Modular Performance Modelling for Mobile Applications (Abstracts Only)

Modular Performance Modelling for Mobile Applications

Niaz Arijo, Reiko Heckel
Department of Comp. Sci.
University of Leicester
nha2|reiko@mcs.le.ac.uk

Mirco Tribastone
Institut für Informatik
LMU München
tribastone@pst.ifi.lmu.de

Stephen Gilmore
School of Informatics
University of Edinburgh
stg@staffmail.ed.ac.uk

ABSTRACT

We propose a model-based approach to analysing the performance of mobile applications where physical mobility and state changes are modelled by graph transformations from which a model in the Performance Evaluation Process Algebra (PEPA) is derived. To fight scalability problems with state space generation we adopt a modular solution where the graph transformation system is decomposed into *views*, for which labelled transition systems (LTS) are generated separately and later synchronised in PEPA. We demonstrate that the result of this modular analysis is equivalent to that of the monolithic approach and evaluate practicality and scalability by means of a case study.

Keywords

mobility; performance modelling; graph transformation; process algebra

1. INTRODUCTION

We present a methodology for the performance modelling of mobile systems using stochastic graph transformations. Mobility aspects such as location-aware behaviour and physical proximity are captured by graphs, which enjoy a formal characterisation yet preserve the look and feel of mainstream visual notations. Structural changes and computations are described by transformation rules endowed with exponentially distributed delays. This leads to transition systems labelled with rate information which admit a continuous-time Markov chain (CTMC) interpretation, ultimately used for the evaluation of the performance of the system.

Computationally, the major bottleneck of this analysis is the generation of the transition system via the graph transformation rules, since the typical *pattern-matching* operation that is carried out is known to be NP-complete. In this paper, we discuss a novel approach to mitigating this problem which consists in disassembling the overall system under study into *views*, i.e., subsystems addressing distinct concerns of the system. We propose a running example of a traffic information system (TIS) where we consider one view for physical mobility and a separate one for communication.

Each view will give rise to a separate labelled transition system (LTS), which is usually much smaller than the overall model's LTS. We define a suitable synchronisation semantics between the views' LTSs based on the stochastic process calculus PEPA [6], interpreting a view as process-algebraic component. We prove that the resulting synchronised LTS (hence the related CTMC) is isomorphic to the model's overall LTS. However, with this approach the (demanding) application of the graph transformation rules is confined to smaller graphs. Furthermore, the synchronisation operation is much faster because it does not require any form of pattern matching. The computational advantages with respect to a monolithic derivation of the overall LTS are practically shown via numerical tests on the running example.

Related Work. Occasionally, formal languages are integrated with standard modelling languages like the UML to provide a more mainstream front-end notation for modelling [1]. In our case the visual language itself has a formal semantics, which allows us to verify the relation with the process calculus. Other timed modelling languages which allow the representation of location and mobility include the Gnosis modelling and simulation language [2], where models are nondeterministic programs semantically based on a process calculus. Rather than on a programming language, our approach is based on a high-level visual modelling notation with a formal semantics.

We extend and apply earlier work on stochastic modelling and modularity of graph transformation systems, especially [5].

2. PERFORMANCE MODELLING

In this section we introduce a first, monolithic version of our approach consisting of modelling communication and mobility by means of graph transformation rules, generating the labelled transition system for a given start graph in the graph transformation tool GROOVE [9], and deriving a PEPA process for performance analysis.

2.1 Mobility and Interaction

We use graphs to model the structure of the application, including the topology of locations, the current locations of relevant devices, but also existing links between application components as well as their states. Our graphs come in two flavours, as type and instance graphs. The type graph provides a structural model of the admissible states of the system, similar to the way a class diagram describes valid object structures.

Fig. 1 shows the type graph of the TIS model, with an instance graph in Fig. 2 representing a map of Roads and Junctions as well as two Cars following predefined Paths. The model allows to rep-

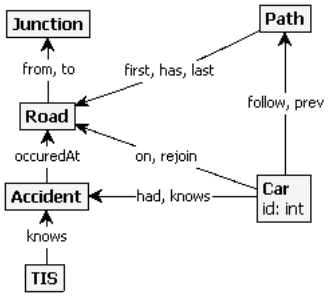


Figure 1: Type Graph of a Traffic Information System (TIS).

