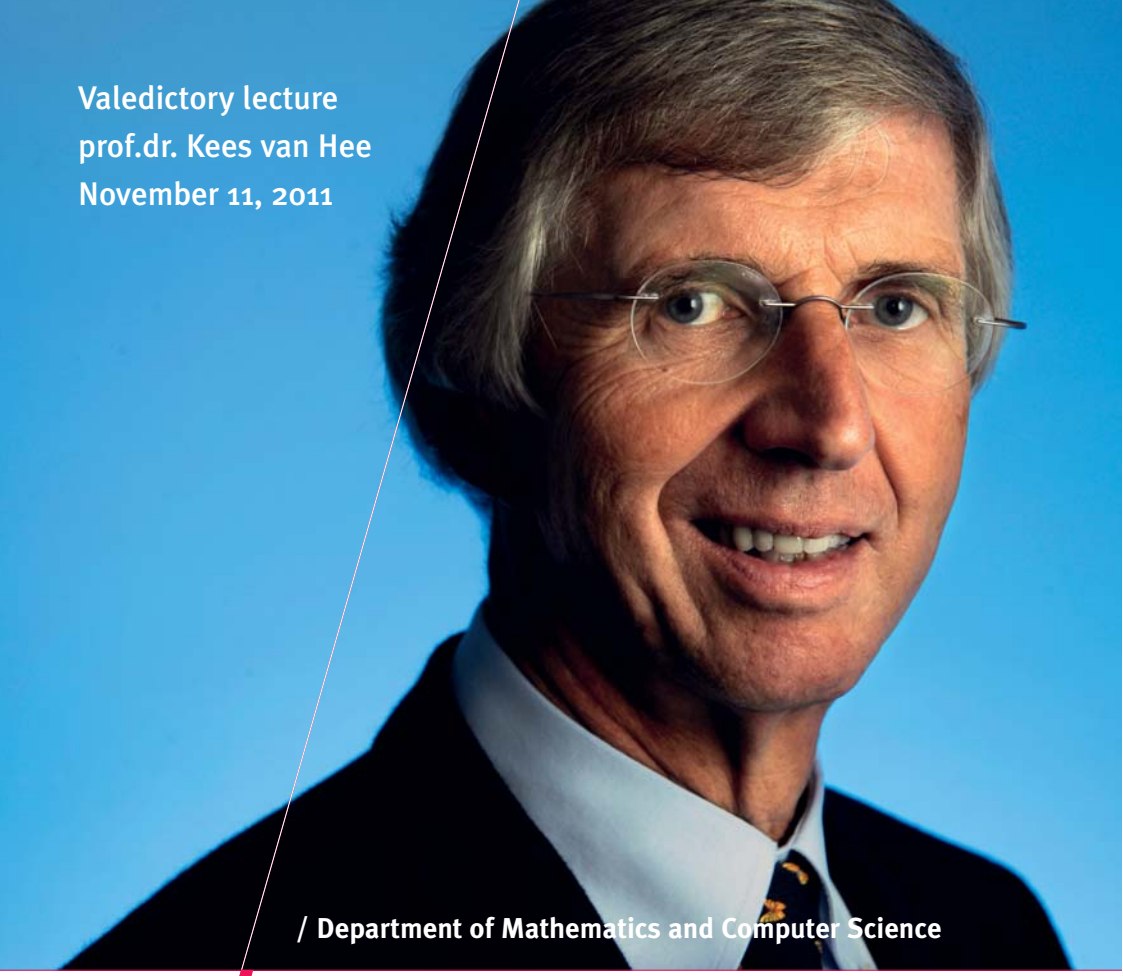
Model engineering : balancing between virtuality and reality

Valedictory lecture
prof.dr. Kees van Hee
November 11, 2011



/ Department of Mathematics and Computer Science

TU/e Technische Universiteit **Eindhoven** University of Technology

Model Engineering: Balancing between Virtuality and Reality

Where innovation starts

Valedictory lecture Prof.dr. Kees van Hee

Model Engineering: Balancing between Virtuality and Reality

Presented on November 11, 2011
at the Eindhoven University of Technology

Introduction

Model *engineering* is the *synthesis* of models and their *analysis* to solve real-world problems. We make a *model* of a *system* to derive information about that system by analyzing its model. The term 'system' is used here for any 'complex thing'. Systems can be man-made, which we call *artifacts*, or they can occur in nature. The elements of a system may be material or immaterial things. Examples of systems are an airplane, a biological cell, the metabolism of a mammal, a software package, a business process or the set of laws to run a country. A *model* is used as a *representation* of a *system*. Models can have different forms. For example we have physical models such as a scale model of an airplane.

I shall restrict myself to *mathematical* or *symbolic* models and *computer* models. The latter are representations of symbolic models in a computer language so that a computer can manipulate the model, for instance by simulating the modeled behavior. In making a model we have to map the elements of the system we want to model onto the elements, often symbols, of the model. In fact a model is a virtual image of a part of a real-world system. The main assumption of model engineering is: *The properties we can derive from the model are also valid for the real system*.
This may all sound a very lofty activity, but everybody uses mathematical models every day. The most well-known model is *counting*. If we have a flock of sheep and we would like to know how many sheep there are, we make a mapping from the sheep to the natural numbers, consecutively, 1,2,3,.. until each sheep has a number. This is fairly easy if they don't move, but counting many moving objects is difficult. Probably the second most commonly used model is an *economical calculation* represented in a spreadsheet where we map the products we make to the money it costs to produce them and the sales activities to the value of the orders. So we arrive at the profit we make by subtracting the total cost from the total revenue of sales. In the first example we were only interested in the *number* of sheep and in the second in the *amount* of money we could earn. Other day-to-day examples of models are a geographical map and a floor plan of a building. The questions we want to answer determine the type of model we will have to use.

For advanced model engineering the availability of fast computers and of data in digital form are essential. When I started my career, computers were brand new devices. They were extremely slow compared to modern computers and their data storage capacity has since then increased by a magnitude of 10^9. Hardly any data were available in digital form. Many of the model engineering techniques we apply today were completely unfeasible when I started my career. So I have experienced an evolution in model engineering comparable with the evolution from traveling in a horse-drawn carriage to traveling in a space shuttle (see [0]).

It is a good tradition that retiring professors give a valedictory lecture in which they look back on their working life and summarize the lessons they have learned. I will conform to this tradition. In total I spent 16 years of my career in the consultancy business and 26 years in the academic world and during 10 of those 16 years in consultancy I was a part-time professor. In all those years model engineering played an important role in my life. In this lecture I will reflect on model engineering and I will use my rather unusual career path to structure this lecture.

