Safety and Response-Time Analysis of an Automotive Accident Assistance Service

Ashok Argent-Katwala[1], Allan Clark[2], Howard Foster[1], Stephen Gilmore[2], Philip Mayer[3], and Mirco Tribastone[2]

[1] Imperial College, London, England
[2] The University of Edinburgh, Scotland
[3] Ludwig-Maximilians-Universität, Munich, Germany

Abstract. In the present paper we assess both the safety properties and the response-time profile of a subscription service which provides medical assistance to drivers who are injured in vehicular collisions. We use both timed and untimed process calculi cooperatively to perform the required analysis. The formal analysis tools used are hosted on a high-level modelling platform with support for scripting and orchestration which enables users to build custom analysis processes from the general-purpose analysers which are hosted as services on the platform.

1 Introduction

Service providers who sell services which are concerned with human health and human safety have a responsibility to assess the quality of the service which they provide in terms of both its correctness of function and its speed of response. One way to carry out such an assessment is to construct a precise formal model of the service and perform the analysis on the model to shed light on the behaviour of the service itself. Such an assessment exercises the ability to apply both qualitative methods (such as model-checking) and quantitative methods (such as transient analysis) in service evaluation. The service providers delivering these critical services may not themselves have the technical skills to apply methods such as these. Further, even if they are able to source the necessary skills from expert users elsewhere, they may not be happy to take advantage of this because they would then risk revealing information about their current service provision which they might be unwilling to disclose to anyone outside their organisation.

One possible way in which the stakeholders of formal analysis methods can contribute to alleviating this problem is by embedding their analysers in modelling environments which lower the barrier to use of the methods. These environments can then be adopted and applied by the service providers in-house, allowing them to evaluate their service provision without revealing sensitive information about their current service. The SENSORIA Development Environment (SDE) assists us in the goal of bringing state-of-the-art analysis methods closer to the service providers who need to apply them.

The SDE brings together analysis tools for process calculi and allows users to combine them using scripting. In particular the SDE includes analysis tools for the two process calculi which we use in the present work:

– Finite State Processes (FSP), and
– Performance Evaluation Process Algebra (PEPA).

Specifically, the SDE hosts the following tools:

– the LTSA model-checker for FSP, and
– the `ipclib` response-time analyser for PEPA.

We describe the use of the SDE on an analysis problem which is of particular interest to one of the industrial partners in our current research project. The partner in question is a consultancy providing advice to a major Bavarian car manufacturer. They have been asked to consult on a subscription service which uses the on-board diagnostic and communication systems in high-end cars to provide an accident assistance service. In this paper we use the SDE and other tools to assess the accident assistance service against both safety properties (using model-checking over labelled transition systems) and response-time properties (using transient analysis of continuous-time Markov chains).

2 Service Design

Our model of the Accident Assistance Service details the events which are the area of responsibility of the service itself. That is, those activities which occur between an accident report being received and the service discharging its responsibility to act on the accident report. In some cases this will lead to an ambulance being sent, and in other cases not. Our model does not require us to know – or allow us to predict – anything about activities which happen before or after these events. For example, we do not estimate how often accidents occur and we do not calculate how long ambulances take to arrive. Both of these many be interesting to know, but our model here does not speak of them.

The activity diagram shown in Figure 1 provides a high-level view of the events in the scenario. Events are triggered by an incoming accident report which the service begins to process (*Process Accident Data*). From this incoming report is obtained all of the available information about the status of the vehicle. Multiple attempts are made to contact the driver (*Contact Driver*) and the service must then respond (*Classify Severity*). In the cases when the accident has been classified as critical medical assistance is dispatched (*Dispatch Ambulance*). Events are logged in a central log for audit purposes, whether an ambulance was dispatched or not.

The service needs to take two major decisions during its realm of responsibility. First, whether to continue to attempt to contact the driver, or assume that they are injured. Second, to classify this accident as critical or not.

Considering the accident assistance service at a lower level of detail we note that the service is triggered by any impact or collision which causes the car

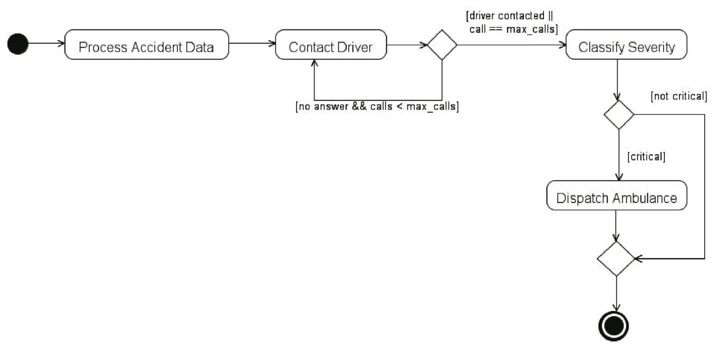


Fig. 1. UML2 Activity Diagram of Accident Assistance Service

airbag to deploy. Immediately after the airbag has deployed the on-board communication module transmits to the assistance service a report with as much information as it can obtain from the car's diagnostic system. This report includes information about the state of the car itself obtained from sensors in the engine and the braking system. The report also specifies the speed of the car at the moment of impact and, most importantly, the geographical location of the car as obtained from the on-board GPS.

