I. Troch, F. Breitenecker, eds. ISBN 978-3-901608-35-3

INTRODUCING SOFTWARE COMPONENTS TO ROAD TRAFFIC MODELING AND SIMULATION

P. Zelenka
University of West Bohemia, Czech Republic

Corresponding author: P. Zelenka, University of West Bohemia, Dept. of Computer Science and Engineering
306 14 Plzeň, Univerzitní 8, Czech Republic, pzeli@kiv.zcu.cz

Abstract.

This article presents an application of contemporary software engineering practices, in particular software components, into the area of road traffic modeling and simulation. It shows that by building road traffic models from individual components, both the topology of the simulated traffic system and the behavior of the simulation model can be easily changed according to the requirements. In fact, it is possible to develop a single road traffic model, the properties of which can be adjusted almost arbitrarily. This flexibility, resulting in a multi-adjustable simulation, is of a great advantage as opposed to the existing road traffic modeling and simulation tools that usually implement only one behavioral model and are thus predetermined for simulating a limited set of problems.

1 Introduction

Modeling and simulation of road traffic has gained in popularity in recent years. This is not surprising, because it constitutes the only tool available so far for predicting behavior of a traffic system. Using traffic models, it is possible, for example, to evaluate the impact of the traffic restrictions related to a road work (whether construction or maintenance) and to determine which combinations of road works can be performed simultaneously without causing unnecessary congestion, or to compare the performance of several different road design alternatives. In connection with the recent onset of intelligent traffic management and information systems, traffic models can also be used to test these systems during their development in a real-like environment.

Basically, there are two different approaches to road traffic modeling (see [2], p. 25 ff.). The first one, based on physical theories of fluid dynamics, describes the traffic by differential (or, in the case of computerized models, difference) equations, using physical quantities such as traffic flow and traffic density. Such models are called macroscopic, because they deal with the traffic as a whole. They can be used to evaluate the performance of a traffic system under different conditions, but they are somewhat limited in their capabilities (for example, it is not possible to measure travel times between a given pair of points, or, it is difficult to estimate the effect of making a traffic lane restricted). Moreover, they often suffer from statistically significant inaccuracies once the traffic system gets congested. The second approach makes use of a greater level of detail. It describes the behavior of the individual traffic participants (cars, trucks, streetcars, and so on) by a combination of difference equations and decision trees, using quantities such as acceleration and speed. The quantities describing the traffic as a whole then need to be extracted using statistical measurements. On the other hand, any statistically measurable quantity can be obtained, including distance traveled by a traffic participant or the above-mentioned travel time between a pair of points. Such models are called microscopic and their capabilities are almost unlimited. However, their computational complexity (both in terms of time and space) can be enormous.

2 Software Components

The concept of software components became popular in 1990s, especially in connection with increasing demand for reusability. Object-oriented programming has some mechanisms for supporting reusability (namely inheritance and method overloading), but the level of reusability that can be achieved this way is still far from the level that is usual in other engineering disciplines. Machinery can serve as a great example. Suppose that you need to fasten some things together using a bolt and a nut. Further suppose that you have some bolts left in stock, but no nuts. Because bolt and nuts are standardized (they can be uniquely described by a set of their dimensions), you need to buy only nuts that correspond to the bolts you already have. Moreover, they even do not have to originate from the same manufacturer.

Although machinery and software engineering are indeed not very similar, there is no good reason for software to be different in this respect. Machines are being constructed from components (that is, self-contained operational parts), which in turn can be constructed from subcomponents, and so on. These components are then assembled together in a way that allows for the intended function of the machine. Software is developed in a similar way. The overall intended functionality is decomposed into smaller pieces (top-down approach), which are then designed and implemented (bottom-up approach). In object object-oriented programming, these pieces take the form of objects, whereas in procedural programming they take the form of modules. Either case, they communicate together using method (or function) calls, so that the overall functionality is achieved. The only important difference between

the components on one side and either objects or modules on the other side is thus the formal description of their interfaces (the points where they meet each other).