It is a bad tradition that retiring professors like to give direction to future research or to criticize the academic system they have been part of. That is the job of their successors. I shall only use my experience to share some ideas about the future with you.

Prelude

My interest in science started at high school where I was inspired by very good teachers in mathematics and physics. I liked physics more than mathematics. Physicists make models of parts of the real world and they analyze these models to understand or to predict phenomena. This is what *model engineers* do. Mathematicians like to answer generic questions of a modeling framework, such as “is there always a solution?”, “is there a best solution?”, “is it unique?”and “is this the fastest method to arrive at a solution?”. I like these questions, but I like making models to solve practical problems better. So I consider myself more as a model engineer than a mathematician.

I started my Bachelor study at Leiden University with a first major in physics, a second in mathematics, and a minor in astronomy. I was discouraged by the bad laboratory facilities, my own clumsiness in these labs and the way physics was taught in Leiden. In contrast mathematics was taught very well, so I swapped physics for mathematics. I had no idea what to become later, but I did not like the idea of becoming a high school teacher. In those days some interesting developments occurred: the mathematics department of Leiden University was extended with chairs in probability theory, statistics and numerical analysis using computers and a chair in what we now would call the science of programming. In particular the last two were brand new scientific topics with great expectations about their applicability. Until then mathematics was mainly used in physical sciences, but now it was also being applied to economic phenomena. Also econometrics had just started as a new discipline, though unfortunately not in Leiden. For my Master’s I chose applied mathematics as my major and I arranged with some friends a special minor in economy. I learned a lot during my last two years as student-assistant in statistics with Prof. Willem van Zwet.

When I graduated I had no ambition to become a scientist. But my wife was still studying in Leiden and teaching at a university meant not having to do military service. So when I was offered a job in computer science by Prof. Alexander Verrijn Stuart in Leiden I gratefully accepted it. I had to teach many courses, in particular about programming languages and simulation techniques. Although I saw many challenges in computer science, I was more interested in applied mathematics and

particulary in Markov chains. After two and a half years I moved to Eindhoven for a PhD position in Markov Decision Processes with Prof. Jaap Wessels. I worked on how to control these processes in case the underlying transition mechanism should have unknown parameters that have to be estimated during the course of the system. So I had to deal with the dilemma of using the estimates of the parameters for optimal control, versus obtaining more information about these parameters. I made my own modeling framework and I analyzed it as far as I could. I could solve problems that were a magnitude larger than the state-of-the-art, but they were still so-called toy problems. With today's computers and the availability of operational data in digital form, these techniques could be applied to real life problems. In addition to working on my thesis I did some research in other areas of the field of Markov Decision Processes.

While I enjoyed that period very much, when I finished my thesis I did not accept an attractive offer to stay at the university because I wanted to put mathematics, in particular operations research and statistics, into practice. So in 1978 I left the ivory tower where we were able to work in a virtual world where we invented the problems ourselves. I started as deputy director of a small consultancy firm, AKB (Adviesbureau voor Kwaliteitsbeleid en Besliskunde) in Rotterdam.

Model engineering in practice

Most of the practical experience in model engineering I obtained during my AKB period. At that time AKB was the only firm in the Netherlands that offered model engineering to answer business questions. AKB was owned by a larger consultancy organization with about 400 employees, called Bouwcentrum, where the focus was on the building and construction industry. AKB had a wider scope: it functioned as the computer center for Bouwcentrum and it was involved in almost all the Bouwcentrum projects where mathematical models played a part. AKB's rich history started in 1946. One of the most important projects was the construction of a new measurement system for fashion. It was much more efficient than the existing systems at the time, but the system was never introduced because a number of major fashion firms refused to make the switch. During the first half year at AKB my coach was Prof. Hans Sittig, who retired as director of AKB. He was a self-made scientist and without a MSc or a PhD he became part-time professor at Erasmus University. He was taught statistics by the famous Prof. Van Dantzig during WWII when both had to go underground. Sittig's mathematical toolkit was rather limited, but what he could do with it to solve business problems was really amazing. It was a very exciting period in which I learned a lot about model engineering. This was one of the best learning experiences for me: it was a very good business school training as well as training in model engineering.

I shall now briefly sketch some characteristic model engineering projects we did at AKB. Of course in most cases I did this with some colleagues. The first two cases occurred at AKB before my time, but are interesting for their 'side effects'.

1. *The grade-day system*
   In times that houses used oil for heating, the oil companies had to have oil trucks available to supply houses that ran out of stock during the weekends. AKB constructed a model to forecast the oil consumption of all clients. This dramatically reduced the number of trucks needed. The model was based on simple regression techniques and used the daily recorded temperatures. Only once there was a problem: one house repeatedly ran out of oil whereas all other houses had no such problem. Analysis showed that the client drove a diesel powered car and he used, illegally, heating oil to fuel his car.

2. *Biscuit bakery*
   A large bakery had a problem with their biscuit production. The problem was that some biscuits were broken when they left the production line. The analysts of AKB made a model of the production process and did a statistical analysis. So they found out that one mould for the biscuit production had a burr. This problem was solved! However the client was not pleased because now he had too few broken biscuits to supply the demand for bread-crumbs, so they had to break good biscuits to fulfill this demand.
3. *Standardization for the building industry*
   One of my first jobs at AKB was for the Dutch Normalization Institute (NEN). I made proposals for new standards for the height of doors and for the size of a shower space. The door height standard was based on measurements of candidates for military service. A complication was that the age at which these candidates were measured had changed. The model was simple: first we estimated the growth trend by linear regression and then we extrapolated it to 2020. Next we determined a one-dimensional normal distribution with the extrapolated mean value and the current coefficient of variation. Finally a 95% confidence upper bound of 210 cm was calculated. For the showers we obtained data from experiments and we used a 4-dimensional normal distribution to obtain the standards. (I believe the fourth dimension was the orientation in the shower). That was a hell of a job using only a hand calculator!
4. *Glass plate cutting*
   A small software company had offered the development of a software package to a cluster of eight glass factories. The software would now be called an ERP system. Part of the package was a module that could determine the optimal plan to cut large so-called mother panes into small window panes. This job was too difficult for them so they called on some experts at the TU Delft. Although they arrived at some kind of solution, based on integer linear programming, the task ran for more than an hour on a large computer, while the software house had offered the solution on their mini-computer with a response time of less than five minutes. So the question was if we could help them. A week later we had a heuristic method that delivered acceptable cutting patterns within the five-minute limit. We first solved a one-dimensional cutting stock problem, in which we divided strips over the mother plates and then we divided the windows over the strips. Later the operators of the glass industries complained that our method was not always efficient. We learned the heuristics they used and incorporated them in the software. After a year there were no complaints anymore.

