# Formal Specification of 'Traffic'

ir. Frank van den Berk

*Will Formal Specification Methods bring heaven on earth? Frank van den Berk, ex-OOTI and currently working for software house BSO/Origin, gives his reflections on the value and limitations of formal methods for specifying large software systems. To demonstrate his points of view, he uses the metaphore of specifying traffic rules.*

In my first year at BSO/Origin I had several discussions about Formal Specification Methods (FSMs) with my colleagues. In these discussions there was a large gap between 'theory' and 'practice' and quite often the people involved did not really understand each others arguments.

In this article I will try to clarify both the 'practical' and the 'theoretical' arguments with a hypothetical, metaphorical case study that is familiar to us all. Hopefully this case study will narrow the gap a little. The case: specifying and implementing a system called 'Traffic', which indeed corresponds to the streets, cars, pedestrians, bicycles, etc., that we all know.

## The case

We assume that some engineers on some other planet have never seen traffic and that their prime minister has ordered them to build a system in which various kinds of vehicles and pedestrians should be able to travel from A to B in two dimensions and in a safe way. The system should be ready and operational within six months.

In the specification phase of the Traffic project, a few engineers are involved. They make some rough drawings of a street plan, think that it will work, and start to build some roads. When the first vehicles are released, the results are disappointing. The inhabitants of our planet are a little aggressive by nature, drive much too fast, and cause a lot of accidents.

The engineers are desperate and decide to phone the Computing Science department of the local university for help. The scientists tell the engineers that they should use mathematics to specify their Traffic system, and teach them to use Formal Specification Methods.

With these FSMs, formal rules are constructed that specify the conditions under which a transformation of a vehicle from position X to position Y can take place. After some experiments and a lot of mathematical reasoning the engineers have constructed a system of which they can prove that under no circumstances an accident will happen.

The engineers are very enthusiastic and immediately start to build Traffic, because they only have four months left. Each person involved in Traffic gets a copy of the formal specification of Traffic. The rules of this specification are called 'Traffic Rules'.The results of their work are familiar to us all. Just take a look outside your window.

## Formal specifications are good

Did the use of formal traffic rules help in this case? Of course it did. Everyone would agree that a formal rule like: ''A car on a cross road that comes from the right has right of way'' is much more effective than an informal rule like: ''You should avoid to bump into another car''. Why? Simply because a formal rule can only be interpreted in one way.

This is also true for software engineering. A software engineering project that is based on a formal specification will be much more efficient than one based on an informal specification. In areas like Software Quality the importance of formal rules is already fully accepted and understood. Informal software quality attributes, i.e., attributes that can not be quantified nor measured, are regarded useless in this field.

In most software engineering projects today, however, specifications and designs are still informal. Why do the people involved manage to build soft-

ware from these specifications and designs that actually works? To answer this question, we return to our case. Let's suppose that the traffic system is built without formal traffic rules. How can accidents be prevented in this case?

The obvious answer is that we should only use drivers that are skilled, experienced, and feel responsible. Skilled and experienced drivers know the way and are familiar with their car and with all dangerous locations along the road. They will do everything they can to prevent accidents from happening. In their long driving history they may even have developed some sort of code, such as: ''When I blink my head lights you may go first''. New drivers should be slowly introduced in such a traffic system, and should be coached by an experienced driver.

In software houses we see the same kind of situation. Experience, both with soft- and hardware and in the application area of the client, is a very important asset. Getting the 'right men on the right place' is crucial for the failure or success of a software engineering project. New employees are slowly introduced in this 'software engineering world'.

Relying on experience, however, has its limits. In very complex and crowded traffic systems experience alone will not be sufficient. Formal traffic rules are needed to prevent accidents or situations where everyone is waiting for each other. The same is true for software systems. Experience alone is not sufficient when faced with a complex system in which many people, information and processes are involved. That is why future use of formal methods is inevitable.

## Formal specifications are bad

Did the use of formal rules result in a perfect traffic system? The answer, of course, is 'no'. We notice every day that specifying formal traffic rules only has a limited effect on safety. There are various reasons for this, reasons that are obvious for most of us because we all participate in traffic each day.

In the specification, a lot of relevant aspects of traffic are left out because the engineers simply did not anticipate these aspects beforehand and thought that their abstractions were justifiable. A nice example in real traffic is the fact that there is a formal rule for the maximum speed of a car, but there is no formal rule for the minimum distance between two cars. The designers of the traffic rules simply did not foresee the density of traffic today. As a consequence, all cars are equipped with an instrument that measures speed but hardly any car is equipped with an instrument that measures distance. Moreover, the police very intensively check your speed, but hardly ever checks your distance. And still a lot of head-to-tail collisions are caused by a combination of speed and distance and not by speed alone.