There are many definitions what a software component is and what it is not. One of the most generally accepted one is that by Szyperski (see [7], p. 548):

> *"A component is a unit of composition with contractually specified interfaces and explicit context dependencies only. [...] A component can be deployed independently and is subject to composition by third parties."*

Because both objects and modules are units of composition and can be deployed independently (in the form of shared libraries, for instance), the fundamental part of this definition are the *"contractually specified interfaces"*. It means that a software component needs to explicitly define any functionality it provides as well as any functionality it requires from other components. For this purpose, a reasonable choice is Meyer's Design by Contract (DbC) principle (see [4]). Using DbC, the semantics of an operation (represented by a method or function) can be described by a union of preconditions, postconditions, and invariants. Preconditions describe the state expected before the operation can be used (that is, what the caller of the method or function must provide and what the provider of the implementation may rely on). Conversely, postconditions describe the state expected after the operation was used (that is, what the implementer must provide and what the caller of the method may rely on if he has fulfilled the corresponding preconditions). Finally, invariants describe the state that will not be affected by the operation itself and are therefore used as consistency constraints. These explicit statements represent a contract between the provider and the user of the component and allow for the above-mentioned third-party composition (where the first party set up the contract and the second party provided the implementation).

Contractually specified interfaces also have an interesting implication. Any two components implementing the same contract can be substituted one for the other (see [7], p. 83 ff.). In other words, replacing one of them with the other does not require any changes in the rest of the software. In fact, unless the components representing the rest of the software are explicitly told, they are not able to learn about such a change. Components thus make the functionality of the software much more flexible and adaptable to changes.

Implementing the same contract does not mean that the components must perform the functionality in exactly the same way. For example, consider a component dealing with logging. Such a component is expected to make records of important events that have occurred, but the resulting log may differ in the recorded level of detail (critical errors only or a full spectrum of events including debugging information) or even in its form (text file, binary file, or a set of records in a database). Another great example of a set of operations implemented in a possibly different ways are database connectors, which provide a uniform interface for performing database operations in order to allow for easy change of the underlying database engine.

3 Components in Road Traffic Models

In theory, both macroscopic and microscopic road traffic models are able to make use of the advantages of the component approach. Nevertheless, using software components for macroscopic modeling is much more reasonable in connection with hybrid models (see Section 3.3) than for standalone macroscopic models.

Every traffic system can be decomposed into two different aspects: its topology (the layout of road segments and intersections) and its behavior (the way in which the state of the system is changed). These two aspect can be modeled each by a separate set of components.

3.1 Road Topology Change Using Substitution of Components

Road topology, constituted by so-called structural components, is more or less static, at least in the sense that it will usually not change during a simulation run. However, it is important that the topology can be modified easily in order to test and compare different road design alternatives. So, an appropriate decomposition of the road topology is needed.

Graph theory can help a lot in identifying structural components. Consider the following graph representation: each intersection (that is, a place where three or more road segments meet) corresponds to a vertex and each road segment corresponds to an edge. Then each vertex and each edge shall be modeled by a separate structural component (see Figure 1). In such a topology, components can be easily substituted. For example, a crossroad can be replaced by a roundabout (see Figure 2). More generally, in the terms of graph theory, any connected subgraph can be replaced by another connected subgraph.

3.2 Model Behavior Change Using Substitution of Components

Microscopic road traffic simulation models consist of several submodels, where each submodel handles a specific task in the simulation (see [5], p. 5). The most essential submodels are the car-following model and the lane-changing model. The car-following model describes movements of a vehicle in response to the behavior of the