5. *Harbor pool*
   At that time there were about 30 big stevedoring companies active in the port of Rotterdam. Due to highly fluctuating workload in stevedoring companies, a separate labor pool had been formed with about 2000 dock workers divided among the respective stevedoring companies. Every day each company had the right to use its own share of the pool. If they needed fewer dock workers, the remaining number could be used by other stevedoring companies. In the event that dockworkers were unemployed, the stevedoring companies together had to pay for them, based on their share in the pool. At that time, there was a government ruling that compensated for idle dock workers. The cost of a dock worker on the payroll of the stevedoring company was lower than hiring one from the pool, but the pool was cheaper than hiring one from another source, specifically because of the compensation ruling. Each month AKB received the expected workload forecasts of all the stevedoring companies. This was highly confidential information. We combined these data with the order portfolio of the German industry. So we had better forecasts of the workload than the companies themselves! We used these forecasts to determine the optimal distribution of the pool shares for the next period. One way to solve this problem is by solving simultaneously the *newsboy problem* for each company. We also developed a *quadratic programming* method for this problem [1]. So AKB had what we now call a *data warehouse* of workload data. This service ran for at least 10 years.
6. *Portplan*
   Because of our knowledge of the harbor pool we became logistics experts in the port of Rotterdam. We developed a software package that was used by several stevedoring companies to forecast their own workload in much more detail. The workload system was based on their contracts, the arrival patterns of ships and of the types of commodity they had to handle. This system was one of the first *decision support systems.* Managers and controllers of the stevedoring companies could use it themselves. It was used for planning and budgeting over many years. The underlying models were queuing models and heuristics for optimization [2].
7. *Container terminals*
   Portplan was intended for general cargo. But more and more general cargo was packed in containers. So we also got the assignment to build a similar decision support system for container terminals. The goals were the same but the techniques totally different, since here we had to model the handling of containers at the terminal, which was done with sea cranes that took the

container out of the vessel and put it on a truck to move it to a stack area where a yard stacker or transtainer put the container in the stack. Later the client trucks would pick them up. This was the inbound flow, but the outbound flow was similar. Here we applied the brand new *mean value analysis* techniques for queuing networks where the trucks that moved between sea cranes and transtainers formed a closed loop network. This system was also used for many years [3].

8. *Meat production*
   For one of the leading supermarket chains we developed a *linear programming* model (packed in software) to plan the production in their two butcheries, each of which catered to about 180 supermarkets. Each butchery had its own orders for a week ahead and to enable the purchase of half cows and pigs or smaller parts of them to fulfill the demand. Although I never liked this type of production, I learned a lot about meat production. One of the constraints was that meat should not stay in the butchery for more than a day and that the demand for Saturday was more than could be produced on Friday. So on Thursday a part of the production for Saturday had to be done. In the beginning the head of the butchery was not keen on us. We had several meetings where we presented the data of simulated production of 4 weeks in the past to test our model. We came up with a much more efficient production plan than was being used. But according to the head butcher our ground beef was far too fat. So we incorporated a new constraint which was actually a recipe for ground beef. Later we were very disappointed because we lost a couple of tons of good-quality beef in our model. Ashamed, we confessed it to the head of the butchery. But then he said: “Well in that week we had a campaign for meat of lesser quality and in order to meet that demand we sold the better beef.” He continued: “It is amazing that your computer was able to discover this.” So now we had him on our side. Although savings could be made of 10,000 guilders a week in each of these butcheries, top management did not dare to let the production planning be done by a computer.

There are many more interesting cases I can recall. I only once used a Markov Decision Process, which was disappointing since I had hoped to apply my thesis work in practice. I used Markov chains a bit in manpower planning with the Formasy software package [4], originally developed in the group of Jaap Wessels. AKB obtained the rights, reengineered it and applied it in several large organizations.

The most important lessons learned in this period are:

- Always start with data analysis, using simple statistical methods.
- Never trust interviews as a single source of information: people are seldom able to tell precisely what they do, how they do it and to distinguish exceptions from main stream activities. Always try to verify their answers with objective data.
- Always ask your client why he needs the information you are supposed to produce. And more precisely, what he or she will do with the possible outcome. Often clients do not need what they ask for!
- Decision support systems should not dictate decisions but they should help the decision maker by *structuring* the decision process, giving *insight* in the consequences of a decision and *recommending* decisions.
- You seldom solve a practical problem with only one mathematical model. For different aspects you need different models and the consistent *integration* of these models into one decision support system is a typical model engineering challenge.

Instead of giving advice in the form of a report, we provided a decision support system that the client could use himself to create a new recommendation in response to changing circumstances. It seemed that the construction of such systems would become an interesting research field. This was one reason for my renewed interest in computer science. The other was based on a remark of a potential client who said to me: “You offer me a system to predict the daily workload for the coming weeks. That is beautiful, but I do not even know what the workload was yesterday.” He was right, of course. First we should build information systems that record the operational events of the business. So AKB went into software house activities and we built up expertise in database management systems: the *network model* was the new trend after the *hierarchical model* introduced by IBM™. We even delivered hardware: non-stop computers of GEAC™, which later became a major software producer.

The science of model engineering

In 1983 I received some offers for a professorship, in different disciplines. It seemed that there was interest in somebody with a solid scientific background who had experience in real business applications. So in 1984 I came back to Eindhoven as a professor in computer science, in particular the theory of information systems. The existing staff in my Information Systems group was working on *database theory* and I added a research activity in *intelligent systems* using my experience in decision support systems. At that time Eindhoven was famous for its style of programming, developed by Prof. Edsger Dijkstra and others. Programming was moving from an art to a science. The derivation of programs from *formal specifications* was the main goal. This led to beautiful methods of program construction. However for programming on a large scale these methods were less suitable. The business man in me said that I should not try to beat my colleagues in the field where they were among the world's elite. Since they always assumed there was a formal specification of the program, I asked the question: "Who provides these specifications?". Since there was no clear answer, this became my main topic for the next ten years. Of course, I did not restrict myself to specifications of small programs but I studied the specifications of complex software systems.

There are many different ways to specify a system. For me a *formal specification* of a system has three elements:

- A model of the environment in which the system to-be-built should operate.
- A model of the system itself.
- A set of additional properties the system has to fulfill.

Some of these properties concern the system itself, others the system in interaction with its environment. Preferably these properties are formulated in some *logic*, using the notions of the models. The model of the environment is usually less detailed than the model of the system to-be-built. Only the interaction with the system really matters. In *information systems engineering* one of the main functions to realize is the recording of events that happen in the environment. Then modeling can become confusing. Are we modeling the

environment or the part of the information system that records the events in the environment? Are we looking at the real world or at the virtual world?