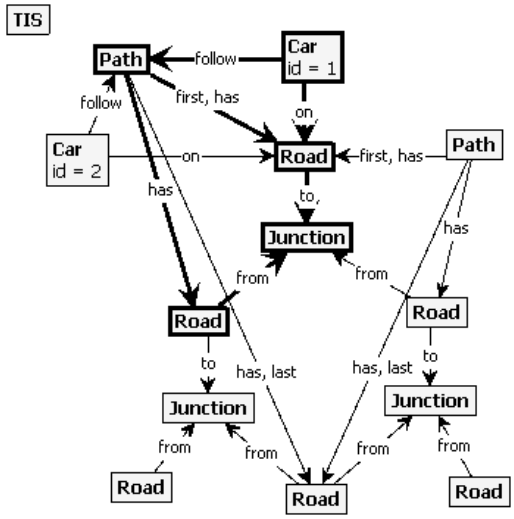


Figure 2: Start graph of the Traffic Information System (match for moveCar(1) indicated in bold)

resent Accidents that can be reported to the TIS, to share the information with other Cars. To identify Cars, these nodes have been given id attributes of type integer. In general we allow graph nodes to be attributed by values of predefined data types, such as strings or natural numbers. In this paper all attribute values will be positive integers, but see [8] for a general treatment of attributed graphs and their transformation.

Instance graphs are transformed by rules modelling operations, distinguished into two (not strictly disjoint) categories: Mobility operations access and change the location of devices while service operations capture the state changes brought about by the sending and receiving of messages between services and their client applications. Note that we do not model the communication itself, i.e., rules will not describe the sending and receiving of messages, but only their effects on the states of components.

Formally, a graph transformation rule $p(\bar{x}) : [N]\ L \rightarrow R$ consists of a rule name p, a declaration of formal parameters $\bar{x} = x_1 \dots x_n$ with variables x_i ranging over attribute values, a left-hand side L representing the pattern of elements required for the application of the rule, a right hand side R describing the situation this pattern is to be replaced with, and a negative application condition N stating the absence of certain elements in the context of this pattern. The intersection graph $L \cap R$ defines the elements that are read by the rule, but are not consumed. Nodes in L, R, and N are attributed by expressions over variables in X.

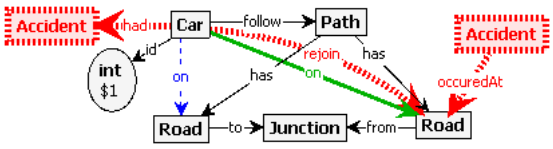


Figure 3: Rule $moveCar(car)$ models the mobility of a Car following a Path.

In GROOVE notation [9] the various components of a rule (called readers, erasers, creators or embargoes) are combined within a single rule graph, distinguishing them by different colours and styles. Consider for example the rule in Fig. 3 modelling the movement of a Car from one Road to another across a Junction. Readers in $L \cap R$, such as the nodes of type *Car, Path, Road, Junction*, and the edges of type *follow, has, to, from*, are thin and solid ordinary graph elements that are required, but preserved by the transformation. Erasers in $L \setminus R$, such as the edge of type *on* pointing to the left-most *Road* node, are shown as thin and dashed elements, to be deleted. Creators in $R \setminus L$, such as the other edge of type *on*, represented by slightly wider, solid outlines, are to be created by the rule. Embargoes in N, such as the nodes of type *Accident* and the edges of type *occurredAt*, *had* and *rejoin*, are represented by wider and dashed outline. They prevent a rule from being applied if the corresponding elements are present in the graph. In a printed version of this paper, reader elements are shown in black, while all other elements are in grey. The GROOVE tool (and the pdf version of this paper viewed on-screen or printed in colour) uses blue for erasers, green for creators and red for embargoes. Attribute values are depicted as circles pointed to by an edge from the attributed node. For example, in the rule in Fig. 3 node Car has attribute id whose value is depicted as parameter 1. It corresponds to the 1st formal parameter in the rule's signature *moveCar(car)*.