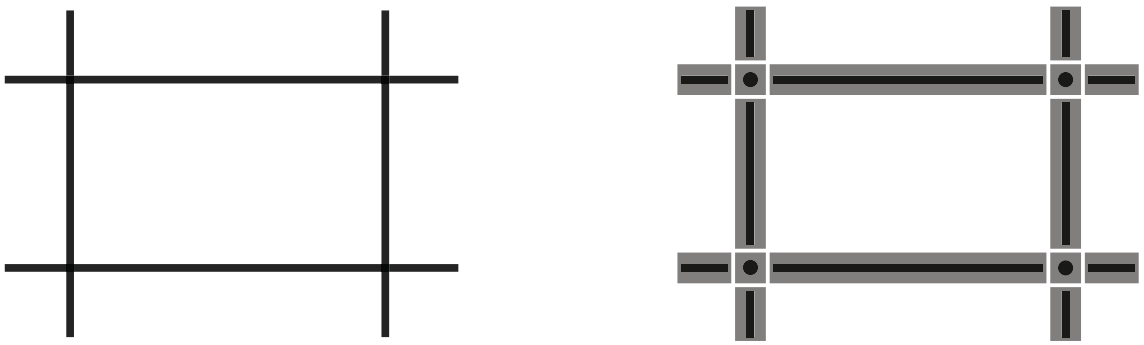

Figure 1: An example of using graph theory for identification of structural components. Here, the original road topology consisting of four crossroads (left) and its decomposition with structural components highlighted in grey (right).

preceding vehicle traveling in the same lane, whereas the lane-changing model describes movements of a vehicle in connection with a lane change (which may be necessary for turning at the next intersection, for instance). There are many different car-following models (Gazis-Herman-Rothery model, Gipps model, or Fritzsche model, to name a few), each taking a slightly different set of aspects into account (Fritzsche model considers driver's reaction time, while most other models do not, for instance; for more details, see [5] or [8]). These differences may seem to be small, however, depending on the problem being simulated, they can affect the results in a statistically significant way. The same applies to the lane-changing models, as well as to all other models not yet mentioned, such as overtaking models or yielding models.

Existing microscopic road traffic modeling and simulation tools (for example, Aimsun, Corsim, Mitsim, Paramics, or Vissim) usually select one from these models and stick to it in all cases. This limits their range of application, because, as said above, not all models are suitable for simulating a particular problem. Extracting each such model to a separate component (this time called behavioral component) allows to make use of the component substitutability described in Section 2. We can then select a set of models that are expected to provide the best results for a particular purpose.

Further, we can use several different models at the same time. This can be particularly useful when different classes of traffic participants shall behave differently. For example, streetcars cannot travel off the track, so they do not need a lane-changing model (track junctions can be simulated much more easily). Or, in most countries, streetcars have right of way in all situations, so the other traffic participants are required to yield. However, Germany is a notable exception. German traffic law considers a streetcar to be an ordinary traffic participant in this respect. As a consequence, yielding models for streetcars and the rest of traffic participants will be different, except where a German traffic system is simulated. Such changes in behavior are difficult to achieve in most of the existing road traffic models.

It is also obvious that although the traffic participants belonging to the same class share the same behavior, they can differ in some parameters. Unfortunately, these parameters may affect the behavior considerably. For example, mass and braking efficiency affect the braking distance. As a result, traffic participants may need to start braking at a different time if they need to stop at a given point. This means that the behavioral components need to be

