties of SOA components are known and are considered during orchestration [11]. Both tasks are non trivial. Since

*This research was supported by the Deutsche Forschungsgemeinschaft as part of the Collaborative Research Center “Modelling of Large Logistics Networks”(559).

performance and dependability issues should be considered during the whole software design process [9], modeling is often the method of choice. However, the common description level of SOA is too high to allow a direct derivation of performance models. Thus, a common way is to manually derive an abstract performance model from a SOA specification and solve this model analytically [10, 13]. Although the use of simple analytically tractable models is often a good choice, in particular in the early design phases, these models reach their limits when detailed design decisions have to be made. In those situations simulation models are more adequate. However, it is known that the realization of simulation models is cumbersome and error prone. To obtain more reliable simulation models, the description language has to consider the specifics of the application domain [14] and predefined standard components have to be used whenever possible. For SOA this is a problem, if apart from the high level orchestration of services also details of the communication between services have to be described in the model. At the higher level, processes have to be described using the same graphical notation like process chains or one of the description languages like BPEL. Lower levels consisting of the protocols and resources required by the different operations cannot be adequately specified with these approaches. More appropriate at this level is the model world of some network simulator which on the other hand cannot be applied for the specification of SOA.

In this paper we propose a hybrid specification approach that combines the best of both worlds. On the one hand we use *ProC/B* [2] for the specification of SOA. *ProC/B* is a modeling paradigm to specify process chain models at a high level using a graphical interface. On the other hand we apply *OMNeT++* [12] for the specification of network resources used by the components of SOA for communication. The resulting model is mapped onto a C++ simulation model using the simulation kernel of *OMNeT++*. The approach has several advantages since it allows one to use the high level graphical description format of *ProC/B* and the predefined network components of *OMNeT++*. Furthermore, it results in relatively efficient simulation models (see also [3]) based on the simulation functionality of *OMNeT++*. Despite from an in principle very efficient simulator, the problem of large models and different time scales in large models remains and can only be resolved by choosing an appropriate abstraction level.

The combination of the high level view of *ProC/B* and the low level view of *OMNeT++* is not straightforward, especially if the models at the different levels should be kept as they are. In this paper, we propose an approach to combine both models by assigning remote service calls in *ProC/B* to message transfers in *OMNeT++*. This can be done extending existing models only slightly. One part of the extension involves annotations of the ProC/B model, similar to annotations of UML models for performance analysis [1, 16]. The other part of the extension concerns the network resources which are adapted to account for the annotations of the ProC/B model.

The paper is structured as follows. In the next two sections *ProC/B* and *OMNeT++* are briefly introduced. Section 4 presents the new concepts and constructs to combine both worlds. Then, in section 5, the approach is clarified by means of an example. The paper ends with the conclusions.

2 Process Chain Models

Various versions of process chains exist in the literature. Our work is based on a variant introduced by Kuhn ([7, 8]) which is used within the collaborative research center “Modelling of Large Logistics Networks” 559 (CRC 559;[5]) for modelling

and performance evaluation of logistics networks. Since process chains are a descriptive tool they do not allow one to derive simulation models automatically. In former times simulation models had to be build on their own without any formal relation to the process chain model, implying well-known problems of additional modelling effort or inconsistencies between the models. *ProC/B* [2] is an approach to diminish these problems by enhancing and stating the informal process chain description more precisely (cf. [4]). *ProC/B* captures the structure of a system in form of function units (FUs) and the behaviour by process chains (PCs). In *ProC/B*, FUs might offer services, which can be used by activities of process chains. Each service is again described by a process chain and can use services of internal or imported FUs, thus resulting in a hierarchical model description.

Fig. 1 shows the top level of an example of a *ProC/B* model. The model consists of a single PC, named `customer` and a single user-defined FU `Travel_agency`, which offers several services. In this example all services, except `submit_form`, require no input parameters and give no result values. A process chain element (PCE) of a PC might call a service of a FU by specifying the name of FU and service together with the necessary parameters. The mechanism is similar to most programming languages, e.g. the activity `submit_form` calls service `submit_form` of FU `Travel_agency` setting the formal parameter `travel_info` to `data.travel_info`[1].

The *ProC/B* model of Fig. 1 represents a use case of the "Web Services Architecture Usage Scenarios"[15] where customers contact a travel agency's Web site and ask for information on flights and hotel rooms, select a specific combination, which is then booked by the travel agency. PC `customer` in Fig. 1 directly visualises a customer's behaviour: First the customer requests a Web form, completes it and submits the filled form to the agency's Web site. The travel agent sends back a list of available flights from which the customer makes a choice. After submitting the option, the travel agent responds with a list of available accommodations, so that the customer again can make his choice. The travel agent finalises the transaction by booking flight and hotel room and charging the customer's credit card, which all is done by service `submit_accommodation_choice` within FU `Travel_agency`. The shown model describes the customer's activities at an abstract level and we only specified those parameters, which we thought of being relevant for performance evaluation. E.g., responses of the travel agent are not modelled, since we assume that replies are of similar size; the customer's request is classified into 3 categories (attribute `travel_info`) and initially set by random (see "`randint(1,3):INT`") and the customer activities at her local PC (filling out the form, choosing a flight etc.) are modelled by delays with a randomly chosen duration. Customers determine the load of the model specified according to exponentially distributed inter-arrival times with a mean of `3000` time units and might arrive single or in bulks of at most 3 (see "`randint(1,3)`"). We consider a time unit to be equal to 1 ms in our example.

The internals of FU `Travel_agency` are shown in Fig. 2. Each service is described by a PC and several other FUs offer services being used in activities of the travel agency. All FUs except FU `Travel_Agency_Server` are user-defined FUs and their internals can be specified analogously to FU `Travel_agency` (cf. Fig. 6). It is a matter of choice whether these FUs are modelled within or on the same level as FU `Travel_agency` using *ProC/B*'s capability of importing services (cf. [2]). The hierarchical model description ends at standard FUs which have a pre-defined behaviour,

[1] Access notations to parameters and variables of processes are prefixed with keyword *data* for technical reasons in order to distinguish them from global variables. Global variables are not shown in Fig. 1.