On receipt of such a report, the subscription service attempts to contact the registered driver of the car by mobile telephone. If the driver answers the telephone and confirms that they are unhurt then no further action needs to be taken. If they instead say that they have been hurt in the accident then the service will dispatch an ambulance to the reported location to assist them.

The third case to consider occurs when the service cannot get confirmation from the driver that they do not need assistance. It might seem that the obvious course of action should be to consider not getting an answer to be a critical case but there is evidently a possibility that the service will send an ambulance when it is not needed. That is, the driver is unhurt but did not have their mobile telephone with them, or it had no battery charge, or they had no signal from their telephone service provider, or many other similar reasons. Because critical services should not be deployed without good reason, the accident assistance service would like to reduce the number of occasions when an ambulance is dispatched in error.

The information on the car status and the speed of the car at the moment of impact sent with the accident report become significant in the case where we have no answer from the driver. The service needs to classify this accident as critical or not and many factors will influence the classification of an accident. Speed at the time of impact is a major factor, as is degree of damage to the car, but the geographical location and the time of day also impact on the classification. The reason for this is that the injured driver is less likely to get help from passing motorists if the car accident happens in a remote location late at night than if the accident happens in a heavily-populated area during the day.

In the case of no answer and car diagnostics which point to very little damage (say, the car was stationary at the time of impact, and the engine, brakes, lights and other critical functions seem to be functioning normally) then the service will decide not to send an ambulance to prevent sending one when it could be needed elsewhere.

3 Safety Analysis of the Assistance Service

In this section we discuss the safety analysis of the Accident Assistance Service. Safety Analysis is concerned with assuring that properties of the service behaviour are upheld and in particular, that there are no undesirable behaviour traces exhibited given the various constraints of the service. The nature of this service exhibits various specified conditions of progress, for example, if the driver answers his or her cellphone within a number of attempts, then the progress is different to that if he or she does not. Such conditions need to be examined for behaviour consistency. For this reason, we focus on the behaviour process of the service rather than data analysis, given the design of the service specified in section 2 and an implementation written in some software process language. For the purpose of our analysis, we translate the service process workflow in to the Finite State Process (FSP) notation to concisely and formally model the workflow states and transitions.

3.1 FSP, LTS and Behaviour Models

The FSP notation [1,2] is designed to be easily machine readable, and thus provides a preferred language to specify abstract processes. FSP is a textual notation (technically a process calculus) for concisely describing and reasoning about concurrent programs. FSP supports a range of operators to define a process model representation.

A summary of the operators for FSP is given as follows.

Action prefix "`->`": `(x->P)` describes a process that initially engages in the action `x` and then behaves as described by the auxiliary process `P`;

Choice "`|`": `(x->P|y->Q)` describes a process which initially engages in either `x` or `y`, and whose subsequent behaviour is described by auxiliary processes `P` or `Q`, respectively;

Recursion: the behaviour of a process may be defined in terms of itself, in order to express repetition;

Sequential composition "`;`": `(P;Q)` where `P` is a process with an `END` state, describes a process that behaves as `P` and when it reaches the `END` state of `P` starts behaving as the auxiliary process `Q`;

Parallel composition "`||`": `(P||Q)` describes the parallel composition of processes `P` and `Q`;

Relabelling "`/`": Re-labelling is applied to a process to change the names of action labels. The general form of re-labelling is `/ {newlabel/oldlabel}`;

The hiding, trace equivalence minimisation, and weak semantic equivalence minimisation operators of FSP are not used here. We omit their descriptions for brevity.