Formally, a *typed attributed graph transformation system with rule signatures* (GTS) is a tuple $\mathcal{G} = (TG, P, \pi)$ where

- TG is a type graph,
- P is a set of rule names,
- $\pi : P \longrightarrow X^* \times Rules(TG, X)$ assigns to each rule name a pair $\pi(p) = (\bar{x}) : [N]\ L \rightarrow R$ of a parameter declaration $\bar{x} = x_1 \dots x_n$ and a rule $[N]\ L \rightarrow R$. We call $p(\bar{x}) : [N]\ L \rightarrow R$ a parameterised rule and refer to $p(\bar{x})$ as its signature.

Given a graph G and a rule $p(\bar{x}) : [N]\ L \rightarrow R$, we can apply the rule if there is a match $m : L \rightarrow G$ embedding L into G such that none of the forbidden patterns in N are present. The transformation is denoted by $G \stackrel{p(\bar{a})}{\Longrightarrow} H$, where $\bar{a} = a_1 \dots a_n$ is the list of actual parameters, i.e., attribute values occurring in G given by $a_i = m(x_i)$. Thus instantiated rule signatures serve as labels.

For example, an application of rule $moveCar(car)$ to the Car at the top of the instance graph in Fig. 2 will lead to a transition labelled $moveCar(1)$ that will transform the graph by replacing the *on* edge of the Car by one pointing to the Road referred to be the second *has* edge of the Path the Car is following. This is possible because the rule's left hand side is matched to the graph as shown by the highlighted nodes and edges in Fig. 2. In particular notice that, given the match of the Car to the one with *id = 1*, the dashed (blue) *on* edge in the rule enforces the matching of the left-hand side Road node in the rule to the Road Car 1 is *on*, while the Path the Car is following determines the other Road node. This is where

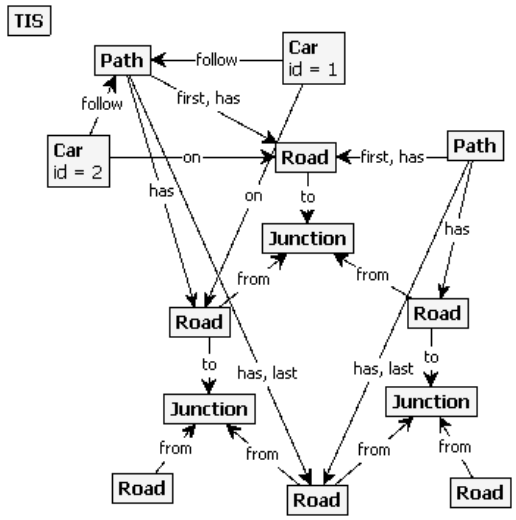
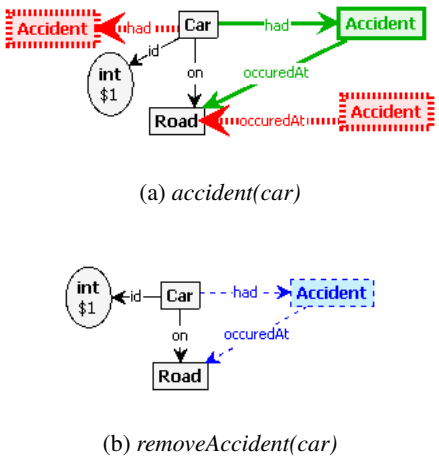


Figure 4: Result graph of applying *moveCar(1)* to the graph in Fig. 2



(a) *accident(car)*

(b) *removeAccident(car)*

Figure 5: Rules modelling the occurrence and removal of Accidents.

the new (solid, green) *on* edge will be pointing. The result of the transformation is shown in Fig. 4.

In general, the transformation follows the algebraic single-pushout approach [7], where the new graph H is constructed from the given G by removing $m(L \setminus R)$ and then embedding into the resulting structure a copy of $R \setminus L$, suitably renamed to avoid name clashes. If $m(L \setminus R)$ contains a node that is connected to an edge in G outside $m(L \setminus R)$, this edge is removed as well. Given a rule and match, the result of the transformation is only determined up to isomorphism of graphs. This principle of *invariance under isomorphisms* is relevant for the scalability of the state space generation because it allows to choose as "the" state any representative of an isomorphism class if graphs—a very effective way of symmetry reduction.