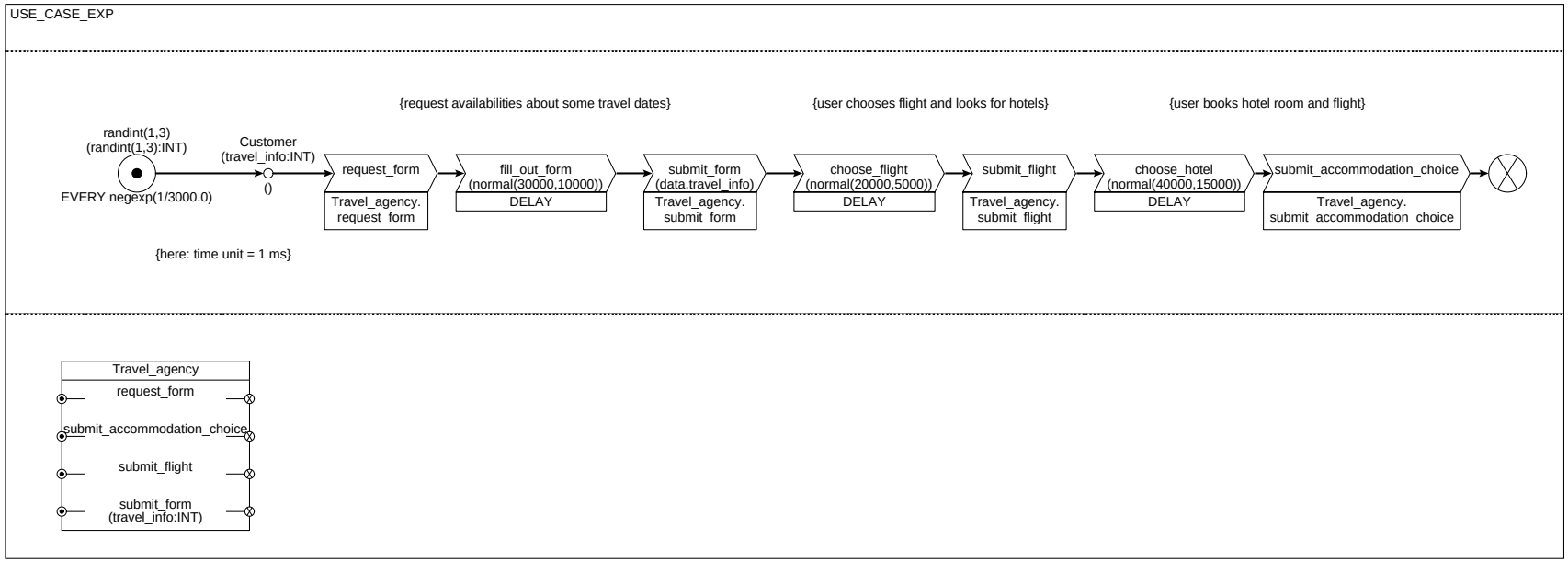


Figure 1: Example of a *ProC/B* model

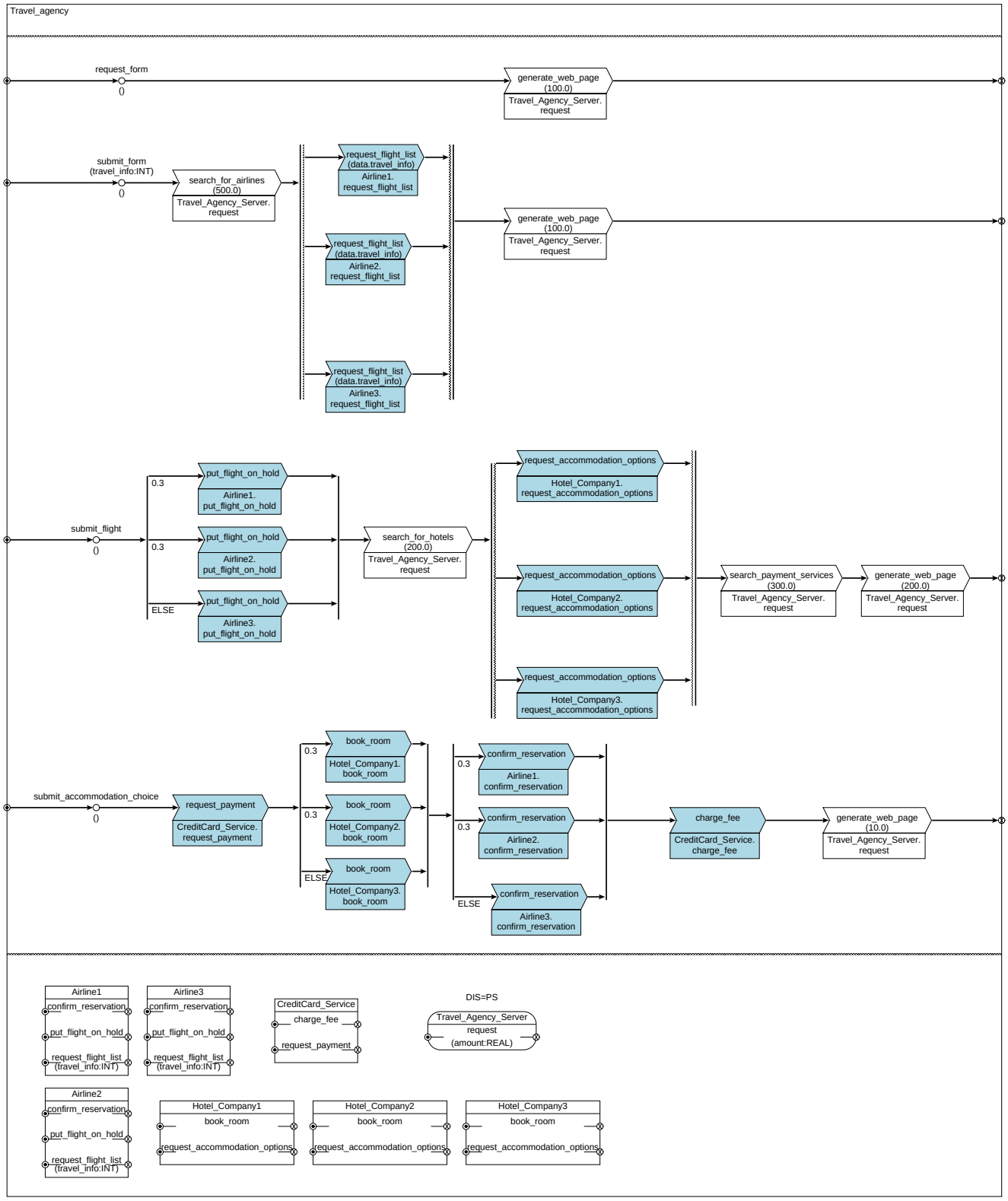


Figure 2: FU `Travel_agency`

like `Travel_Agency_Server`. *ProC/B* offers two kinds of standard FUs: servers act as traditional queues, and so-called counters support the manipulation of passive resources.

It is possible to specify *ProC/B* models precisely enough to obtain a simulation model for a performance analysis. In the course of the CRC 559 we developed a toolset which provides a graphical user interface to specify *ProC/B* models and transformer modules which map *ProC/B* models to the input languages of existing analysis tools, so that *ProC/B* models can be analysed automatically (cf. [2] and Fig. 3). Fig. 4 shows a possible result from a simulation run. The diagram shows the average response time of a customer request to FU `Travel_agency`. The *ProC/B* toolset also offers the possibility to get similar results for each specific activity of PC `customer`. We do not want to go into those details of the model now and refer the reader to [2]. Instead we consider refinements of *ProC/B* and the model in order to capture the system's behaviour more accurately.