3.2 Translation of Service Design to FSP

The Accident Assistance Service design illustrated in Figure 1 specifies a number of activity transitions linked either as a sequence or through decisions. To translate these to the FSP notation, and a formal model, we traverse the workflow and build a series of FSP processes composed to build a complete process architecture model. To begin with we start with the *initial node*. The initial node specifies a transition to the *Process Accident Data* activity, which represents a report from the on-board vehicle diagnostics system. Additionally at this step, we need to determine and store a variable which holds a report status. In the FSP (listed below), we represent these actions by creating a process (PROCESSACCIDENTDATA) and a sequence for the choice of status reported from the vehicle diagnostics. The VEHDIAGCHOICE process has two options, one for a normal status or one for a critical status. Note that we need to look ahead to see which activity follows this to determine whether this process composition is complete. In this case the next activity is again a simple transition.

```
// Diagnostics Composition
DIAGCHOICE =
 ( vehicle.emergsrv.diags_normal->emergsrv.diag.write[0]->END
 | vehicle.emergsrv.diags_critical->emergsrv.diag.write[1]->END ).
REQUESTDIAGS = (emergsrv.vehicle.requestdiags->END).
REQUESTDIAGSEQ = REQUESTDIAGS; DIAGCHOICE; END.
||DIAGNOSTICS = (REQUESTDIAGSEQ).
```

Immediately following the *Process Accident Data* activity is the *Contact Driver* activity. This activity is linked with two steps in the workflow. Firstly the activity itself is linked with a decision step, to determine if either the driver was successfully contacted or a maximum number of calls has been reached. The composition for this therefore is the action of calling the driver, and then a choice of either proceeding to the next step of the workflow (because the call was successful or maximum call attempts reached) or the action is repeated.

```
// Call Attempts Composition
const Max = 3 // no of calls before automatic dispatch
range Int = 0..2 // 0 - not critical, 1 - critical, 2 - unknown

CALLATTEMPT(N=0) = CALL[N],
CALL[v:Int] = (emergsrv.driver.callphone->ANSWER[v]),
ANSWER[v:Int] =
  ( driver.emergsrv.noanswer->CALL[v+1]
  | driver.emergsrv.answer->ANSWEREDACTION ),
ANSWEREDACTION =
  ( emergsrv.phone.write[0]->END
  | emergsrv.phone.write[1]->END ),
CALL[Max] = (emergsrv.phone.write[2]->END).

set ACTSET = {emergsrv.driver.callphone,
                  driver.emergsrv.noanswer, driver.emergsrv.answer}
TERMS = (ACTSET->TERMS).
||CALLATTEMPTS = (TERMS || CALLATTEMPT(0)).
```

Lastly, two processes are built to represent the *Classify Severity* and *Dispatch Ambulance* activities. In the first activity again the workflow specifies a decision point. The classification activity represents a choice of transition depending on the status of both the accident data report and contacting the driver. To represent this in FSP we recall the values assigned as part of the two previous compositions (*Process Accident Data* and *Contact Driver*). The choice of process transition is represented using a conventional structured construct of IF..THEN..ELSE. The FSP below represents the conditional operation of the diagnostics status reported by the vehicle. A similar model is built to represent the result of calling the driver and then these two choice models are composed. The comparison of whether the status is critical determines if the *Dispatch Ambulance* is undertaken.

```
// check phone answered
QUERYPHONESTATUS = (emergsrv.phone.read[i : 0..2]->QUERYPHONESTATUS[i]),
QUERYPHONESTATUS[i : 0..2] =
    if (i==2) then QUERYDIAGSTATUS; END
    else if (i==1) then DISPATCH; END
         else LOGREPORT; END.

// check diagnostic information received
QUERYDIAGSTATUS = (emergsrv.diag.read[i : 0..1]->QUERYDIAGSTATUS[i]),
QUERYDIAGSTATUS[i : 0..1] =
    if (i==1) then DISPATCH; END
    else LOGREPORT; END.
```

A simple sequence process represents the actual *Dispatch Ambulance* activity, which is triggered if the status is critical. In preparation for analysis a complete architecture model – representing the sequence composition of the workflow processes we have defined – is also summarised in the code below.

```
// Dispatch Ambulance
SENDAMBULANCE = (emergsrv.station.send_ambulance->END).
||DISPATCH = (SENDAMBULANCE).

// Dispatch Report (Final Action)
LOG = (emergsrv.log.result->END).
||LOGREPORT = (LOG).

// Service Main sequence
set PHONE_ALPHABET = {emergsrv.phone.{read,write}.[0..2]}
MAINSEQ = ACCIDENT; AIRBAG; GETSTATUS; QUERYPHONESTATUS; END
        + {PHONE_ALPHABET}.
```

3.3 Analysis Using LTSA

The constructed FSP can be used to model the exact transition of workflow processes through a modelling tool such as the Labelled Transition System Analyzer (LTSA) [1], which compiles an FSP model into a state machine and provides a resulting Labelled Transition System (LTS). LTSA is made available as a component of the SDE. The LTSA tool has an inbuilt safety check to determine whether a specified process is deadlock free. Deadlock analysis of a model involves performing an exhaustive search of the LTS for deadlock states (i.e. states with no outgoing transitions). A default deadlock check of the service process results in no violations being found (i.e. that there are no deadlock states in the model).