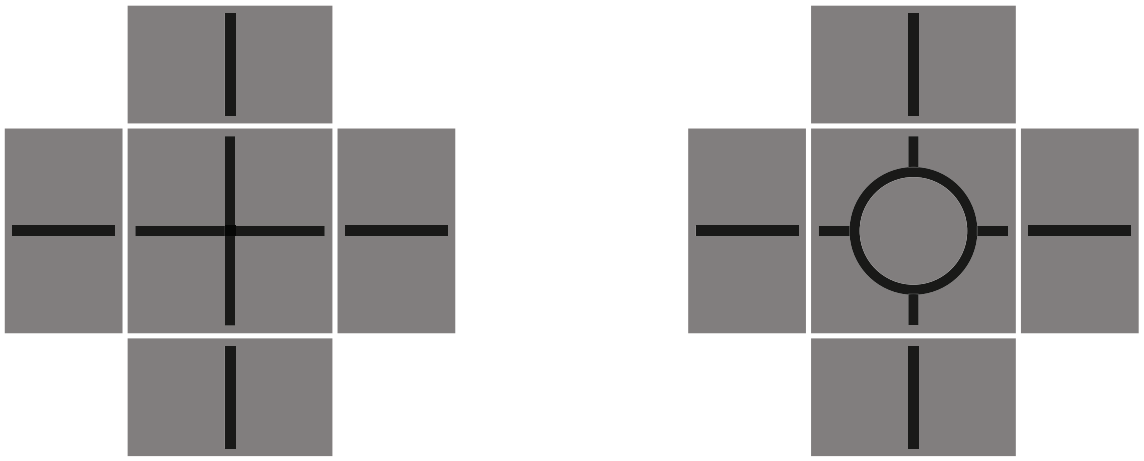

Figure 2: The topology of a traffic system can be changed easily using substitution of structural components. Here, a crossroad (left) has been replaced by a roundabout (right).

parametrized and each traffic participant needs its own instance with parameters set to the corresponding values. However, there are many traffic participants in a typical microscopic road traffic simulation and creating an instance of all the necessary behavioral components for each of them would cause an extremely high consumption of memory. In such situations, the flyweight design pattern (see [1]) comes in handy. The idea of flyweight consists in extracting the shared part of two or more entities into a new entity and referencing this new entity from the original ones. In our case, a traffic participant would contain only a set of its own parameter values and a reference to the corresponding behavioral model (see Figure 3).

3.3 Multiple-Level-of-Details Modeling Using Component Adapters

So far, we have considered component substitution only when both the substituted and the substituting components implemented the same contract. In fact, any two components can be substituted one for the other provided that there is an adapter able to overcome the differences between their contracts. An adapter in this sense is actually an intermediate component (sometimes also called a connector) that implements both contracts and provides a logic to translate from one to the other. In other words, an adapter serves as an interpreter between components that could not otherwise communicate.

In a road traffic simulation, the impact of a traffic restriction (or any other traffic control measure) typically diminishes with increasing distance from its point of action. So, if a point of action is located somewhere in the heart of downtown, there is little interest in results from a distant suburbs. However, even these areas still need to be simulated, because they are an integral part of the traffic system and can therefore influence the areas of interest over time. But they indeed can be simulated at a much lower level of detail, for instance using a macroscopic model. Doing so can substantially reduce the computational intensity of the overall simulation.

Models capable of switching between the microscopic and the macroscopic level of details are also called hybrid. An introduction to the problematics of hybrid traffic modeling can be found in [3].

Implementing the adapters between microscopic and macroscopic models is actually not difficult at all. The only important difference is that macroscopic models work in terms of physical quantities, whereas microscopic models work in terms of individual traffic participants. As a result, adapters from microscopic to macroscopic models need to estimate the values of the respective physical quantities (traffic flow and traffic density). Traffic flow is defined as the number of vehicles per a unit of time, so the corresponding value can be obtained using a counter and a timer. The counter is increased by every traffic participant passing through the adapter. The timer then periodically calculates the value of the traffic flow and to resets the counter to zero. Traffic density is defined as the number of vehicles per a unit of length, which is a little bit more problematic to measure. The reason is that adapters do not represent a road segment or any other piece of road topology and are thus dimensionless. Fortunately, traffic density can be calculated from the traffic flow and the mean speed, which is measurable much more easily. It shall be obvious that during this conversion, a lot of information, such as the classes of traffic participants or their parameters, is lost. However, it does not constitute a problem, because this conversion is only done in the direction to the areas out of interest and the lost information will not be needed anymore (see Figure 4). Conversely, adapters from macroscopic to microscopic models need to generate the individual traffic participants according to the values of traffic flow and traffic density. This can be done using a pseudorandom generator parametrized by these values. The additional information about the traffic participants that either has been lost during the reverse conversion or has never existed needs to be generated at this moment. As a consequence, these adapters need additional sources