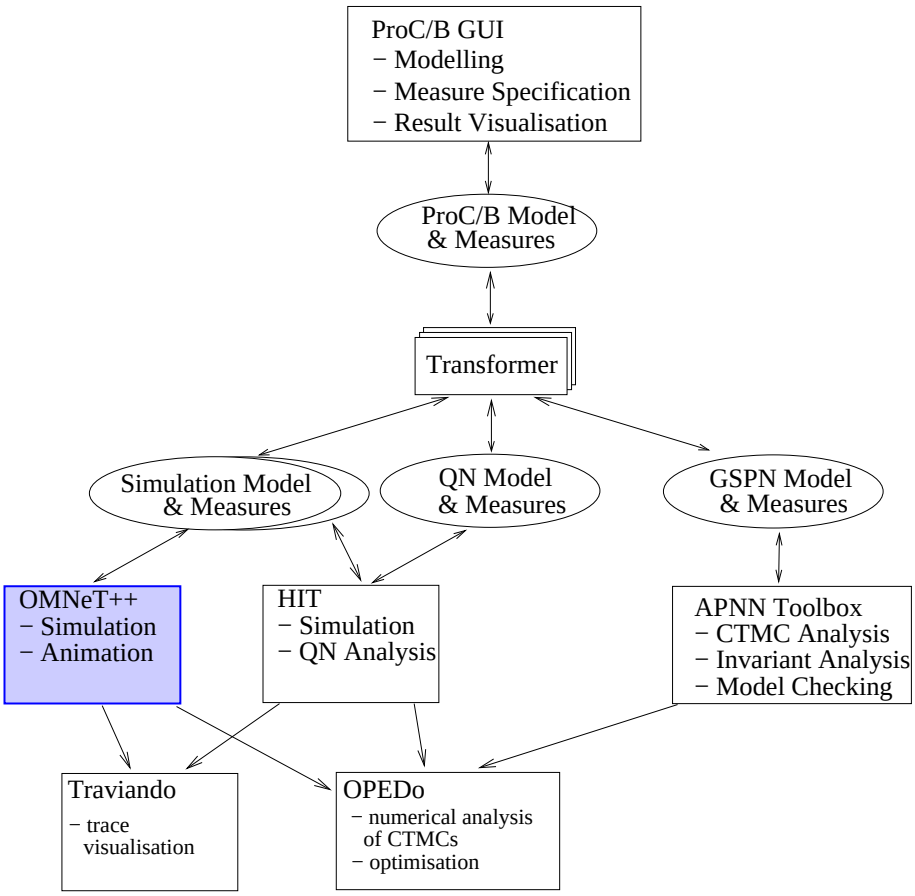


Figure 3: *ProC/B* toolset

In many practically relevant applications timers and timeouts are used, e.g. in operating systems or network protocols. Timers can also be used to model system characteristics on higher levels, like e.g. for the behaviour specification of a customer. Up to now timeouts were not considered in process chain models. Recently we extended *ProC/B* by a timer construct as follows.

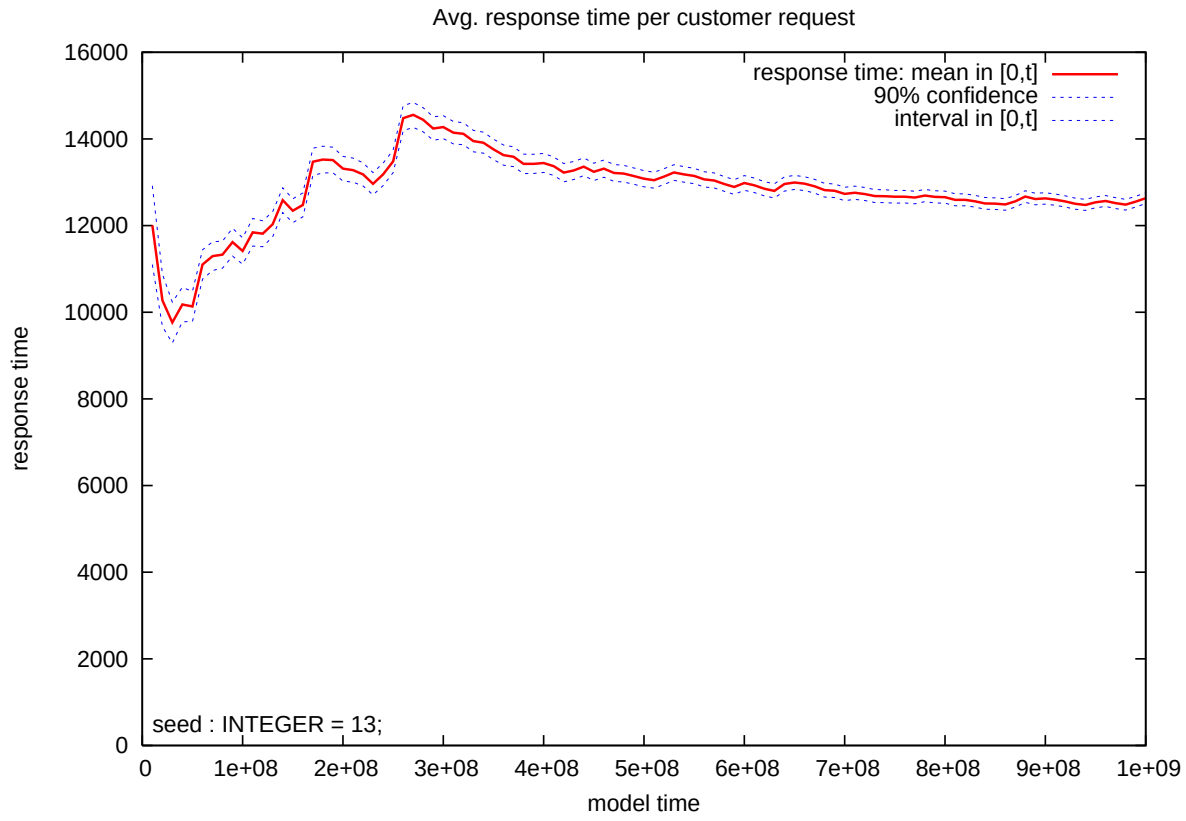


Figure 4: A possible simulation result

In Fig. 5 we use the specification of timeouts for modelling the impatience of Web customers. When requesting a form calling service `request_form` of FU `Travel_agency`, a customer sets his individual timer with a timeout value of `3000` time units. At latest after 3000 time units he will receive a boolean value stored here in variable `data.in_time` as the answer. The (informal) semantics of this syntactical construct is that at the same time as the timer has been set the specified service is called. If the service call returns on time, the timer is deleted and the process proceeds with a true boolean value for the specified variable, i.e. with `data.in_time = TRUE` in the example. If the timer expires before the service call returns, the specified variable is set to FALSE and the process proceeds immediately. Other possibly user-defined result values are set to default values. A service call returning after the timer has elapsed will be ignored. In the example, the behaviour of a customer depends on the value of the variable `data.in_time` and possible other result values of the service call. In Fig. 5 an expired timer implies counting this incident in the user-defined result measure `lost_customers` and it implies the termination of the customer process.

Normally, an initiated service call will be executed until it ends, as specified by the PC. The user can have influence on this behaviour by setting the pre-defined process local variable `CANCEL_ALLOWED`, see Fig. 6. After executing the activity specified by a PCE, a process checks whether the corresponding timer, if available, has expired and whether its boolean variable `CANCEL_ALLOWED` is true. If both conditions are met, the process terminates immediately. This construct allows also the modelling of lower level network mechanisms at an abstract level including the saving of resource capacities.

In Sect. 5 we will revisit this example and show some more details.

As indicated by Fig. 3 we also integrated a mapping to *OMNeT++* [3] into the *ProC/B* toolset. This does not only allow us to benefit from the features of a modern object-oriented simulator but also offers the possibility to use existing *OMNeT++* frameworks for modelling communication aspects. Sect. 4 describes the use of the INET framework for considering network aspects in performance models of service-oriented architectures.