However, we need to check properties of the service to meet the requirements in operation. For example, that an ambulance is dispatched only when requested by the driver or, (in the case of no answer) when the car diagnostics indicate severe damage. We add this property to the model through a further FSP statement using the keyword **property** and specifying that both the driver asked for an ambulance (`emergsrv.phone.read[1]`) or did not answer but the diagnostic information on the car indicated severe damage (`emergsrv.diag.read[1]`). The formal statement of this property is listed below.

```
// FSP Property to check only critical status leads to dispatch
property PROP =
 ( emergsrv.phone.read[2] ->
       emergsrv.diag.read[1] ->
            emergsrv.station.send_ambulance -> END
 | emergsrv.phone.read[1] ->
       emergsrv.station.send_ambulance -> END
 ).
```

Using this property specification language we were able to apply model-checking to uncover errors in our original model which we corrected before going on to response time analysis of the model.

4 Response-Time Analysis of the Assistance Service

Response-time analysis considers the timed behaviour of the system under study in the context of a particular workload and a particular sequence of activities which must take place. It is possible to think of this as a sub-scenario with a distinguished start activity which starts a clock running and a distinguished stop activity which stops it. The analysis will determine the probability of completing the work needed to take us from the start activity to the stop activity, via any possible path through the system behaviour. This probability value can be plotted against time to give a complete picture of the response-time distribution. With respect to the accident assistance service we will consider the response-time from the airbag being deployed until the assistance service logs that it has completed its investigation and has discharged its duty to send an ambulance if one was required (or it has determined that an ambulance was not required).

For this aspect of the work we require a timed process algebra (FSP is untimed). We will use Performance Evaluation Process Algebra (PEPA) [3], a stochastically-timed Markovian process algebra. The PEPA language is supported by the Sensoria Development Environment and by formal analysis tools on the SDE such as the PEPA Eclipse Plug-in project [4] and the `ipclib` tool suite [5].

4.1 PEPA, CTMCs and Response Time

Many of the combinators of the PEPA language resemble the operators of FSP seen in Section 3. The most significant difference is that all activities in PEPA are timed. We provide a brief summary of the language here, referring the reader to [3] for the full formal details.

Prefix: $(\alpha, r).P$ describes a process which will first perform the activity α at an exponentially-distributed rate r and then evolve to become P.

Choice: $(\alpha, r).P + (\beta, s).Q$ describes a process which either performs activity α at rate r and evolves to become P, or it performs activity β at rate s and evolves to become Q. The two activities α and β are simultaneously enabled and whichever completes first will determine the continuation of the process.

Cooperation: In $P \underset{\mathcal{L}}{\bowtie} Q$ the processes P and Q cooperate over all of the activities in the set $\mathcal{L}$, meaning that they must synchronise on these activities. Activities not in $\mathcal{L}$ are performed independently, without any synchronisation. We write $P \parallel Q$ if $\mathcal{L}$ is empty.

Hiding: The process $P/\mathcal{L}$ is identical to P except that any uses of the activities in the set $\mathcal{L}$ have been renamed to τ (the silent activity) and no other process may cooperate with these activities. The duration of the activity is unchanged so that, for example, (β, s) becomes (τ, s).

Because an exponentially-distributed random variable is associated with the rate of each activity a PEPA model gives rise to a stochastic process, specifically a continuous-time Markov chain (CTMC). The generator matrix, Q, of this CTMC is "uniformised" with: $P = Q/q + I$ where $q > \max_i |Q_{ii}|$ and I is the identity matrix. This process transforms a CTMC into one in which all states have the same mean holding time $1/q$. The required computation for the response-time distribution is to compute the probability of reaching a set of designated target states from a set of designated source states. This rests on two key subcomputations. First, the time to complete n hops ($n = 1, 2, 3, \ldots$), which is an Erlang distribution with parameters n and q. Second, the probability that the transition between source and target states occurs in exactly n hops. After generating the state-space of the model to derive the generator matrix of the CTMC the required response-time distribution can be computed by uniformisation [6,7].

4.2 Analysis Using `ipclib`

Because of the strong similarity between FSP and PEPA translating our existing FSP model into the PEPA language was straightforward. We checked the consistency of the PEPA model against the FSP model by translating the FSP logical propositions into PEPA stochastic probes [8] and confirmed that (the translations of) these propositions held for (the translation of) the model.