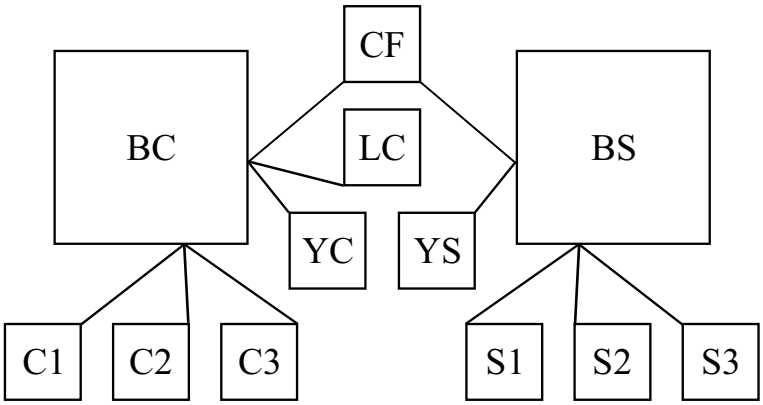

Figure 3: An example of applying the flyweight design pattern to behavioral components. Here, three cars (C1,C2,C3) and three streetcars (S1,S2,S3), each referencing the corresponding behavioral model (BC, BS). Both models share the car-following model (CF) and each has its own yielding model (YC, YS). In addition, the behavioral model for cars has a lane-changing model (LC), whereas the behavioral model for streetcars has none.

of information, such as the probability of each class of traffic participants in the system. For an example of a boundary between macroscopic and microscopic models, see Figure 5.

At first glance, it may seem that the process of adapting between microscopic and macroscopic models can affect significantly the results of the simulation. But this is not necessarily true. In an effort to reduce the amount of information transferred between microscopic road traffic models in a distributed computing environment, Potužák (see [6], p. 3 ff.) has studied the process of extracting the macroscopic quantities and using them for generating the traffic participants again. He showed that the statistical deviation caused by this process can be kept under 5 %.

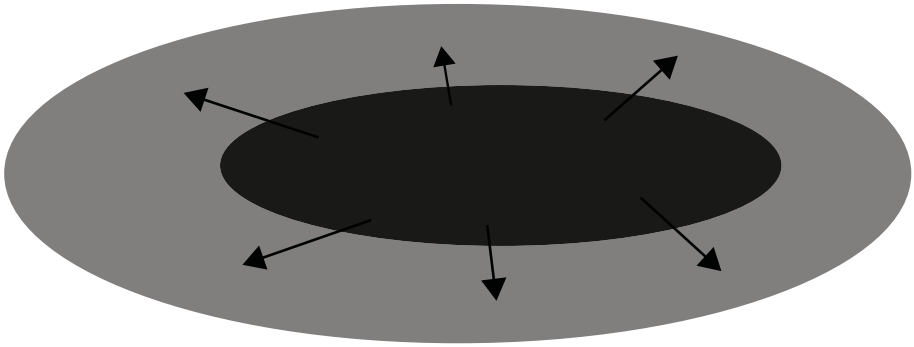

Figure 4: An example of a hybrid simulation model. The area of interest (dark grey) is simulated using a microscopic simulation model, whereas the area out of interest (light grey) is simulated using a macroscopic simulation model. The arrows show the direction in which the information is lost in microscopic-to-macroscopic adapters.

3.4 Transparent Distributability Using Component Adapters

Even if the areas out of interest are simulated using a macroscopic model, the overall computational intensity may still be too high for a single computer. In such situations, a distributed computing environment may be necessary to get the simulation results in a reasonable time. But this also means that the simulation model needs to be adapted for a distributed environment. In particular, the model must be divided into parts (in this case, disjoint subsets of structural components), it must be decided which part will be assigned to which node, and remote connections between the parts must be established.

This is another situation where software components can help. The remote connections can be made transparent to the components using remoting adapters, that is, connectors that encapsulate the additional logic needed for remote communication. In remote procedure call technologies, they are typically called stubs and skeletons. Because adapters can be placed between any two components in general, there are many places where the simulation model can be divided into parts (see Section 3.2). This fact could be utilized to implement some load-balancing mechanism.