We assume that the actual parameters carry enough information to make transformations deterministic, that is, for transformation steps $G \overset{p(\bar{a})}{\Longrightarrow} H$ and $G \overset{p(\bar{a}')}{\Longrightarrow} H'$, if $a = a'$ then the resulting graphs H and H' are isomorphic.

The complete model of the Traffic Information System consists of the type graph in Fig. 1 and eight rules with the following rule signatures. Parameter declarations refer to the numbered id attributes, with the nth formal parameter referring to the attribute value labelled $\$n$. Rule *moveCar(car)* moves Car from one Road to the next on allocated Path, see Fig. 3; in *accident(car)* the Car suffers an Accident, see Fig. 5(a); in *removeAccident(car)* the Accident is removed, see Fig. 5(b); in *getAccidentInfo(car)* the TIS receives information about an Accident, see Fig. 6(a); in *sendAccidentInfo(car)* the TIS informs the Car about an Accident that happened to another one, see Fig. 6(b); in *detour(car)* the Car, after receiving Accident information, takes a detour leaving current Path and follows another Path, see Fig. 7; in *rejoin(car)* the Car rejoins its previous Path, leaving the detour Path after passing the Accident, see Fig. 8; in *arriveAtDest(car)* the Car arrives at its destination, i.e., the end of its path, see Fig. 9;

Of course, in a simulation of the model the choice of rules (e.g., whether to move or suffer an Accident) and matches (e.g., which Car to move) is taken nondeterministically. We will therefore guide the nondeterministic choice of rules by rates to describe the relative speed of these actions.

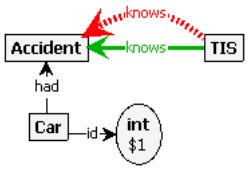
2.2 Performance Modelling

Stochastic graph transformation systems add to each rule a rate, i.e., a positive Real number representing the parameter of the exponential distribution associated with the frequency of execution of the rule (or, equivalently, the reciprocal of the average delay once the rule is enabled for a given match).

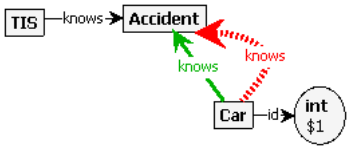
Formally, a stochastic GTS is given by a GTS $\mathcal{G} = (TG, P, \pi)$ together with a mapping $\rho : P \to \mathbf{R}_+$ assigning each rule name its rate. The behaviour of the underlying GTS $\mathcal{G}$ with a given start graph G_0 is expressed by a labelled transition system $LTS(\mathcal{G}, G_0) = (S, L, \Longrightarrow, s^0)$ where the *set of states* S is given by all graphs reachable from the start graph by application of rules, avoiding isomorphic copies (formally, one representative for each isomorphism class of reachable graphs); the *set of labels* L is given by all instantiated rule signatures $p(\bar{a})$; the *transition relation* for all $G, H \in S$ is given by transformations $G \overset{p(\bar{a})}{\Longrightarrow} H$; the *initial state* s^0 is the start graph G_0. In case the state space is small or can be suitably restricted, such a labelled transition system can be generated by GROOVE. With the assignment $\rho : P \to \mathbf{R}_+$ of rates to rule names given, we analyse performance properties by deriving a sequential PEPA process from the transition system.

2.3 Performance Analysis

Following is an extract of the PEPA model generated from the GTS and the start graph presented above, which will be used to describe the syntax and semantics of PEPA informally. The reader is referred to [6] for a formal presentation of the language. Process variables such as $N563261$ correspond to states in the transition system $LTS(\mathcal{G}, G_0)$. An equation like $N563261 = (\text{"}removeAccident(1)\text{"}, removeAccident).N563269$ means that there exists a transition with rate $removeAccident = 8$ from $N563261$ to $N563269$, labelled $removeAccident(1)$. The rates listed at the beginning are to be read with a unit of *times per day*. Operations such as *detour, rejoin, arriveAtDest* represent local computations, e.g., of a navigation system deciding to follow or change to a specific route. On the opposite end of the scale are physical actions such as *moveCar = 288*, representing a delay of 5 minutes for moving a Car to a new stretch of road, or $removeAccident = 8$ representing a delay of 3 hours for



(a) *getAccidentInfo(car)*



(b) *sendAccidentInfo(car)*

Figure 6: Rules for the TIS to receive Accident information and pass it to other Cars