We analysed the PEPA model with the `ipclib` [5] tool suite. We investigated the response time of the model across a range of feasible rates for each of the activities performed. Each of the rates can vary independently of the others and to cover all of the cases being considered we generated more than 300 experimental runs of the `ipclib` tools with different parameter values. We performed sensitivity analysis of response-time profiles for the possible parameter values.

We varied the three rates corresponding to the rates at which each subsequent call attempt is ended either by the customer answering the phone or a timeout.

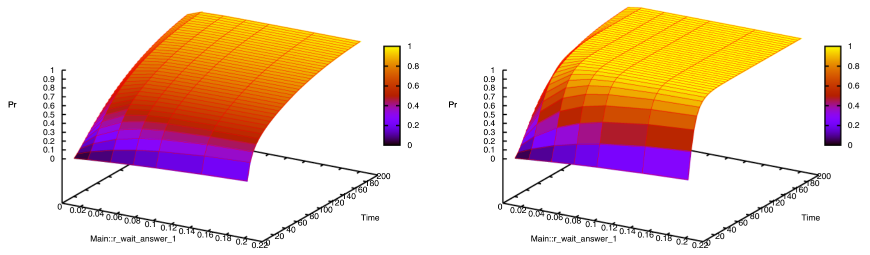


Fig. 2. Response-time graphs for the accident assistance service

These are the rates $r\_wait\_answer\_1$, $r\_wait\_answer\_2$ and $r\_wait\_answer\_3$. The analysis tool then produces a group of sensitivity-analysis graphs for each of the three rates. Each graph in the first group plots the cumulative distribution function of completing the passage against the varying rate of $r\_wait\_answer\_1$ while the other two rates are kept constant. There is one such graph in the first group for all possible combinations of the two rates $r\_wait\_answer\_2$ and $r\_wait\_answer\_3$. Each graph in group one relates the effect that varying the rate $r\_wait\_answer\_1$ will have on the completion of the passage. There must be one for every combination of the other two rates because the effect that $r\_wait\_answer\_1$ has on the outcome depends upon the values of the other two rates.

Figure 2 shows two graphs in the first group of sensitivity graphs. In the graph on the left rates $r\_wait\_answer\_2$ and $r\_wait\_answer\_3$ are at low values while in the graph on the right the two rates are held at high values. We see that in either case the rate of $r\_wait\_answer\_1$ does have an effect on the probability of completion. We can see this because each line in the graph is different resulting in a 'warped' surface plot. The graph on the left is less warped than the graph on the right. This suggests that varying the rate of $r\_wait\_answer\_1$ has more effect whenever the two other rates are at high values. This is because when those rates are high, the bottleneck in completing the passage becomes the activities performed at rate $r\_wait\_answer\_1$. When the other two rates are lower they become the bottleneck and indeed we see that by time $t = 15$ there is not yet a probability of approximately 1.

Figure 3 shows two graphs in the third group of sensitivity graphs. These look similar to the graphs in Figure 2 but we see that they are less warped. This tells us that $r\_wait\_answer\_3$ has less effect on the probability of completing the passage. This confirms our intuition because the activities which occur at rate $r\_wait\_answer\_3$ are not always performed at all in a successful completion of the passage. In some cases the driver will answer the phone call at the first or second attempt. The graph on the left shows the sensitivity of $r\_wait\_answer\_3$ when the rates $r\_wait\_answer\_1$ and $r\_wait\_answer\_2$ are held at low values and in the graph on the right those rates are held at high values. As before we see that the varied rate has more effect when the unvaried rates are held high. Again this confirms our intuition; all of the rates measured here are performed along the passage and increasing any one of them has a positive effect on the

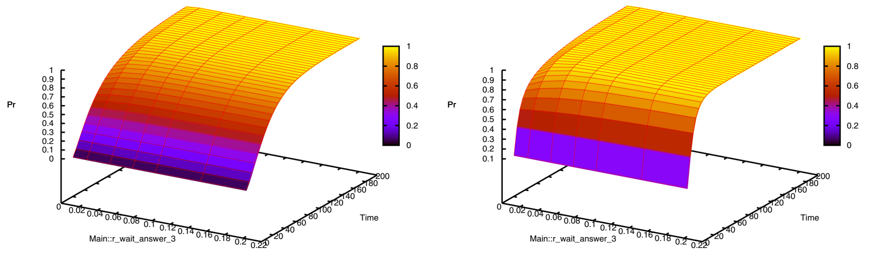


Fig. 3. Response-time graphs for the accident assistance service

probability of completion, therefore we expect that when one rate is the slowest rate varying that rate will achieve more of a performance gain than varying the already faster rates. Overall from these sensitivity-analysis graphs we can surmise that if one wishes to increase the performance then it is of most benefit to increase the rate of the first rate ($r\_wait\_answer\_1$) before the others and the second rate ($r\_wait\_answer\_2$) before the final rate ($r\_wait\_answer\_3$). However if any of the rates are significantly slower than the others then that is the rate which should be increased even if it is $r\_wait\_answer\_3$.