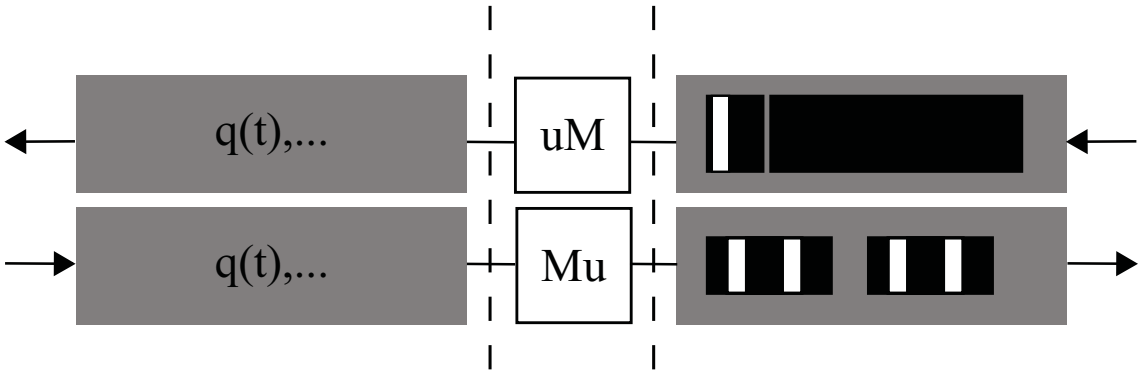

Figure 5: An example of a boundary between macroscopic and microscopic models. Here, the boundary lies within a road segment consisting of one lane in each direction. In the top half of the figure, the microscopic structural component (right) is connected to the macroscopic structural component (left) through an microscopic-to-macroscopic adapter (uM). Conversely, in the bottom half of the figure, the macroscopic structural component (left) is connected to the microscopic structural component (right) through an macroscopic-to-microscopic adapter (Mu).

4 Summary

In this paper, we dealt with the possibility of using software components in the area of road traffic modeling and simulation. We showed that building road traffic models from individual components can be of a great advantage and can lead to development of a single multi-adjustable simulation model. First, we introduced the concept of structural components and outlined how their substitution can be used to change the topology of the simulated traffic system. Then, we applied the same principle to the behavior of the model and showed that by introducing behavioral components, several different models can be used at the same time, each modeling a specific task. Further, we outlined adapters between microscopic and macroscopic models and their use for simulating different parts of the traffic systems at a different level of detail. Finally, we showed that component approach in connection with remoting adapters provide for an easy adaptation of the model for a distributed environment, possibly with some load-balancing mechanism.

The experience gained from this research and the still ongoing research of structural components will be used to develop an experimental multi-adjustable road traffic model.

Acknowledgment

This work was supported by the Grant Agency of the Czech Republic (project number 201/08/0266).

5 References

[1] Gamma, E., et al.: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, USA, 1994.

[2] Hämäläinen, A.: *Studies of Traffic Situations Using Cellular Automata*. PhD dissertation, Helsinki University of Technology, Espoo, Finland, 2006.

[3] Hartman, D.: *Switching Scalability of Hybrid Model for Complex Traffic Simulation*. PhD dissertation, University of West Bohemia, Plzeň, Czech Republic, 2008.

[4] Meyer, B.: Design by Contract. In *Advances in Object-Oriented Software Engineering*, Prentice Hall, Englewood Cliffs, NJ, USA, 1991.

[5] Olstam, J. J. and Tapani, A.: *Comparison of Car-Following Models*. Swedish National Road and Transport Research Institute, Linköping, Sweden, 2004.

[6] Potužák, T.: Distributed Traffic Simulation and the Reduction of Inter-Process Communication Using Traffic Flow Characteristics Transfer. In *Proceedings of the Tenth International Conference on Computer Modelling and Simulation*, Cambridge, UK, 2008.

[7] Szyperski, C.: *Component Software: Beyond Object-Oriented Programming*. Second Edition. Addison-Wesley, Reading, MA, USA, 2002.

[8] Zhang, Y.: *Scalability of Car-Following and Lane-Changing Models in Microscopic Traffic Simulation Systems*. Master's thesis, Louisiana State University, Baton Rouge, LA, USA 2004.