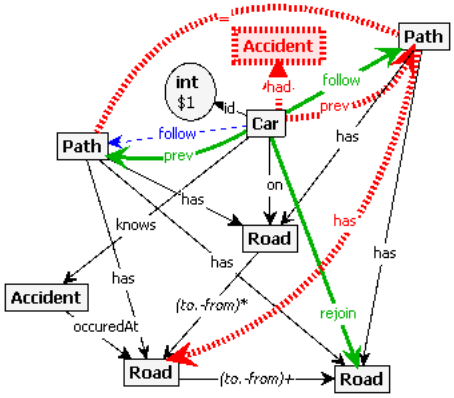


Figure 7: Rule $detour(car)$ shows how a Car takes a detour to avoid an Accident.

clearing an Accident. Actions involving communication with the TIS have rates in between, such as $sendAccidentInfo = 1440$ with a delay of 1 min. The rate for $sendAccidentInfo = 72$ (a 20 minute delay) reflects the fact that in this version of the model traffic information is received from the local radio station, which only broadcasts traffic news at fixed intervals and requires the receiver in the Car to be tuned to the right station.

```
// Rates
accident = 1;              arriveAtDest = 17280;
detour = 17280;            getAccidentInfo = 1440;
moveCar = 288;             rejoin = 17280;
removeAccident = 8;        sendAccidentInfo = 72;

//Processes
N563261 =
    ("removeAccident(1)", removeAccident).N563269+
    ("removeAccident(2)", removeAccident).N563396+
    ("getAccidentInfo(1)", getAccidentInfo).N563357;
N563262 =
    ("removeAccident(1)", removeAccident).N563416+
    ("removeAccident(2)", removeAccident).N563424;

...
```

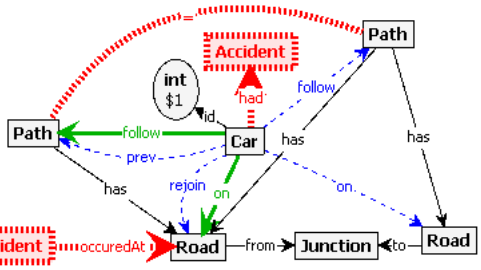


Figure 8: Rule, $rejoin(car)$ shows how a Car rejoins its previous Path.

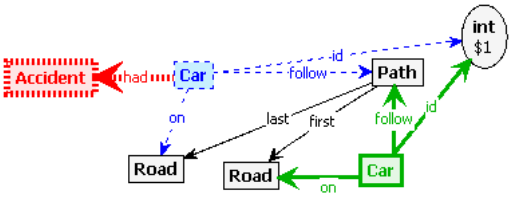


Figure 9: Rule $arriveAtDest(car)$ shows a Car finishing its journey.

Based on the PEPA process we can extract performance measures such as the *steady-state solution* providing long-term probabilities for all states, the *transition throughput* giving the actual long-term frequencies at which transitions are executed, or the *passage-time* between occurrences of specific transitions. An interesting property to be analysed in our model is the throughput of the rule *arriveAtDest(car)*, showing the long-term frequency at which Cars finish their journey. They are replaced by Cars at the start of their trip in order to yield a non-reducible (i.e., strongly connected) CTMC. The throughput is therefore a measure of the long-term average time between these arrival transitions.

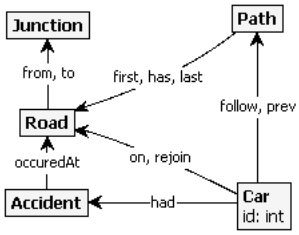
We analyse two different versions of the model, using different rates for rule *sendAccidentInfo(car)* by which the TIS informs Cars about Accidents. This is to evaluate the reduction in journey time possible by subscribing to a more up-to-date traffic information system than the traffic news on the local radio. We assume that a subscription service will provide updates on Accidents on our Path within a minute of being notified, setting the rate from *sendAccidentInfo = 72* to *sendAccidentInfo = 1440*.

The results reported in the table below show that there is indeed an improvement of about 10% in the number of cars completing their journeys per day.