We also obtain summary information about all of the experiments performed. The graph (Figure 4, left) shows best case, worst case, median and the 20-80 percentiles over all response-time graphs. From this we can see that we need to wait until time 80 to be over 50% confident that the passage will have completed regardless of the configuration. However at this time the median is above 95% meaning that in half of the configurations we are very confident that the passage will have completed. Also at this time removing the worst performing 10% of the configurations gives us over 80% confidence of completing the passage. Also from this graph we can see that at time 120 the median begins to approximate 100% confidence that the passage has completed (meaning that in half of the configurations we can be sure that the passage has completed).

Probability of completion at a time bound can be plotted across the more than 300 experiments which we performed, leading to a different summary (Figure 4, right) from which we can make conclusions such as "the probability of service being completed by $t = 195$ seconds is at least 90%, even if all calls to the driver take the longest time which has been allocated for them". We can also see from both styles of summary graphs that the difference between the best and lowest performing configurations starts off small at low time bounds where there is little probability of completing the passage regardless of the configuration. As the time increases the gap grows wider until at some point the worst performing configuration begins to close the gap until eventually there is a near 100% probability of completing the passage for all configurations. For the modeller if this peak in the difference occurs at an important time the configuration of the real system is very important. On the other hand the modeller may be given more confidence if this peak occurs before a time bound in which (for whatever reason) they are particularly interested.

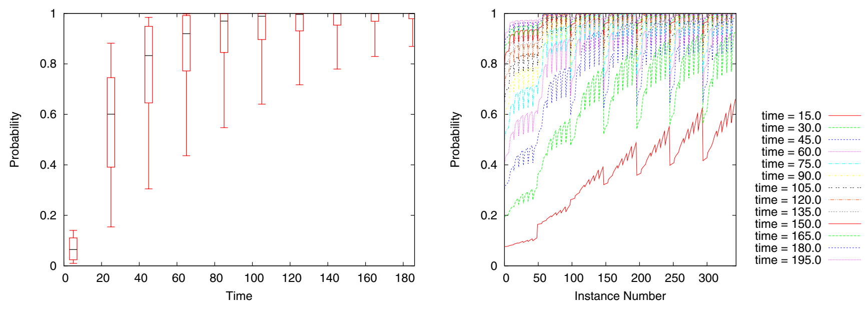


Fig. 4. Summary information for all response-time calculations

5 SENSORIA Development Environment

The previous sections have described qualitative and quantitative methods with corresponding tools for performing safety and response-time analysis of services. In order to make these tools available to software engineers in the field, they have been integrated into the SENSORIA Development Environment (SDE). The SDE is a modelling, simulation and analysis platform which supports the integrated evaluation of both functional and non-functional aspects of systems and services. Based on Eclipse, the SDE may also be integrated with various other tools available for this platform.

5.1 SDE Features

Being based the OSGi platform underlying Eclipse, the SDE is itself built in a service-oriented way. Upon installation, tools register themselves in the SDE core, thereby offering their functionality to all other installed tools, including orchestrators. Through various integrated tools, the SDE currently offers functionality which falls into these major categories:

– **Modelling functionality.** This includes graphical editors for familiar modelling languages such as UML, as supported by industry-standard tools such as the Rational Software Architect, which allow for intuitive modelling on a high level of abstraction. However, there are also text- and tree-based editors for process calculi.
– **Formal analysis functionality.** The SDE offers model checking and numerical solvers for stochastic methods based on process calculi code defined by the user or generated by model transformation.

The tools presented in this paper offer functionality which falls into the second category. In particular, the PEPA tools including the SRMC extensions [4] as well as LTSA and WS-Engineer have been made available as services in the SDE. They offer the follow functionality:

- **Simulators and Single-Step Navigators** which allow the user to investigate a model and look for modelling errors in the input and unexpected behaviour in execution related to liveness or reachability problems.
- **Model-Checkers** which check consistency between the model and an interesting property. In case of errors, a (graphical) violation trace is generated.
- **Steady-State and Transient Analysers** for performance analysis. These analysers provide simulation traces showing variation in the states of the model components over time, and utilisation charts, cumulative distribution plots and other visualisations which represent graphically the numerical results computed.