When specifying software engineering systems, the same kind of mistakes are made. Abstraction from CPU speed and available resources, for example, may be elegant and useful in the specification; it can be very annoying in the implementation.

In the discussion above we have seen that 'human factors' like experience play an important role in the success of a software engineering project. Unfortunately human factors can also play an important role in the failure of a software engineering project. We also see this in everyday traffic. Making formal traffic rules, for example, does not mean that everyone knows or understands those rules or that no-one occasionally forgets a rule or makes a mistake. Sometimes it is even necessary to 'bend the rules' a little, to prevent an accident from happening. Luckily, violating the rules in traffic only has local effects...

A very important factor in traffic accidents is time-pressure. Drivers that are under pressure will drive too fast, make mistakes and violate traffic rules. This, of course, is also the case in software engineering projects. Making formal rules can be rather time consuming. Take, for instance, a rule like: ''Traffic that comes from the right on a crossroad has right of way''. This rule illustrates the difficulty in formulating formal rules. For what exactly is traffic? And what exactly is a crossroad? And are there any exceptions to this rule? Making informal rules is much easier.

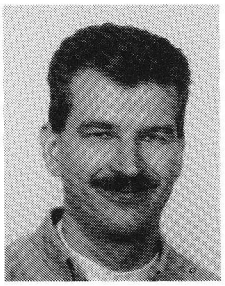
## Conclusion

I personally think that a lot of arguments in the discussions about Formal Specification Methods also appear in the Traffic case. In this case we can see that both sides are right.

Yes, formal traffic rules are not the ultimate solution to traffic accidents, and yes, without formal

traffic rules much more accidents would happen.

For software systems, using more formal techniques will be inevitable to be able to control the growing complexity of these systems. Just like in traffic, formal rules are much more effective, but harder to make, than informal ones. But even with formal techniques, systems will still crash and bugs will still need to be fixed. We can see this everyday we are on the road... □



*Ir. Frank van den Berk completed the post-masters programme Software Technology in 1993 and is currently working for BSO/Origin in Eindhoven. He is a member of* XOOTIC *and co-editor of* XOOTIC MAGAZINE.

# Agenda 1994

**X-Day**
This annual event for all XOOTIC members combines the Annual Members Meeting and the election of new chairpersons with a social get-together. A bus will drive participants up and down from Eindhoven. Afterwards a Chinese-Indonesian buffet in Arnhem will be offered.
Date: Saturday April 16th, 1994, 11.00 - 21.00 h.
Place: National Park 'De Hoge Veluwe'.
Organization: XOOTIC.

**Celebration Dies Natalis EUT**
Eindhoven University celebrates its 38th Dies Natalis. On the occasion of this celebration, prof.dr. Martin Rem will give a speech with the title 'Energy-efficient computing'.
Date: Friday April 22th, 1994, 16.00 h.
Place: Eindhoven University of Technology, Auditorium.
Organization: EUT.

**Trip to Berlin**
It starts to become a tradition that XOOTIC organizes every year a long weekend to a European city. After successful trips to Paris and Prague, it was decided that this year's trip will go to Berlin.
Date: May 12th till May 16th, 1994.
Place: Berlin, Germany.
Organization: XOOTIC.

**VIE-PAO Seminar on Connecting Databases**
Dr. A.T.M.(Ad) Aerts (EUT) will address the problem of how to connect various local databases. This problem arises for example after the merger of two companies. The speaker will discuss recent developments in the field of distributed, federalized, and multi-database systems. These systems will be evaluated on three criteria: distribution, heterogenity, and autonomy.
Maximal 24 participants, of which maximal 8 VIE members.
Date: Wednesday May 18th, 1994, 16.00 - 20.00 h.
Place: Aristo Zalencentrum, Utrecht.
Organization: PAO Informatica together with VIE.
Costs: NLG 150,=, VIE members NLG 75,= (includes dinner).

**IVO graduation ceremony**
IVO students (including OOTIs) hope to receive their graduation certificate on this day. Following on the ceremony, a drink will be held.
Date: Thursday September 15th, 1994.
Place: Eindhoven University of Technology, Auditorium.
Organization: IVO.