3 Modelling Networks and Protocol Stacks

OMNeT++ [6] is a public-source discrete event simulation environment, that has been developed and used extensively for the modelling of communication protocols. Additionally, it has been proved suitable in other application areas as well (cf. [3]). *OMNeT++* models are composed of modules which can be simple or compound. The module interfaces and their relationships are described with *OMNeT++*'s NED language. While simple modules are implemented as a combination of NED files and C++ classes, compound modules, that may consist of other simple and compound modules, are only described by NED files. Modules are connected via gates and can communicate by messages either sent along connections or sent directly to the destination module.

There are several simulation model frameworks available for *OMNeT++* especially for building network models. One of these frameworks is the INET Framework including (among others) protocol implementations of IPv4, IPv6, TCP, UDP, Ethernet and 802.11. These protocols are represented by simple modules and can be combined to compound modules to form network hosts. Several assembled compound modules that implement routers, switches etc. are already included. Additionally, modules for network interfaces, routing tables or the auto-configuration of a network are provided. Some of

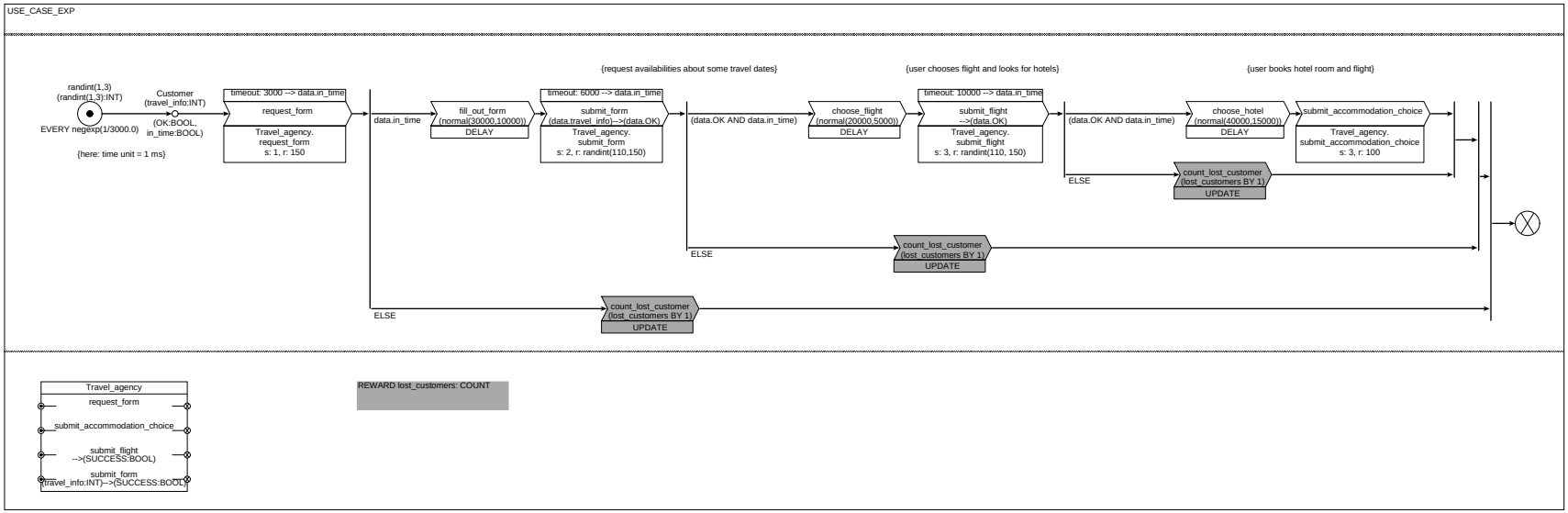


Figure 5: *ProC/B* model with timeout definitions

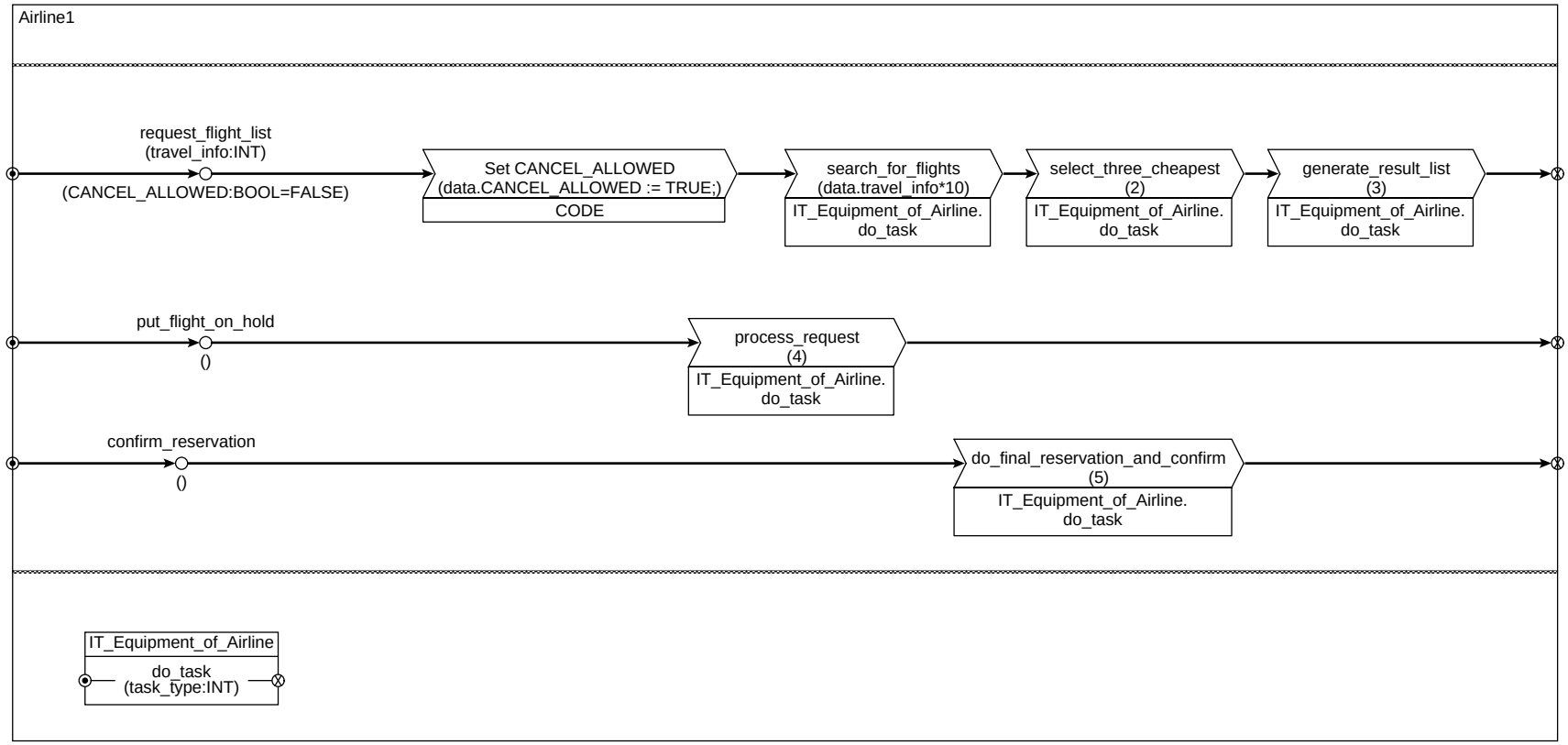


Figure 6: Supporting cancellation of “expired” processes

those modules are part of almost every host like *RoutingTable*, which can be used for querying, adding or deleting routes, or *InterfaceTable*, which contains a list of all interfaces of a host. Other modules are only instantiated once, like for example the *FlatNetworkConfigurator*, that can be used to assign IP addresses to hosts.