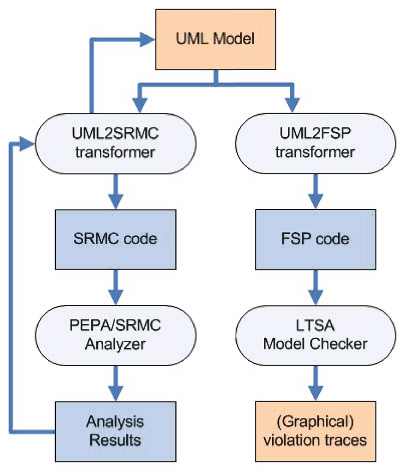
Through scripting, these analyses can be combined, as will be outlined in the next section.

5.2 Orchestrating Tools with the SDE

During software development and analysis of software systems, it is often desirable to run several analyses as a suite, perhaps passing input from one tool to the other, and gathering and presenting the output in a single place – in other words, orchestrating tools to perform as a whole.

To enable such orchestrations, the SDE offers the ability to compose installed tools by means of arbitrary orchestration mechanisms. In particular, we offer the ability to script such orchestrations by means of JavaScript. An orchestration may be written as a set of annotated JavaScript functions, thus in effect creating a new service orchestrating the referenced tools.

As an example, we consider the orchestration of the tools for the methods presented in the previous sections to perform analysis on the Automotive Accident



```
function checkUML(umlSource) {

        // transform to PEPA
        uml2pepa = sCore.findToolById("uml2pepa");
        pepaModel = uml2pepa.transform(umlSource);

        // perform analysis with PEPA tool
        pepa = sCore.findToolById("pepa");
        markovChain = pepa.getMarkovChain(pepaModel);
        distribution = pepa.getSteadyState(markovChain);
        throughput = pepa.getThroughput(markovChain);

        // back annotation
        uml2pepa.reflect(umlSource, distribution, throughput);

        // transform to LTSA FSP (input to WS-Engineer)
        uml2ltsa = sCore.findToolById("uml2ltsa");
        fspModel = uml2ltsa.uml2fsp(umlSource);

        // perform analysis with WS-Engineer
        wse = sCore.findToolById("wsengineer");
        result = wse.analyse(fspModel);

        if (result.hasErrors())
                return ltsa.mscFromLTSATrace(result.getTrace());

        return umlSource;

}
```

Fig. 5. Orchestration of the four tools together with JavaScript orchestration code

Assistance Service. As the orchestration is intended to be used by developers not too familiar with formal methods, we will start with a UML model and, at the end, provide back-annotated UML for showing results of the quantitative analysis with PEPA/SRMC as well as a (graphical) violation trace in case of errors during the qualitative analysis with LTSA.

The JavaScript code for this orchestration is concise (Fig. 5, right). In the beginning, we retrieve the tools by unique identifiers, invoke the functions involved, and finally return the combined output to the user. Within the SDE, a generic wizard handles this call such that users are able to select the input model graphically using a file open wizard, and also get the results opened in appropriate editors inside the Eclipse workbench or externally. Thus, it is easy for developers to employ such orchestrations as part of their work.

6 Related Work

We have considered performance aspects of the accident assistance service previously [9]. Our work in that earlier paper did not incorporate any model-checking aspects and dealt only with a simpler version of the accident assistance service without priority classifications.

Other authors have applied model-checking to analyse automotive safety services. In [10] the authors use high-level UML specification that makes use of domain-specific extensions The on-the-fly model checker UMC [11] is subsequently used to verify a set of correctness properties formalized in the action- and state-based temporal logic UCTL. Subsequently to the authors writing this paper the UMC model-checker has been made available as a service on the SDE, opening the possibility of conjoined use with the methods deployed in this paper.

In [12] an on-road assistance scenario is considered where the authors treat the process of obtaining assistance for a car subsequent to a breakdown (which is not necessarily a life-threatening accident). The authors formalise the problem in the COWS process calculus and give a formal treatment of fault and compensation handling.

Formal model-checking of service compositions has been undertaken mostly on their implementation, rather than the design of the service itself. For example, as a result of new standards to define and execute service compositions (such as the Web Services Business Process Execution Language), model-checking these has included translation to Finite State Machines, graphs and simulation models. We have already considered analysing these models in [13], whilst more recently in [14] using UML Deployment Models to analyse service compositions with deployment constraints. There has also been some similiar work on UML to Finite State Machines, particularly Activity Diagrams in [15]. These works also define a formal semantics for UML Activity Diagrams, but do so with a differing focus of aligning activities as two or more distributed processes and structure of roles within activities.