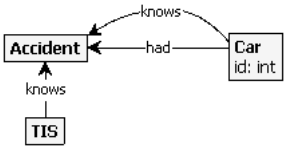
Throughput	Local Radio	Subscription
arriveAtDest	122.7900462	130.9963369

3. VIEW-BASED ANALYSIS

Most approaches to verification based on state space exploration face scalability problems. In the second part of the paper, such problems will be mitigated by separating the model into different views to independently generate their state spaces and derive their PEPA processes. The resulting sequential processes will then be synchronised using PEPA's cooperation operator.



(a) TG_{Car}



(b) $TG_{Service}$

Figure 10: Type graphs of *Car* and *Service* views

3.1 Decomposition of Graphical Models

We distinguish two perspectives, that of the Car and its location and mobility and that of the TIS broadcasting news about Accidents. Given a global model of the system, intuitively a view is defined by identifying in the global type graph the node and edge types that should be abstracted from, such as the TIS in the Car view and Roads and Junctions in the Service view. Start graph and rules are then reduced to the remaining types, removing all instances of types that are no longer present in the smaller type graph.

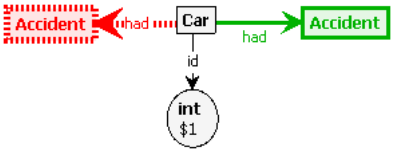
This notion of view of graph transformation system has been introduced in [4]. More formally, from the global type graph TG we define two subgraphs $TG_1, TG_2 \subseteq TG$ such that $TG_1 \cup TG_2 = TG$, in our case TG_{Car} and a $TG_{Service}$. Given a subgraph $TG' \subseteq TG$, a TG-typed instance graph G can be projected to an instance $G' = G|_{TG'}$ of TG' by removing all elements of G whose types are in TG, but not in TG'. The projection extends to rules and transformations [4]. From the start graph in Fig. 2 the project to TG_{Car} just removes the TIS node. Edges of type *knows* would be removed, too, but do not occur in the graph. The projection to $TG_{Service}$ removes all but one TIS and two Car nodes.

Start graphs and rules for the two views are obtained from those of the global model by this projection. Formally, given GTS $\mathcal{G} = (TG, P, \pi)$ with start graph G_0 and $TG' \subseteq TG$, we define $\mathcal{G}|_{TG'} = (TG', P, \pi')$ and G'_0 where

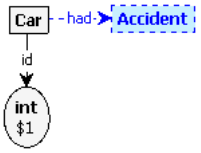
- $G'_0 = G_0|_{TG'}$ is the projected start graph;
- If $\pi(p) = (\bar{x}) : [N]\ L \to R$, then $\pi'(p) = (\bar{x}) : [N|_{TG'}]\ L|_{TG'} \to R|_{TG'}$ is obtained by applying the projection to all graphs of the rule.

With a graph transformation system given as a GROOVE model, we can generate the projection automatically by a transformation on GROOVE's XML representation of rules and graphs.

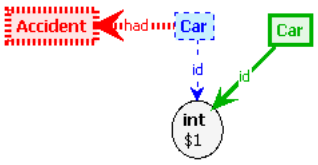
Let us consider what happens when we apply this definition to the global TIS model, using (sub) type graph TG_{Car} in Fig. 10(a) for the projection. Rules *moveCar(car), accident(car), removeAccident(car), rejoin(car)*, and *arriveAtDest(car)* only contain elements of types occurring in TG_{Car}, so they are kept unchanged as part of the Car view, as shown in Fig. 3, 5, 8, and 9.



(a) accident(car)



(b) removeAccident(car)



(c) arriveAtDest(car)

Figure 11: Service view projections of $accident(car)$, $removeAccident(car)$, $arriveAtDest(car)$

Rules *getAccidentInfo(car)* and *sendAccidentInfo(car)* lose all their nodes of type *TIS* and their edges of type *knows*. The remainder rules do not have any effect on the graph and are not needed for synchronisation with rules in the Service view. Rule *detour(car)* in Fig. 7 is retained as is, except for the single *knows* edge, which is not part of the Car view.

For the Service view, type graph $TG_{Service}$ in Fig. 10(b) yields a projection where only rules *getAccidentInfo(car)* and *sendAccidentInfo(car)* are kept unchanged. Rules *moveCar(car), detour(car), rejoin(car)* are reduced to identities without visible effect, while rules *accident(car), removeAccident(car)*, and *arriveAtDest(car)* survive in reduced form, as seen in Fig. 11.