The communication between network layers is realized by messages between the modules representing those layers. An upper layer protocol may send a message representing a data packet to the next lower layer (linked with some additional information to determine the destination of the packet like an IP address), which will encapsulate and forward the data. Receiving packets works in a similar way.

In the next section we describe how *ProC/B* and the INET Framework can be combined to simulate service-oriented architectures. This approach uses an INET model of the network topology, while a *ProC/B* model is used to specify the activities and the process flow.

4 Combining Both Worlds

In *ProC/B* calls from PCEs to FUs are instantaneous. No model time is consumed between the start of a service call and the start of its execution. In the *ProC/B* paradigm this is a reasonable assumption, since process chains and function units are used to separate behaviour from structure descriptions and it is implicitly assumed that services run on the same hardware or communication needs negligible time. In the Web services example of Sect. 2 some PCEs represent access to some remote component, e.g., the query of airline databases for availability of passenger tickets. Typically, a service call to a remote component requires apart from the processing time at the remote component also time for communication which, depending on the communication medium and the distance, might take a large percentage of the overall time and in particular can have some jitter. In Fig. 2 all calls to remote sites are indicated in blue.

Communication between remote sides is usually realized via network components that can be adequately modelled using the modules from the INET framework. Typical INET models represent some network and specify the load generation in host modules employing different pre-defined modules of the INET framework. The main idea to combine *ProC/B* and INET is to define a mapping between all FUs and some hosts of an INET model. PCEs are implicitly related to the hosts of their surrounding constructed FU. The mapping need neither be injective nor surjective. Whenever a PCE specified within an FU A calls a service of an FU B mapped to a different host, messages between the two hosts associated with A and B are exchanged in the INET model. E.g., one can define that all activities of FU `Travel_agency` are performed on host 1, those of FU `Airline1` on host 2, those of FU `CreditCard_Service` on host 3 etc. (cf. Fig. 7) or one can define a mapping onto 3 hosts as we have done in Sect. 5. Surely, the INET model has to be prepared appropriately in order to be simulated in conjunction with the *ProC/B* model. In principle this means that we keep two models, the *ProC/B* model and the INET model. Both are enhanced by some constructs to realize the combination into a single simulation model.

In order to account for network traffic, we enhanced the *ProC/B* description by some information on the amount of data, which needs to be sent. Fig. 8 shows a part of FU `Travel_agency` with additional attributes for PCEs doing a remote service call. Since a service call in *ProC/B* has two directions, calling the service and receiving the result, also two attributes are specified, one for the send direction (`s:`) indicating the amount of data being sent to the FU/host and one for the receive

direction (`r:`) indicating the amount of data being sent back from the FU/host. As mentioned, whenever a remote service call is initiated a message from the *ProC/B* part is sent to the INET part of the *OMNeT++* model and received by modified host modules. They replace similar host models in the INET model to interface with the *ProC/B* model part and lack the random traffic generators for TCP and UDP. Instead they include a new module on application layer called `ProCBApp` which interfaces the process model and the INET network part by translating messages with set `s:` or `r:` parameters into TCP data transmissions with requested bytelength.

The TCP transmission delay for a *ProC/B* remote service call is determined as follows: The initial *ProC/B*-message indicating a PCE $\rightarrow$ FU call is suspended at the PCE, instead a message is sent to the corresponding modified host module (see red arrows in Fig. 9). The host will initiate a TCP connection and transmit the amount of data specified for the request. The called peer will close the connection after the transmission. When the connection reset packet arrives at the first host, the simulation clock has progressed for the amount of time a data transmission including connection handling would take. A signal message is directly sent back to the PCE and the waiting call is released to continue to the target FU in zero time. Similarly the result of a service call is returned: After the last activity of a service call has been performed, a message is sent to the host module of the corresponding FU and the message is transferred to the INET part of the *OMNeT++* model. Once the message arrived back at the originally sending host, it is sent to the calling PCE, so that activities within the *ProC/B* part of the model can proceed.

Holding the service call message back and using a replacement message in the INET model has two important advantages: First, the INET does not transmit information, it transmits the bytelength of the information. Transmitting the original service call with INET's simulated IP stack would require more complex input or output parameters at the service call. Second, there is no interference with process statistics, since a *ProC/B* process is still either in the PCE or in the called FU. By suspending process messages in the domain of a *ProC/B* element for the time of transmission statistics are kept consistent.

By including the INET framework in *ProC/B* models service calls can be delayed by realistic values resulting from connection speed and network topology given by the INET model. More than that, even network bottlenecks or message losses due to overload conditions become relevant for FU calls if there are many calls at the same time. As described above, each FU is bound to a network host. In contrast to network elements, FUs are organised hierarchically and can include each other as components. A call in FU A to an included FU B thus induces a network communication between host X and Y of the INET model, provided the FUs have been mapped to the hosts X and Y. Rather than sending messages from PCEs to FUs as service calls, a TCP transmission is prepared in host X and given to the INET part of the model. The simulated IP-Stack will resolve the target address, establish a connection to the target host Y and transmit the message. All network activities consume some time, typically in range of milliseconds. When the called Function Unit has finished the service call, the network is used again to signal the end of service and again some network latency occurs.

Following this approach, the use of INET models as network topology for *ProC/B* is supported. This allows the modeller to work with a fine-grained network description to evaluate effects of network hardware and bottlenecks to application processes specified as *ProC/B* activities. Currently the INET model has to be adjusted by hand in order to be used together with the *OMNeT++* translation of the *ProC/B* model. But we intend to automate the combination of *ProC/B* and INET in such a form that for INET models with appropriate host definitions only the mapping of FUs to hosts has to be specified.

5 Application Example

In the following we present the model from Sect. 2 in more detail and show some additional modelling constructs that enable the use of *OMNeT++* and the INET framework for *ProC/B* models.

As already mentioned, our model addresses a Web service usage scenario described informally in [15]. It consists of several FUs representing servers that offer Web services and PCs that are used to model the behaviour of those Web services and the behaviour of customers accessing the travel agency Web service.

To reduce complexity we did not include some directory service like UDDI in our model and assume that the Web service discovery has already taken place beforehand. However, from a modelling point of view also those services could be integrated.

In this scenario Web services are offered by a travel agency, three airlines, three hotel companies and a credit card service. The FU `Travel_agency` offers four services that can be used by the PC `Customer` to request the availability of hotel rooms and flights and to book them eventually. The PC `Customer` and the FU `Travel_agency` are shown in Fig. 5. The behaviour of the PC has already been described in Sect. 2. For the use of the model with *OMNeT++* and the INET framework additional attributes can be specified for service calls that comprise messages sent over a network. We use the timeout mechanism described in Sect. 2 to model the customer behaviour. We assume that a customer will wait 3 seconds for the input form of the travel agency to show up. In later steps of the process he or she accepts to wait 6 and 10 seconds for the availability of his travel dates and the list of hotels, respectively. Additionally, the modeller can specify the amount of data that has to be transmitted and that will be received when sending over a network. These amounts may be a fixed number or drawn from a probability distribution. For example for the call of PC `Customer` to the service `submit_form` of `Travel_agency` a fixed number of 2 KB has to be transmitted. The data returned may vary between 110 and 150 KB, since in reality it will depend on the number of available flights that the travel agency has found. The amounts of data are summarised in table 1. At this abstract level the modeller does not have to specify the actual contents of the messages, only a message size is required to model the delay for sending the message over a network. In fact the messages sent in this model may even be of different types: While the communication between the customer and the travel agency is made up of simple HTTP requests and responses for accessing several websites, the travel agency and the airlines exchange SOAP messages. However, when calling a service additional parameters may be passed, so that activities of that service may depend on these parameters. While the network latency only depends on the message size and not on the actual content, additional delay may be caused by processing the messages, e.g. marshalling and unmarshalling of XML-based messages may take up some CPU resources. The latter delay has been omitted in our example, though it can easily be modeled by additional servers, that are accessed whenever a message needs to be processed.