Since the models are abstractions of reality, it might occur that some properties are not applicable to the models. However, the properties that are applicable should be *verified* by the system engineer. Some properties might be simple consequences of the model itself, others might be difficult to verify. A typical example of a difficult property is *deadlock freedom*, which means that a system will never 'hang', an ideal that is still to be realized by commercial software vendors. Today we call such an approach *model-based software engineering,* and it is a hot topic, but at the time I started this I was quite a pioneer in the field.

In computer science models are used in a different way than in operations research, where I came from. So I became interested in the theory behind modeling. There are many philosophical questions concerning systems and their models, but we will consider their practical use. Models can be used for different purposes:

1. *Descriptive* models: The model is used to document a system. This occurs in almost all the engineering disciplines.
2. *Explanatory* models: The model is used to explain some phenomena. This is what historians do: they link historical events by presumed causality relationships.
3. *Predictive* models: The model is used to forecast some phenomena under certain assumptions. The weather forecast is the most well-known one.
4. *Optimization* models: The models are used to tune the parameters of a system in order to improve its performance. Typical examples are the scheduling of trains, traffic lights or the mixture of food components to produce food.
5. *Control* models: The model is used to influence the behavior of a system. For instance, to move a robot arm to perform some task or to regulate the influx of fuel in a combustion engine.
6. *Construction* models: To design, synthesize and test a system. Construction models can be considered blueprints.

In computer science and particularly in software engineering, models are used to *construct* a system. In that sense model engineering is part of *systems engineering*. However, the scope of model engineering is wider, since we also make models for other purposes than constructing a system. Up to the time I became a professor I never used models to construct a system, except for designs

of a rabbit hutch which I also realized. Software systems differ from other complex artifacts in the sense that even a small software system may have an infinite set of inputs, each with a specific output. This makes it impossible to verify the functionality by exhaustive testing. Complex software systems have many components operating in parallel, which generates exponential growth in possible behavior. The main engineering challenge in high-tech systems, like cars, robots and airplanes is the software system embedded in it. The production of copies of airplanes, for instance, is very costly but if we have constructed a software system, it is easy to make a copy of it. So for software systems the design phase is the most crucial one and the verification of its behavior at the model level is essential.

Mechanical engineers, civil engineers and architects use *geometry* to make models of the system they want to construct. In fact, geometry is their *modeling language*. Geometry is a fine language to express artifacts that have a physical form. However, software does not have such a form. Software defines the behavior of a system and so for *software engineering* new languages had to be invented. A *modeling framework* consists of languages to express *models* and to express *properties* of a model, and it often has a set of *techniques* and software tools to *analyze* models. Modeling languages often have a graphical notation and they should have *formal semantics*. The model engineer can chose from a large variety of modeling frameworks. Not all of them satisfy these requirements. There are several ways to classify these frameworks. We distinguish three classes of modeling framework:

- *Equation models*: Equations and inequalities over variables in a Euclidean space (R^n), where the variables are vectors or functions and where, in the case of more solutions, there is an optimization criterion to select one solution.
- *Data models*: To represent relationships between *entities* or *concepts*. The term ‘data’ may also be replaced by ‘information’, ‘knowledge’ or even ‘belief’.
- *Process models:* To express the dynamic behavior of a system in the form of *events* or state changes.

In all modeling frameworks we can discover the notion of a *state*. The set of all possible or allowable states is called the *state space*.

In equation models the state space is often a Euclidean space. In these models we can distinguish different types of variables: *exogenous* variables that are determined by the context of the system, *control* variables that may be determined by the designer at design time or the decision maker at run time and *endogenous* variables that are the results of solving the equations.
In data models the state is a concrete *instance* of the data model and in process models the state space is mostly an arbitrary finite or countable set. In addition to having a state space, the process models and some of the equation models also have a *transition mechanism* that relates each state to zero or more *futures*.

Based on these notions we can classify modeling frameworks along the following five *characteristics*:

- *Static* or *dynamic*
  If there is the notion of a transition mechanism, it is called a dynamic model. Otherwise, it is a static model.
- *Discrete or continuous state space*
  If the state space is part of some Euclidean space, it is called continuous. If it is an arbitrary finite of a countable set, it is called discrete.
- *Discrete or continuous time*
  If the futures, associated to a state, are functions from intervals of R^1 into the state space, we have continuous time. If each future of a state contains exactly one state, we have discrete time. In the latter case, the transition mechanism is a binary relation over the state space.
- *Deterministic, non-deterministic or stochastic transition mechanism*
  If the transition mechanism determines for each state exactly one future, then we have a deterministic model. If a state may have more than one future, it is non-deterministic. Stochastic models are like non-deterministic models but each future (or set of futures) has a given probability.
- *Stated-based or event-based*
  A dynamic model with discrete time can be specified by its state space and a transition relation. This is called state-based specification. Another approach is to have a set of events and a mechanism to relate successive events. This is called event-based specification.

Without the pretention to be complete, we present 18 modeling frameworks in table 1:

		***state*:**	***time*:**	**stochastic**	
	static	**discrete**	**discrete**	**determ.**	**state-based**
	dynamic	**continuous**	**continuous**	**non-determ.**	**event-based**
Equation models					
(non)linear programming	s	c	-	-	-
difference equations	d,s	d, c	d	d	-
differential equations	d,s	c	c	d	-
control models	d	d,c	d,c	d,s	-
(non)linear regression	s	c	-	-	-
Data models					
relational model	s	d	-	-	-
ER model	s	d	-	-	-
functional model	s	d	-	-	-
XML	s	d	-	-	-
RDF	s	d	-	-	-
Bayesian networks	s	d,c	-	-	-
Process models					
automata	d	d	d	d,n	s
Petri nets	d	d	d	d,n	s,e
process algebra's	d	d	d	d,n	e
Markov processes	d	d	d,c	s	s
renewal processes	d	d	d	s	e
time series	d	d	d,c	s	e
queuing networks	d	d	d	s	s

Table 1 Basic modeling frameworks

I have not mentioned the languages, like DFD and IDEF, for hierarchical decomposition of processes. Although they had no formal semantics for behavior they were used a lot by software engineers in practice. Neither have I mentioned simulation languages because they usually have no formal semantics and therefore no methods for analysis. Besides these formalisms there are logical languages like *Prolog*, *Linear Time Logic* (LTL) and *Computation Tree Logic* (CTL) to express properties of processes. The *System Dynamics* framework that became popular by the Club of Rome in 1972, is in fact a difference equations framework.

Several of the listed frameworks are competing in the sense that one can model the same system in different frameworks. To be able to use most of these formalisms at a professional level, one needs a serious course and a lot of practice. That is one of the reasons that engineers do not like to switch from framework. There are schools or 'tribes' of users of a specific framework and there are even 'tribal wars' between these communities.
Although most equation models and most data models do not have the notion of a transition mechanism, this does not mean that it is impossible to express dynamic behavior in these frameworks, but then it is the responsibility of the model engineer.