We assume (as happens in our example) that node types equipped with *id* attributes that are used as parameters (just *Car* in our case) are preserved in the projection. This ensures that the actual parameters in labels are preserved and are used consistently in both local views, so that synchronisation over shared labels leads to the correctly composed model in PEPA.

3.2 Synchronisation of Views

After generating sequential PEPA processes from both the Car and the Service view, the resulting processes `Car` and `Service` are synchronised using the PEPA cooperation operator, i.e., as `Car` $\bowtie$ `Service`. That means, transitions carrying labels that are shared between the two processes must be executed simultaneously, while transitions whose labels occur only in one of the two pro-

cesses are independent.

In terms of transition systems, given $LTS_1 = (S_1, L_1, \Longrightarrow_1, s_1^0)$ and $LTS_2 = (S_2, L_2, \Longrightarrow_2, s_2^0)$, their product $LTS_1 \otimes LTS_2 = (S, L, \Longrightarrow, s^0)$ has as states S all pairs of states (s_1, s_2) with $s_i \in S_i$. The set of labels is defined by $L = L_1 \cup L_2$ and the transition relation is the smallest one satisfying

- if $l \in L_1 \setminus L_2$ and $s_1 \stackrel{l}{\Longrightarrow}_1 s_1'$ then $(s_1, s_2) \stackrel{l}{\Longrightarrow} (s_1', s_2)$;
- if $l \in L_2 \setminus L_1$ and $s_2 \stackrel{l}{\Longrightarrow}_2 l s_2'$ then $(s_1, s_2) \stackrel{l}{\Longrightarrow} (s_1, s_2')$;
- if $l \in L_1 \cap L_2$, $s_1 \stackrel{l}{\Longrightarrow}_1 s_1'$ and $s_2 \stackrel{l}{\Longrightarrow}_2 s_2'$ then $(s_1, s_2) \stackrel{l}{\Longrightarrow} (s_1', s_2')$.

The initial state s^0 is (s_1^0, s_2^0), the pair of initial states of the two systems.

If we assume PEPA processes `P1, P2` and let $LTS(\texttt{P})$ be the labelled transition system generated by a PEPA process (ignoring rates for the time being), the product of transition systems is the semantic equivalent of PEPA's cooperation operator, i.e., the reachable portions of $LTS(\texttt{P1} \bowtie \texttt{P2})$ and $LTS(\texttt{P1}) \otimes LTS(\texttt{P2})$ are isomorphic. As a consequence, the two systems are bisimilar. Moreover, a shared label will have the same rate in both sequential processes and the global process, and synchronisation of transitions in PEPA retains that rate. Therefore, the relationship between composition and product carries over to transition systems with rates (or CTMC).

Based on this terminology, we have the following property in support of our methodology.

Proposition (Compositionality) Assume a graph transformation system $\mathcal{G}$ with type graph TG decomposed as $TG = TG_1 \cup TG_2$. Then, the two transition systems $LTS(\mathcal{G}, G_0)$ and $LTS(\mathcal{G}|_{TG_1}, G_0|_{TG_1}) \otimes LTS(\mathcal{G}|_{TG_2}, G_0|_{TG_2})$ are bisimilar.

Proof (Sketch). Given a decomposition of type graphs $TG = TG_1 \cup TG_2$, we can project global steps $G \stackrel{p(\bar{a})}{\Longrightarrow} H$ over TG to local views $TG_i \subseteq TG$, resulting in steps $G_i = G|_{TG_i} \stackrel{p(\bar{a})}{\Longrightarrow} H|_{TG_i} = H_i$ using the corresponding projections of rule p [4].

We write $(G_1, G_2) \sim G$ iff there exist G_i' isomorphic to G_i such that $G_i' = G|_{TG_i}$ and $G_1' \cup G_2' = G$. We show by induction that $\sim$ is a one-to-one correspondence between reachable states in $LTS(\mathcal{G}, G_0|_{TG_1})$ and $LTS(\mathcal{G}|_{TG_1}, G_0|_{TG_1}) \otimes LTS(\mathcal{G}|_{TG_2}, G_0|_{TG_2})$, extending to transitions.