The inner view of the FU `Travel_agency` is shown in Fig. 2. The FU has been extended by some additional modelling constructs for the simulation with *OMNeT++* as one can see in Fig. 8 and as described in the following.

Each of the four services is modelled by a PC. Additionally, it contains further FUs for hotel companies, airlines and the credit card service and a server that is accessed when generating the websites that are delivered to the customer.

The simplest service, `request_form`, will just generate the initial website for a customer by an access to the server.

Service `submit_form` (see Fig. 8) is invoked after a customer has entered date and destination of his travel and returns a list of possible flights to the customer. It makes use of the variable `CANCEL_ALLOWED` that has already been explained in Sect. 2, and thus the PC can be interrupted when a timeout has occurred. The service looks up eligible airlines in its local directory, sends messages to the airlines and receives flight dates afterwards. We assume that `Airline1` is a large airline and returns a longer list of flights than the other airlines as one can see from table 1. All calls to the airlines make use of the timeout mechanism again. If all three airlines fail to deliver any flight information within 3 seconds the travel agency cannot serve the customer's request and will return the boolean variable `SUCCESS` set to `false` finally resulting in a loss of the customer. If at least one of the airlines returns the flight options in time this variable is set to `true` and the PC `Customer` will continue with the next step. The service `submit_flight` is invoked after a customer has chosen his flight. The service needs to contact an airline to put the flight on hold and request accommodation options from the hotel companies. The former is done by sending a message to one of the airlines, while the latter is modelled in a similar manner as the compilation of possible flights in the service `submit_form`. First the hotel companies are looked up in a local database and after that messages are sent to them (again using the timeout mechanism). For the final step in the booking process the service `submit_accommodation_choice` is invoked. This service first contacts the credit card service to negotiate payment options. After that a hotel company is contacted again to book a specific hotel, the reservation of the flight is confirmed and finally the credit card service is contacted again to charge the fee. The booking process is completed after a website is generated for the customer summarising the travel plan.

Modelling of the FUs for the airlines, hotel companies and the credit card service is less complex, since no further remote services are invoked from there. Each of the FUs for the airlines contains a server with discipline processor sharing, that is used for modelling the IT equipment of the airline. Most of the tasks like searching for flights, generating result lists and reserving flights are performed by an access to the server. The three airlines only differ in the capacity of the server. The inner view of one of the FUs for the airlines is shown in Fig. 6. A similar situation holds for the FUs that represent the hotel companies: Their services are modelled by one or more accesses to processor sharing servers (that have a different speed for each of the hotel companies) as well. Finally requests to the services of the credit card company are only delayed for an uniformly distributed duration.

As already mentioned, customers might leave the website of the travel agency when the time they are willing to wait for a response is exceeded. Additionally, the results from some hotel companies and airlines might be ignored when the travel agency service assembles the result list, if those results are not delivered before a timeout has occurred. *ProC/B* offers the possibility to specify measurements [2], called rewards, at any FU. When simulating *OMNeT++* will estimate results for those rewards [3]. Apart from standard rewards like throughput, response time or the population, *ProC/B* allows for the specification of user-defined rewards. As one can see in Fig. 5 a reward has been defined to estimate the mean number of lost customers. Further rewards are used to estimate the mean number of hotel companies and airlines that did not respond in time (see Fig. 8).

For simulation the *ProC/B* model has been combined with the FlatNet model (cf. Fig. 9), which is one of the standard models that are part of the INET Framework. Next to the host for customers, travel agencies and airlines share a server in our mapping. Hotel companies where separated from the booking process to a dedicated server. Locating services to different

machines in the INET network model requires data communication for each service call between PCEs and distant FUs.

Results of some simulation runs are shown in table 2. Two model parameters are varied here: inter-arrival times of new customers and the transmission delay on cable lines between two routers of the INET model. Remember that customers might arrive in bulks.

The first value of each block is the number of lost customers per second from which we calculated the relative loss. The effects of intense customer arrivals are clearly visible in increased response times of the travel agencies booking system resulting in higher customer losses. Surely, the reason is that the database systems inside the model are slowed down by the increasing number of simultaneous requests. If communication network latencies are increased, many user requests that have been in time before become late. The two rightmost columns indicate the line between significant loss of customers and the complete failure of service.

6 Conclusions

In this paper we presented an approach supporting modelling of service-oriented architectures also accounting for lower level network operations. Web services and their orchestration are described on a higher level using a process chain-like description (*ProC/B*) and lower network activities are modelled using (possibly available models of) the INET framework. As a matter of course the combination of *ProC/B* models for Web services and INET models for networks seems not always appropriate due to the different time scales, but the presented approach gives at least the principal possibility to validate this assumption.

Currently we have to adjust INET models by hand for being used together with *ProC/B* models, but we head for an automated support for appropriate INET models.

So far only synchronous communication has been considered. Future research is directed to support also asynchronous communication by extension of *ProC/B*.

References

[1] S. Balsamo and M. Marzolla. Performance evaluation of UML software architectures with multiclass Queueing Network models. In *WOSP '05: Proceedings of the 5th international workshop on Software and performance*, pages 37–42, New York, NY, USA, 2005. ACM.

[2] F. Bause, H. Beilner, M. Fischer, P. Kemper, and M. Völker. The ProC/B Toolset for the Modelling and Analysis of Process Chains. In T. Field, P. G. Harrison, J. T. Bradley, and U. Harder, editors, *Computer Performance Evaluation / TOOLS*, volume 2324 of *Lecture Notes in Computer Science*, pages 51–70. Springer, 2002.

[3] F. Bause, P. Buchholz, J. Kriege, and S. Vastag. Simulating Process Chain Models with OMNeT++. In *Proc. of 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTools)*, 2008.

[4] F. Bause, P. Buchholz, and C. Tepper. The ProC/B-approach: From Informal Descriptions to Formal Models. In *ISoLA - 1st International Symposium on Leveraging Applications of Formal Method*, Paphos, Cyprus, 2004.

[5] Collaborative Research Center 559 “Modelling of Large Logistics Networks”. http://www.sfb559.uni-dortmund.de.

[6] R. Hornig and A. Varga. An Overview of the OMNeT++ Simulation Environment. In *Proc. of 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTools)*, 2008.

[7] A. Kuhn. *Prozessketten in der Logistik - Entwicklungstrends und Umsetzungsstrategien*. Verlag Praxiswissen, Dortmund, 1995.

[8] A. Kuhn. *Prozesskettenmanagement - Erfolgsbeispiele aus der Praxis*. Verlag Praxiswissen, Dortmund, 1999.