In general, the *goal* of modeling determines the modeling framework we need. So for the evaluation of the *performance* of a system Markov processes or queuing networks are suitable. To verify if a system behaves *conform* a set of rules, a process algebra might be preferable. Since a systems engineer has to deal with many of these goals, he often needs to use more than one framework.

For software systems we use *dynamic* models with *discrete time* and *discrete state* space. To understand this note that the state of a computer system can be represented by a (very long but finite) sequence of zeros and ones and that computers may change their state after each pulse of their internal clock or after a new input. For construction purposes we do not need stochastics although when performance analysis is required we do have to assume potential usage which is modeled with a stochastic process. We use non-determinism to model the influence from the environment, for instance, the behavior of the users.

Remember we were looking for a framework to model information systems. In the eighties the development of information systems was *data-centric* and not *process-centric,* as it is today. The database was the heart of the information system and as soon as the data model was fixed, programmers could start programming different applications that shared the database, which should be created according to the data model. A data model is perfect for defining a state space but bad for defining state transitions. So I decided to choose a process framework. After an excursion with a self-defined framework based on a network of automata (see [5]) I moved to Petri nets. Petri nets had been well known for a couple of decades at that time. It is the first model of concurrency. Petri nets have an elegant graphical notation which allowed defining infinite state spaces with a finite graph and there are interesting and useful analysis methods for Petri nets. I will explain very briefly what Petri nets are.

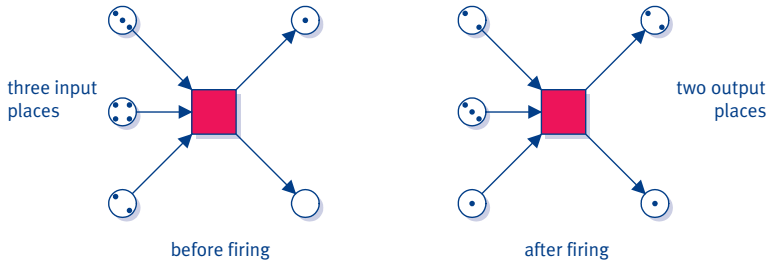


Figure 1 Firing a transition

A classical Petri net has three types of building blocks:

- *Transitions* model (instantaneous) *events* or time consuming *actions*. In diagrams: squares.
- *Places* model *stages*, to express that some part of a system is in a certain stage waiting for a new event or *buffers* to store objects. In diagrams: circles.
- *Tokens* can be used as stage markers or to model objects. Objects can be of any kind, such as data elements like a message, physical objects like a container, or abstract objects like an agreement. In diagrams: dots in places.

A state of a Petri net is a *distribution* of tokens over the places. The transition mechanism of a Petri net is defined by a very simple but powerful rule, called the *firing rule*. In fig. 1 this is illustrated: a transition may change the *global* state of the system *locally* by *consuming* one token from each of its input places and at the same time *producing* one token for each of its output places. This is only allowed if there are enough tokens in the input places.

Fig.2 shows a Petri net that models a complex container transportation system. The tokens are not displayed in the left-hand subsystem but the places could contain tokens that represent ships. In the right hand subsystem the tokens represent trucks and in the middle they represent containers or containers on a truck. The transitions model actions such as unloading a truck or events like arriving in a harbor. The places model stages of the containers, trucks and ships, such as waiting at a terminal and stripping or stuffing a container at a client site.

However, classical Petri nets are too simple to model all the details we would like to model. For instance, the names of ships and the identities of the containers are not expressed in the Petri net of fig.2. In classical Petri nets, as in many other pure process frameworks, state spaces and the transition relation can only be specified by *enumeration*, which is impractical for large data-intensive systems. So we

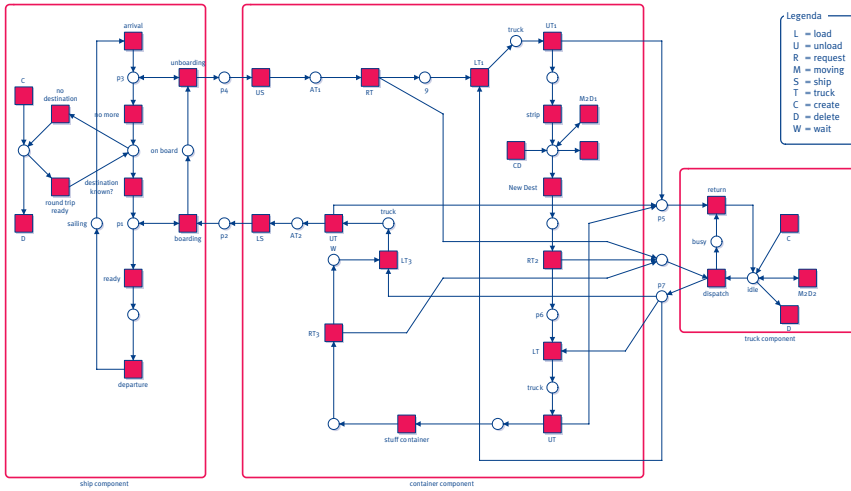


Figure 2 Petri net of a container transportation system

decided to *enrich* the classical Petri nets with a language to define state spaces and transition relations implicitly, by means of formula.

The language was *a typed functional language* that we designed in the Information Systems group. In this language one can define arbitrary complex *data types* and arbitrary complex *functions* to transform elements of these data types. We added a data type to each place, meaning that all tokens in that place should have a *value* belonging to that type. We also associated each transition with a *function* for each output place that could transform the values of the input tokens to values of the output tokens. The domain of the function could be smaller than the Cartesian product of the input data types, which means that we could have a *precondition* for firing. It is also possible to suppress some output tokens. In fig.3 we see a simple example in which there is a precondition so that only even numbers are added.

We also enriched the framework with the notion of *time*. Each token was associated with a *timestamp*. The firing rule was restricted in the sense that a transition could only fire if (1) all tokens had timestamps not greater than the *current* time and that (2) if one or more transitions can fire at the current time, one of them will do so. We called the last property *eagerness*. A consequence of this mechanism is that for each state of the model we can determine the time, the system will leave the state to move to the next state. This time mechanism turned

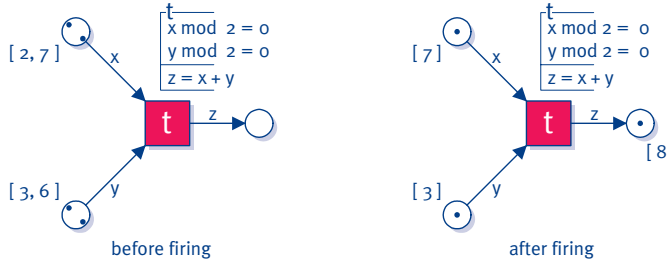


Figure 3 Firing in extended Petri nets

out to be very useful to model real-time systems. Since it is easy to model *random number generators* in the framework it is possible to model stochastic behavior. To gain an overview of large systems we enriched the framework with the notion of hierarchy to cluster subsystems into a kind of 'super transition'.