For the start graph, $G_0 \sim (G_0|_{TG_1}, G_0|_{TG_2})$ because $G_0 = G_0|_{TG_1} \cup G_0|_{TG_2}$. Assume graphs $(G_1, G_2) \sim G$. Then $G \stackrel{p(\bar{a})}{\Longrightarrow} H$ in the global LTS yields projections $G_1 \stackrel{p(\bar{a})}{\Longrightarrow} H_1$ and $G_2 \stackrel{p(\bar{a})}{\Longrightarrow} H_2$ such that $(H_1, H_2) \sim H$. Vice versa, local steps $G_1 \stackrel{p(\bar{a})}{\Longrightarrow} H_1$ and $G_2 \stackrel{p(\bar{a})}{\Longrightarrow} H_2$ compose to the global step since they agree in their actions on the shared part. Formally, this is an application of the Distribution Theorem for graph transformation [3]. The key ingredient is the fact that labels determine matches and transformations up to isomorphism. Therefore, two steps in different views with the same label starting from compatible graphs are consistent as distributed steps and can therefore be amalgamated.

Not surprisingly, therefore, the results of analysing the model obtained by synchronising the two projections coincide with those of analysing the global model. What is more interesting is the reduction in the state spaces produced. To witness, the table below shows the size of the global and local views as generated by GROOVE for two and three Cars, respectively.

	Global model	Car view	Service view
2 Cars	756 states	84 states	36 states
3 Cars	out of memory	936 states	1000 states

It turns out that we are unable to generate the LTS from the global system directly, but the synchronisation in PEPA of the LTS generated from the two local views produces a CTMC with 117098 states.

4. CONCLUSIONS

The methodology and case study presented in this paper demonstrate the principle of using graph transformation for modular stochastic analysis. We showed that state space explosion can be mitigated by decomposing the system into views, generating separately their labelled transition systems and synchronising these in the stochastic process algebra PEPA, which also serves as the basis for stochastic analysis. But modularity is not only motivated by scalability concerns. The two views chosen, of physical mobility and services, represent two distinct but related concerns of the system model. Other concerns such as wireless network connectivity or the allocation of emergency services could be added in much the same way. With increasing complexity this will require more dedicated language and tool support to manage a number of views and derive their corresponding projections automatically.

5. REFERENCES

[1] BALSAMO, S., MARCO, A. D., INVERARDI, P., AND SIMEONI, M. Model-based performance prediction in software development: A survey. *IEEE Trans. Software Eng. 30*, 5 (2004), 295–310.

[2] COLLINSON, M., AND PYM, D. Algebra and logic for resource-based systems modelling. *Mathematical Structures in Computer Science 19* (2009), 959–1027. doi:10.1017/S0960129509990077.

[3] EHRIG, H., HECKEL, R., KORFF, M., LÖWE, M., RIBEIRO, L., WAGNER, A., AND CORRADINI, A. Algebraic approaches to graph transformation, Part II: Single pushout approach and comparison with double pushout approach. In *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations*, G. Rozenberg, Ed. World Scientific, 1997, pp. 247–312.

[4] HECKEL, R., ENGELS, G., EHRIG, H., AND TAENTZER, G. A view-based approach to system modelling based on open graph transformation systems. In *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages, and Tools*, G. Engels, H.-J. Kreowski, and G. Rozenberg, Eds. World Scientific, 1999.

[5] HECKEL, R., LAJIOS, G., AND MENGE, S. Stochastic graph transformation systems. *Fundamenta Informaticae 74* (2006).

[6] HILLSTON, J. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[7] LÖWE, M. Algebraic approach to single-pushout graph transformation. *Theoret. Comput. Sci. 109* (1993), 181–224.

[8] LÖWE, M., KORFF, M., AND WAGNER, A. An algebraic framework for the transformation of attributed graphs. In *Term Graph Rewriting: Theory and Practice*, M. R. Sleep, M. J. Plasmeijer, and M. van Eekelen, Eds. John Wiley & Sons Ltd, 1993, ch. 14, pp. 185–199.

[9] RENSINK, A. The GROOVE simulator: A tool for state space generation. In *Applications of Graph Transformations with Industrial Relevance (AGTIVE)* (Berlin, 2004), J. Pfaltz, M. Nagl, and B. Böhlen, Eds., vol. 3062 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 479–485.