7 Conclusions

In this paper we presented a co-ordinated analysis of safety properties and the response-time profile of an automotive accident assistance scenario. We used the untimed process calculus FSP to express our model of the scenario and model-checked critical properties using the WS-Engineer tool in the SDE. We added rate information to convert the FSP model into one in the stochastically-timed process calculus PEPA. We next converted the FSP logical properties into stochastic probes on the PEPA model. We then used the PEPA Eclipse Plug-in to check that the PEPA model which was obtained by translation from FSP respected the stochastic probes which were obtained by translation from the FSP logic. The PEPA Eclipse Plug-in confirmed that this was the case. We then used the `ipclib` tool suite to perform many response-time evaluations leading to the quantitative results seen.

Our overarching goal in this work has been to make the outputs from the formal analysis tools open to inspection by users who are not experts in process calculi. To this end, reports are often returned in a graphical form such as a message sequence chart or a graph. Our next goal is to streamline the modelling process by allowing users to express their initial model in a language with widespread acceptance, such as UML. We have made some progress on this, and have a scripting infrastructure in place to allow such conversions to be performed automatically but more remains to be done in this area.

Acknowledgements. The authors are supported by the EU FET-IST Global Computing 2 project SENSORIA ("Software Engineering for Service-Oriented Overlay Computers" (IST-3-016004-IP-09)) and the EPSRC PerformDB project (EP/D054087/1). The ipc/Hydra tool chain has been developed in co-operation with Bradley, Knottenbelt and Dingle of Imperial College, London.

References

1. Magee, J., Kramer, J.: Concurrency - State Models and Java Programs, 2nd edn. John Wiley, Chichester (2006)
2. Magee, J., Kramer, J., Giannakopoulou, D.: Analysing the behaviour of distributed software architectures: a case study. In: 5th IEEE Workshop on Future Trends of Distributed Computing Systems, Tunisia (1997)
3. Hillston, J.: A Compositional Approach to Performance Modelling. Cambridge University Press, Cambridge (1996)
4. Tribastone, M.: The PEPA Plug-in Project. In: Harchol-Balter, M., Kwiatkowska, M., Telek, M. (eds.) Proceedings of the 4th International Conference on the Quantitative Evaluation of SysTems (QEST), pp. 53–54. IEEE Computer Society Press, Los Alamitos (2007)
5. Clark, A.: The ipclib PEPA Library. In: Harchol-Balter, M., Kwiatkowska, M., Telek, M. (eds.) Proceedings of the 4th International Conference on the Quantitative Evaluation of SysTems (QEST), pp. 55–56. IEEE Computer Society Press, Los Alamitos (2007)

6. Grassmann, W.: Transient solutions in Markovian queueing systems. Computers and Operations Research 4, 47–53 (1977)
7. Gross, D., Miller, D.: The randomization technique as a modelling tool and solution procedure for transient Markov processes. Operations Research 32, 343–361 (1984)
8. Argent-Katwala, A., Bradley, J., Dingle, N.: Expressing performance requirements using regular expressions to specify stochastic probes over process algebra models. In: Proceedings of the Fourth International Workshop on Software and Performance, Redwood Shores, California, USA, pp. 49–58. ACM Press, New York (2004)
9. Clark, A., Gilmore, S.: Evaluating quality of service for service level agreements. In: Brim, L., Leucker, M. (eds.) Proceedings of the 11th International Workshop on Formal Methods for Industrial Critical Systems, Bonn, Germany, pp. 172–185 (2006)
10. ter Beek, M.H., Gnesi, S., Koch, N., Mazzanti, F.: Formal verification of an automotive scenario in service-oriented computing. In: Proceedings of the 30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, pp. 613–622. ACM Press, New York (2008)
11. UMC model checker (2008), `http://fmt.isti.cnr.it/umc/`
12. Lapadula, A., Pugliese, R., Tiezzi, F.: Specifying and analysing SOC applications with COWS. In: Degano, P., De Nicola, R., Meseguer, J. (eds.) Concurrency, Graphs and Models. LNCS, vol. 5065, pp. 701–720. Springer, Heidelberg (2008)
13. Foster, H., Uchitel, S., Magee, J., Kramer, J.: Model-based Verification of Web Service Compositions. In: Proc. of the 18th IEEE Int. Conference on Automated Software Engineering, pp. 152–161. IEEE Computer Society Press, Los Alamitos (2003)
14. Foster, H., Emmerich, W., Magee, J., Kramer, J., Rosenblum, D., Uchitel, S.: Model Checking Service Compositions under Resource Constraints. In: The European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2007) (2007)
15. Badica, C., Badica, A., Litoiu, V.: Role activity diagrams as finite state processes. In: Second International Symposium on Parallel and Distributed Computing (2003)