Now we had a framework in which it was rather easy to model all details we needed in a rather easy way. We called our framework of enriched Petri nets *ExSpect* [6] and we made a software tool, to *edit, animate* and *simulate* arbitrary complex systems in this framework. For those days we had a very advanced graphic user interface. In our Information Systems group Lou Somers and Marc Voorhoeve were the main software engineers of the ExSpect tool. Soon after the start of the tool development, Wil van der Aalst joined us as a PhD student after he had finished his Master's in our group. He contributed substantially to the development of ExSpect. Later Eric Verbeek, also a former Master student, joined us and he became the main software architect. The first version of ExSpect became available in 1989 and TNO was our 'launching customer'.

In the late eighties we were not very familiar with the Petri net community and we learned Prof. Kurt Jensen had just developed so-called *colored Petri nets* [7] at that time, which was almost the same as what we had done! Kurt also produced a software tool, now called CPN-tools. I believe that we released our tools in the same month in 1989. Both frameworks were able to model discrete as well as continuous time, they could model deterministic, non-deterministic as well as stochastic systems and the properties of classical Petri nets enabled one to use it in a state-based style as well as an event-based style.

In the meantime many other university groups have attempted to make integrated modeling frameworks. In the last decade the software industry has made progress by making an integrated modeling framework and supporting tools. Today the

Unified Modeling Language (UML) is the standard modeling framework for the industry. It is a family of various languages. Some of them have unclear semantics and the integration of the various views has to be improved. However, it is a good start. The most popular language of this family is the Class Diagram, which is an extended data model. One of the languages for process modeling is the Activity Diagram, which is very close to Petri nets. They became available in the nineties as a result of industrial standardization. In table 2 we list six integrated frameworks, counting UML as one family. Again, this is not an exhaustive list.

		state:	**time:**	**stochastic**	
	static	**discrete**	**discrete**	**determ.**	**state-based**
	dynamic	**continuous**	**continuous**	**non-determ.**	**event-based**
VDM	s	d	-		-
Z	s	d	-		-
CPN-tools	d	d	d,c	d,n,s	s,e
ExSpect	d	d	d,c	d,n,s	s,e
UML class model	s	-	-	-	-
UML state charts	d	d	d	d,n	s,e
UML activity diagrams	d	d	d	d,n	e
mCRL2	d	d	d,c	d,n	e

Table 2 Integrated modeling frameworks

The industry was very interested in ExSpect and we made models of a wide variety of systems, such as logistics systems, the control of a satellite and of the Dutch railway system. We stopped the development of ExSpect in 1996 because the scientific challenge was over, but there are still some users today. It is a pity that we did not develop a simple method with supporting software tools to transform our system models into real software code. This is the main reason that ExSpect and also CPN-tools are not so much used for software development. I regret that because I strongly believe it is a very good approach. But for software engineers it was often a big hurdle to start making a detailed model and then transform that by hand into a software system. This omission can still be repaired by making additional tools [9].

Anyway, my goal was reached: I had enough evidence that with this (ExSpect) framework I could model all discrete event systems efficiently in a systematic way. During my sabbatical year 1991-1992 in Waterloo, Ontario, I wrote a book [8] about this framework. It did not become a best seller although a year ago it was reprinted. For me it was time for a new challenge!

Return to practice

I always kept relationships with the field of industry after my AKB period, also using them to create Master projects for students (around 100 in the period 1984-1994). One special relationship had already started in my AKB period, with Bakkenist Management Consultants, at that time one of the leading Dutch consultancy firms with a strong focus on Information and Communication Technology (ICT) and relationships between organization and information. In 1994 I moved to Bakkenist to become one of the managing partners with focus area ICT-consultancy. There I was involved in all kinds of very exciting consultancy projects with large companies. In particular in *ICT strategy* projects, in *business process reengineering* (BPR) and in *e-business*. Finally it became clear to business managers that the principle: '*organize before automate*' was not the best way to apply ICT. The reason is that they organized the work for people and then used computers to automate the human tasks. It is much better to redesign business processes fundamentally using all the possibilities of ICT. The BPR wave started with the redesigning of internal business processes specifically in large bureaucratic organizations. Later, the link with the environment of these organizations was made: the customers and the suppliers. This caused the second wave of e-business. Actually the e-business movement was the logical next step in this revolution: a business process triggered by the customers who should be in control as much as possible.

Bakkenist made me aware of the BPR wave and I saw that the precise modeling of business processes would become very important. I realized that our ExSpect framework was extremely well-suited for this task. It took me a little while to convince my colleagues at TU/e that we should apply our approach to this field, but we did! We introduced the notion of a *workflow net* which is a special class of Petri nets that models precisely what we call a business process. It has one initial place and one final place and every other place or transition is on a path from the initial to the final place. The initial state is the distribution with exactly one token in the initial place and the final state with exactly one token in the final place. Specifically Wil van der Aalst made a great contribution by introducing the notion of *soundness* or *weak termination* for workflow nets: a workflow net is *sound* if from any state that is reachable from the initial state, it is possible to reach the

final state. This notion generalized the deadlock property and it turned out to be a very good *sanity check* for modeling business processes. So we applied ExSpect at Bakkenist in various workflow projects. Wil and I wrote a book on workflow systems [10] which later became a best seller that appeared in five languages.

The BPR and e-business waves saw a boom in the consultancy business. However, it was generally believed that, in order to stay ahead of the competition, consultancy firms should globalize. It was my task to look for a global player to merge with. In 1999 we merged into Deloitte, at that time Deloitte & Touche. The first two and a half years after the merger I was one of the managing partners of the consultancy group. When the initial symptoms of the merger had faded away I became professional director of consultancy, which meant that I became responsible for knowledge management, training, quality control and innovation of the consultancy practice of some thousand consultants. It was a wonderful period and a great experience, but when the consultancy directorate was established and I had spent another ten years in consultancy, it was again time for a move!