[9] D.A. Menascé, V.A.F. Almeida, and L.W. Dowdy. *Performance by Design*. Prentice Hall, 2004.

[10] D.A. Menascé, H. Ruan, and H. Gomaa. QoS Management in Service-oriented Architectures. *Perform. Eval.*, 64(7-8):646–663, 2007.

[11] V. Muthusamy, H.A. Jacobsen, P. Coulthard, A. Chan, J. Waterhouse, and E. Litani. SLA-driven Business Process Management in SOA. In Kelly A. Lyons and Christian Couturier, editors, *CASCON*, pages 264–267. IBM, 2007.

[12] OMNeT++ Community Side. URL:http//www.omnetpp.org/.

[13] D. Rud, A. Schmietendorf, and R.R. Dumke. Performance Modeling of WS-BPEL-Based Web Service Compositions. In *SCW*, pages 140–147. IEEE Computer Society, 2006.

[14] W.T. Tsai, Z. Cao, X. Wei, R. Paul, Q. Huang, and X. Sun. Modeling and Simulation in Service-Oriented Software Development. *Simulation*, 83(1):7–32, 2007.

[15] Web Services Architecture Usage Scenarios, 2004. URL:http://www.w3.org/TR/2004/NOTE-ws-arch-scenarios-20040211/.

[16] C. M. Woodside. From Annotated Software Designs (UML SPT/MARTE) to Model Formalisms. In Marco Bernardo and Jane Hillston, editors, *SFM*, volume 4486 of *Lecture Notes in Computer Science*, pages 429–467. Springer, 2007.