Science of model engineering continued

At that time TU/e asked me to come back as a full professor. I tried to discourage the dean from hiring such an old person, but they still offered me the job. Paul de Bra became professor shortly after I left in 1994. He focused on the database part of the Information Systems group. We renamed my chair Architecture of Information Systems (AIS) and together we formed the Information Systems section. Shortly after my return in 2004, I was asked to become dean of the department. I held this job from 2005 till 2009. Although I first considered it as my 'tour of duty', I liked the job very much. During that period I was very lucky that Wil van der Aalst became my successor as head of the AIS group. I had to renew my line of research. In my former period at TU/e I was focused on methods and tools for modeling, but they were available by now. I still wanted to work on methods that could support the work of model engineers, specifically on methods and supporting software tools to help engineers in finding *errors* in their models. All the work in this period I did in strong cooperation with members of the AIS group, in particular with Natalia Sidorova, Marc Voorhoeve, Jan Martijn van der Werf and Wil van der Aalst. We focused on four topics that I consider also as the main challenges for the future.

- Model *integration*. Often a single model of a system is inadequate or too complex. Therefore, we have to make several models and the challenge is then to make them *consistent*, i.e. to ensure that the different models do not contradict each other. There are two kinds of integration: (1) integration of different *views*, which means models that describe different *aspects* of a system often in different frameworks and (2) different *components* describing different *subsystems* of a large system, mostly within the same framework. I had encountered these problems already in my AKB time when I had to solve complex optimization problems I used a simplified *optimization* model that was a great abstraction of the real system to find *control parameters* and a *simulation* model for calculating the *effects* of the control parameters. However, now we were studying different forms of integration: (1) the integration of data models and process models [10], (2) the integration of different workflow models that represented different use cases of the same system [12] and (3) the integration of communicating workflows [13].

- Model *verification*. This concerns the verification that a model satisfies the properties defined in a specification. This is a popular topic. The standard techniques are: (1) *model checking* which means inspection of the whole state space to check whether the property holds, or (2) to use *theorem proving* techniques to check a human generated proof. I concentrated on another approach: *correctness-by-construction* and specifically for one property: weak termination as described above. In particular, we worked on frameworks that extended the classical Petri nets but not with the full power of colored Petri nets, because it is very hard to prove properties of these systems. Our challenge was to extend the classical Petri nets in such a way that the weak termination property was easy to verify by inspecting the model only and not its complete state space. Extensions we considered are: (1) *nested* Petri nets, in which the tokens themselves are Petri nets [14] and (2) *history-based* Petri nets, where transitions have a precondition based on the history of the process [15].
  We also worked on the verification of more complex forms of weak termination: (1) weak termination of workflow nets in the event that there is an infinite sequence of input tokens, which we called *generalized soundness* [16] and (2) weak termination of workflow nets with *resource constraints* [17].
- Model *validation*. This is the fundamental question of model engineering: 'Is my model a good representation of the real system, i.e. are properties that hold true for the model also valid for the system?' I considered this question from a normative point of view, namely whether the real system is behaving in line with the model: 'Is the behavior of the system allowed in the model?' In the case of business processes this is typically a question for auditors [18].
- Model *identification*. This concerns discovering a good model of a system based on observations of the real system. There are many questions and approaches in this area. Actually in my PhD thesis I had already addressed this problem: using the data of process execution to improve the estimators. In my AKB time we had the problem that no operational data were available in digital form. But today almost all organizations have information systems that record the relevant events in a so-called log file. The activity of reconstructing a process model out of log data is called process mining. Wil van der Aalst has put this topic on the international research agenda and he made several essential contributions [19].

In 2007 I became director of the Stan Ackermans Institute [20], which organizes 16 post-Master programs in design and engineering, concluding with an innovative design project in industry. Here model engineering is a cornerstone of systems engineering!

In 2009, together with Henk Zeegers of Inroads and Michiel van Osch, I started the TSR-project [21]: the development of a so-called *service robot* that can execute basic human activities, in a daily life environment. This project involves 10 organizations working closely together. The *service robot* is tele-operated, which means that it can be controlled completely by a human operator from a *cockpit* at an arbitrary distance. The robot can perform many simple tasks autonomously, but not for longer than a minute or so. This project offers beautiful modeling challenges and the transformation of models into a physical system. I find it much more exciting to build systems that act in the real world than systems that produce information in a virtual world. One of the modeling challenges in robotics is the integration of continuous and discrete behavior. The continuous behavior is described by differential equations and the discrete behavior, for instance, by enriched Petri nets, like colored Petri nets. Tokens that seem to be in rest in a place, waiting for the firing of a transition, are in fact changing their local state in a Euclidean space according to a differential equation. Such models are called *hybrid* models.

Outlook

In this last section I would like to share my expectations about model engineering in the future. It has a positive and a negative side. Let me start with the negative side.

Model engineering is used in almost all scientific disciplines. This is due to the fact that a big part of the model engineering knowledge is packed in software and available as software tools, such as Matlab™ and SPSS™. You can use these tools without deep knowledge of the underlying theories. On the one hand, this is positive but, on the other hand, it is also dangerous. Many scientists use these tools to make models and base conclusions on them. However, they are often not aware of the fundamental theories behind the tools and of their limitations.
For example in the social sciences, economy and earth sciences we see all kinds of misuse of mathematical models. In the social sciences the majority of research is *fact discovery* based on questionnaires. The design of the questionnaire is one of the two creative moments in such research. Everyone who has ever filled out a questionnaire knows how unreliable his answers are. Next statistical machinery, for instance SPSS, is used to discover or test correlations. Then the second creative moment occurs: the researcher selects some of the *correlations* and interprets them as *causalities*.

The same happens in economical research. Economical models are often based on assumptions that do not seem to be true, such as the axioms that the 'markets are efficient' and that 'people behave in a rational way'.
Also in the climate discussion we see that scientific statements are based on mathematical models. The climate is the system that produces the daily weather. The climate is a very complex system and mathematical chaos theory provides good reasons to assume that the climate is not predictable. Nevertheless, many scientists believe that the climate is changing and they even have found the causes. All we know is that the weather is changing all the time. But concluding that the system behind the weather is changing is very difficult to establish based on the weather data only! I am convinced that if these scientists published all the assumptions on which their conclusions are based, their results would not be taken so seriously. In 1959 Darrel Huff wrote the famous book 'How to lie with

statistics' [22]. Now it is time for a new book: 'How to lie with models'. I would like to see a 'driver's license' for scientists who use model engineering to do their research. Such a license should include an ethical code that makes publish the modeling assumptions compulsory.

It is very positive that model engineering is applied so much in systems engineering. Almost all technological design will be done using model engineering.
In the design of *mechanical systems* Computer Aided Design (CAD) systems are already very sophisticated and during the design process all kinds of checks and performance indicators are calculated automatically. The step from design to real product is also being automated. Computer Aided Manufacturing (CAD) systems are coupled directly to the CAD systems. The more 3D printing develops, the sooner the step from design to product will disappear completely.