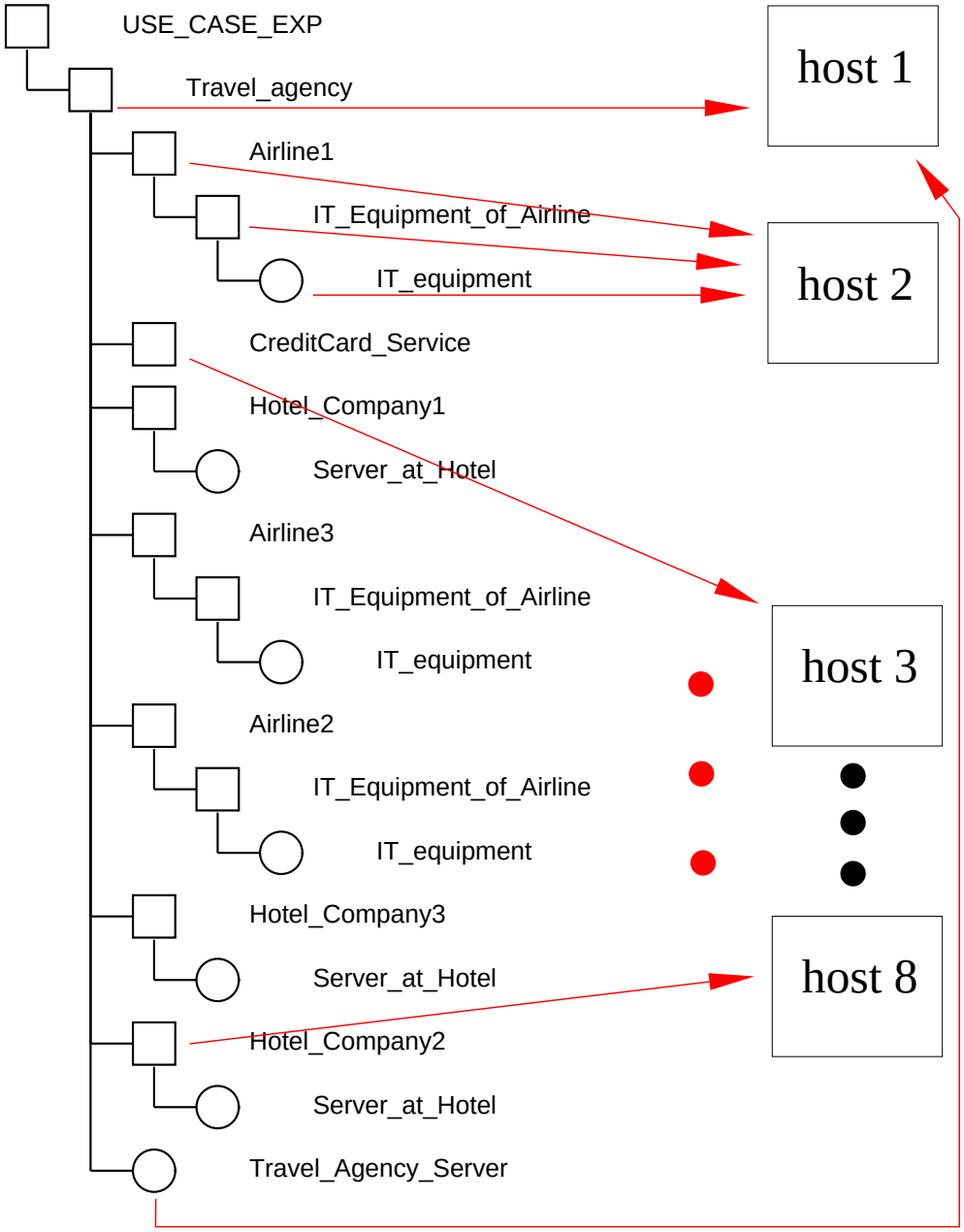


Figure 7: Mapping FUs to hosts

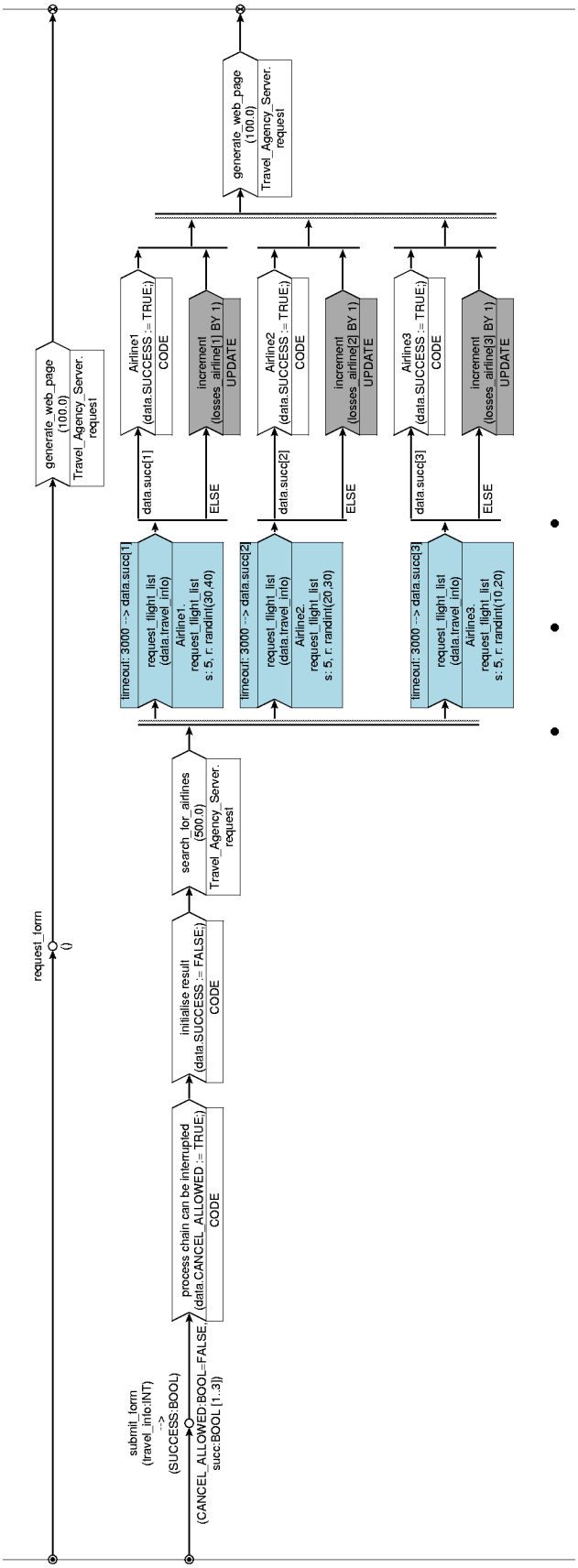
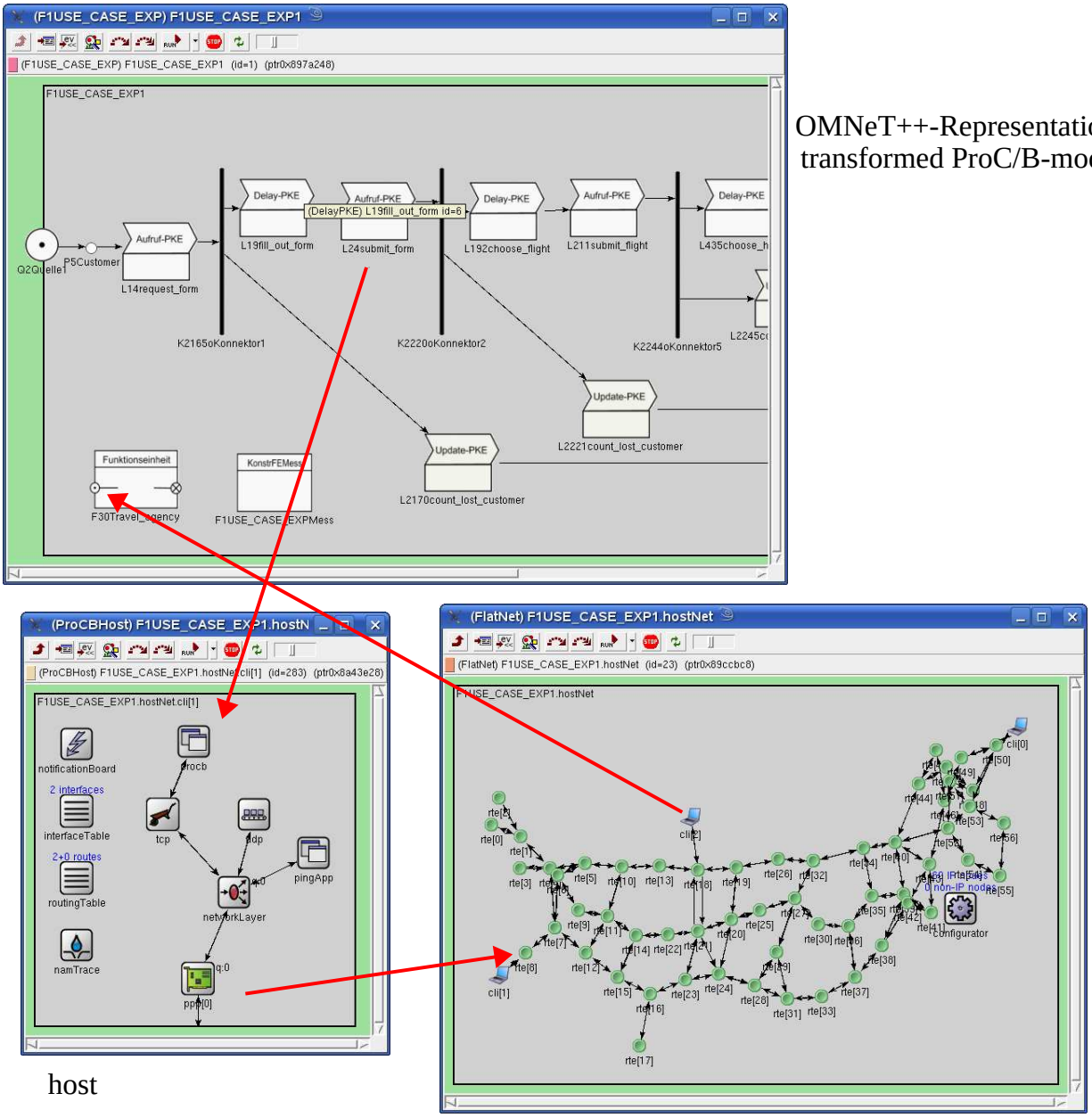


Figure 8: Part of FU `Travel_agency` with send/receive attributes (here in KB) for INET



OMNeT++-Representation of transformed ProC/B-model

host

INET model

Figure 9: Message flow to an FU when calling a service

Table 1: Amount of data sent between different hosts of the model

Source	Destination	Data (KB)	
		send	receive
Customer	Travel_agency.request_form	1	150
	Travel_agency.submit_form	2	110-150
	Travel_agency.submit_flight	3	110-150
	Travel_agency.submit_accommodation_choice	3	100
Travel_agency.submit_form	Airline1.request_flight_list	5	30-40
	Airline2.request_flight_list	5	20-30
	Airline3.request_flight_list	5	10-20
Travel_agency.submit_flight	Airline1.put_flight_on_hold	2	1
	Airline2.put_flight_on_hold	2	1
	Airline3.put_flight_on_hold	2	1
	Hotel_Company1.request_accommodation_options	5	10
	Hotel_Company2.request_accommodation_options	5	10
	Hotel_Company3.request_accommodation_options	5	10
Travel_agency.submit_accommodation_choice	CreditCard_Service.request_payment	2	3
	Hotel_Company1.book_room	2	2
	Hotel_Company2.book_room	2	2
	Hotel_Company3.book_room	2	2
	Airline1.confirm_reservation	2	2
	Airline2.confirm_reservation	2	2
	Airline3.confirm_reservation	2	2
	CreditCard_Service.charge_fee	2	2

Table 2: Lost customers per second (10000 seconds model time).

mean inter-arrival time (sec.)	network delay	0.001s	0.01s	0.05s	0.075s	0.1s
4	lost customers per sec.	0.0650	0.0687	0.1318	0.1879	0.4976
	standard deviation	0.3517	0.3560	0.4982	0.5538	0.7371
	confidence 90%	20.00%	13.21%	16.63%	11,27%	9.76%
	relative loss	13.0%	13.7%	26.4%	37.6%	99.5%
3	lost customers per sec.	0.2007	0.1965	0.2768	0.3754	0.6708
	standard deviation	0.6110	0.6040	0.7003	0.7922	0.8965
	confidence 90%	10.25%	6.30%	10.27%	8.39%	4.47%
	relative loss	30.1%	29.5%	41.5%	56.3%	100%
2	lost customers per sec.	0.5903	0.6245	0.6795	0.7432	0.9856
	standard deviation	1.0100	1.0693	1.1395	1.1632	1.2273
	confidence 90%	10.00%	10.88%	5.19%	10.24%	3.69%
	relative loss	59.0%	62.5%	68.0%	74.3%	98.6%