In the *electronic systems* industry we have already seen the same development for a long time: an application specific circuit (ASIC) is designed and production is completely automated from a model.
For *software systems* there will be excellent modeling tools in the future, that allow for all kinds of verification during the design process. These tools will also have code generators that will generate around 95% of the system code from the model. There will always remain some complicated parts that have to be programmed. Some of this code will be realized as ASIC, others stored in memory. In fact, programming will be replaced by modeling, or modeling will become programming at a higher level of abstraction.

The latest branch of systems engineering is *bio-engineering* where DNA and, in the future, also complete cells and even micro-organisms will be engineered. Here is a great challenge for model engineering. Successes have already been achieved in the DNA-design, but we probably need new modeling frameworks in the future.

Today many complex systems have mechanical, electrical and software subsystems; in the future these will probably also be biological ones. Integration of these subsystems at production level seems to be a natural step. This is only possible if we have an integrated modeling framework. In all these cases the production steps are highly automated. An important side effect of this is that we do not have to ship production to low-labor-cost countries in the future, we can do it here!

Acknowledgements

It is evident that I have learned a lot from my teachers, my colleagues at TU/e, AKB, Bakkenist and Deloitte, and my PhD students. I could not have made this beautiful journey between virtuality and reality without the help of the supporting staff, in particular the great secretaries that I worked with. I am very grateful to all of them! I am also very grateful to TU/e for offering me a very stimulating working environment. I appreciate the focus on science and engineering, the high quality standards and the friendly atmosphere.

References

For concepts, methods and techniques not explained here, see Wikipedia.

0. M. Hilbert and P. Lopez. The world's technological capacity to store, communicate, and compute information. Science vol. 332 (2011)
1. K. van Hee. Sharing profits and liabilities of a pool of dockers. European Journal of Operational Research vol. 15 no.3 (1984)
2. K. van Hee, D. Leegwater and B. Huitink. Portplan, decision support system for port-terminals. European Journal of Operational Research 34, issue 3 (1988)
3. K. van Hee and R. Wijbrands. Decision support system for container terminal planning. European Journal of Operational Research 34, issue 4 (1988)
4. J. Wessels and J. van Nunen. Formasy: forecasting and recruitment in manpower systems. Statistica Neerlandica 30 (1976)
5. K. van Hee, G.J. Houben and J. Dietz. Modeling of discrete dynamic systems: framework and examples. Information Systems vol. 14 no. 4 (1989)
6. K. van Hee, M. Voorhoeve and L. Somers. Executable specifications for distributed information systems. Information System Concepts: an in-depth analysis, Elsevier Science Publishers (1989)
7. K. Jensen and L. Kristensen. Coloured Petri nets. Springer (2009)
8. K. van Hee. Information systems engineering; a formal approach. Cambridge University Press (1994 and 2009)
9. D. Harel. Can programming be liberated, period? IEEE Computer (2008)
10. W. van der Aalst and K. van Hee. Workflow management: models, methods and systems. MIT Press (2004)
11. K. van Hee, J. Hidders, G.J. Houben, J. Paredaens and P. Thiran. On the relationship between workflow models and document types. Information Systems vol. 34 nr 1 (2009)

12. K. van Hee, N. Sidorova, L. Somers and M. Voorhoeve. Consistency in model integration.
Data & Knowledge Engineering 56 (2006)
13. K. van Hee, A. Mooij, N. Sidorova and J.M. van der Werf.
Soundness-Preserving Refinements of Service Compositions.
Lecture Notes in Computer Science vol. 6551 (2011)
14. K. van Hee, I. Lomazova, O. Oanea, A. Serebrenik, N. Sidorova and M. Voorhoeve;
Nested nets for adaptive systems. Lecture Notes in Computer Science vol. 4024 (2006)
15. K. van Hee, A. Serebrenik, N. Sidorova, and W. van der Aalst.
Working with the past: integrating history in Petri nets.
Fundamenta Informaticae, 88(3) (2008)
16. K. van Hee, N. Sidorova and M. Voorhoeve.
Generalized soundness of workflow nets is decidable.
Lecture Notes in Computer Science vol. 3099 (2004)
17. K. van Hee, A. Serebrenik, N. Sidorova , M. Voorhoeve and J. van der Wal;
Scheduling-free resource management.
Data and Knowledge Engineering vol. 61 nr 1 (2006)
18. W. van der Aalst, K. van Hee, J.M. van der Werf, M. Verdonk and A. Kumar;
Conceptual model for on line auditing. Decision Support Systems (2010)
19. W. van der Aalst.
Process mining; discovery, conformance and enhancement of business processes.
Springer (2011)
20. Stan Ackermans Institute, www.3tu.nl/en/education/sai
21. Robot Rose, www.robot-rose.nl
22. D. Huff. How to lie with statistics. Norton, NY (1954)

Curriculum vitae

Prof. Kees van Hee has been full professor of Computer Science at Eindhoven University of Technology since 1 July 1984, although between 1994 and 2004 he was part-time professor. His tenure ends on 1 December 2011 and he will be giving his valedictory lecture on Friday 11-11-11.

Kees van Hee studied Mathematics, with Physics and Economics in Leiden. In 1971 he became assistant professor of Computer Science in Leiden. At the end of 1973 he moved to TU/e where he received a PhD in 1978. The subject of his thesis was Markov decision processes with incomplete information. After his PhD he became director of AKB, a consultancy firm in Rotterdam specializing in operations research and statistics. In 1984 he was appointed professor of Computer Science, in particular the theory of information systems. He focused on methods for modeling information systems and on intelligent information systems. In 1994 he became director of Bakkenist Management Consultants. When this firm merged with Deloitte in 1999, he became a partner of Deloitte Consultancy. In 2004 he returned full-time to TU/e and focused on analysis methods for information systems using Petri nets. From 2005 till 2009 he was Dean of the Mathematics and Computer Science department. Since 2007 he has been director of the Stan Ackermans Institute, which provides professional doctorate programs in design and engineering.
He published over 130 papers. He was first supervisor of 15 and second supervisor of 10 PhD students. Seven of them are professors today. He supervised ca 130 master thesis projects.

Colophon

Production
Communicatie Expertise Centrum TU/e

Cover photography
Rob Stork, Eindhoven

Design
Grefo Prepress, Sint-Oedenrode

Print
Drukkerij Snep, Eindhoven

ISBN 978-90-386-3000-7
NUR 980

Digital version:
www.tue.nl/bib/

Visiting address
Den Dolech 2
5612 AZ Eindhoven
The Netherlands

Postal address
P.O.Box 513
5600 MB Eindhoven
The Netherlands

Tel. +31 40 247 91 11
www.tue.nl

Technische Universiteit
Eindhoven
University of Technology