INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN, ICED'07
28 - 31 AUGUST 2007, CITE DES SCIENCES ET DE L'INDUSTRIE, PARIS, FRANCE

MODELLING THE WORLD IN STATES

Tapio Korpela[1] and Jussi A. Karjalainen[1]
[1]University of Oulu, Finland

ABSTRACT
The aim of our research work is to find out how the total concept of the needed technical process can be captured into the formal model, which can be used for the customer oriented requirements defining and which leads to initial functional description of the phenomena into which the new product is aimed to be designed. Our research work has been inspired by Vladimir Hubka´s description of the technical process in his concept of the Transformation system, which is published in the book: "Theory of Technical Systems", written by Vladimir Hubka and Ernst Eder in 1988 [1]. The theoretical foundation for this kind of modelling the world in states is in the discrete mathematics, which concerns the modelling of the finite state machines for computing.

Keywords: process modelling, requirements modelling, theory of technical systems

1 INTRODUCTION
Technical products are needed to help us to transform an initial unsatisfactory state into a final satisfactory state. Products are used by human or by other systems in a certain environment. A designer of the product should be aware of these matters entirely. He or she should understand for whom the product is designed and how it is used and how it is presumed to function in a certain environment. The designer ought to learn these things in the early phases of the design process. In our paper we are going to introduce ideas how a real world phenomenon can be captured into the state transition model, which is as realistic as possible and which can be used as a foundation of design process of a technical product. This is a top down approach, the purpose of which is to model the chain of states and events capturing inside all the functionality which is needed to transform an initial unsatisfactory state into a final satisfactory state. The theoretical foundation for this kind of modelling the world with states is in the discrete mathematics, which concerns the modelling of the finite-state machines for computing [3]. From the theory of finite-state machines a tool (statecharts) for modelling the behaviour of the computer systems in the third generation object oriented modelling language UML (Unified Modeling Language) has been developed [5]. The research work of David Harel from the Department of Applied Mathematics, the Weizmann Institute of Science, Rehovot, Israel is a foundation of modelling the behaviour of the systems by using UML state models [6].

2 TECHNICAL PROCESS
Every time when a tangible or intangible product is used for some purpose, a technical process proceeds from an initial state through some states to a final state. Vladimir Hubka and W. Ernst Eder describes in their book: "Theory of Technical System" (TTS) [1] the technical process (Tp, in Fig. 1) as a state transformation process in their concept of the Transformation system, which is presented in Figure 1. In this theory the needed technical process is a state transition process from an initial unsatisfactory state through several intermediate states to a final satisfactory state. State transitions are produced by effects, which are produced by the users (HU, in Figure 1.), or by the technical system (TS, in Fig. 1), which is to be designed, or by the active environment (AEnv, in Fig. 1). The purpose of this model is to capture the real world phenomenon so that it could be used for the basement of the design process of a new product. The technical process in this system describes the total state transition from an initial state to a final state. State transitions are caused by humans or other systems, which need this process for their purposes. State transitions take place in a certain environment, which causes also effects on this process. In the concept the technical system is developed to help the users of the system to produce more powerful aids and effects to get over in the technical process.

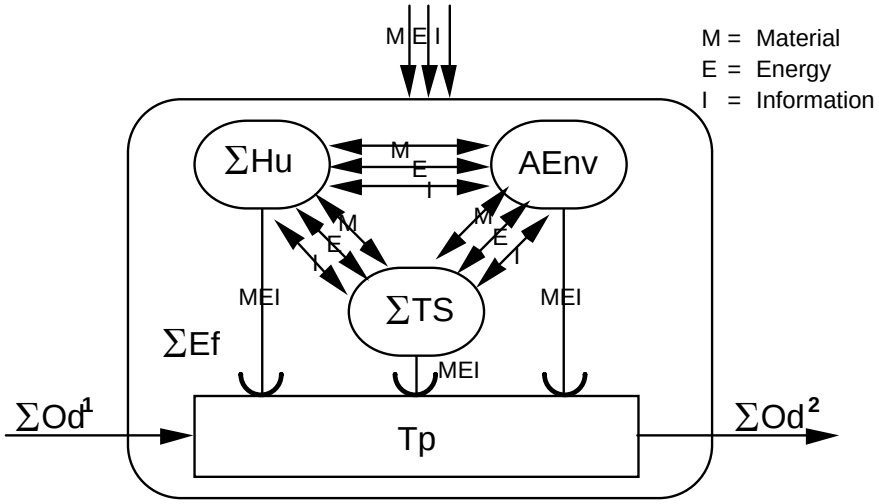


Figure 1. The transformation system by Hubka and Eder [1].

3 LIFECYCLES OF THINGS

When a product is used for some purpose, a sequence of events takes place. It starts, it proceeds through some stages, until it ends. The process has a lifetime in the world. Sally Shlaer and Steven J. Mellor describes the lifecycle of a thing by the behaviour pattern of an airplane in their book: "Object lifecycles: modeling the world in states" [2].

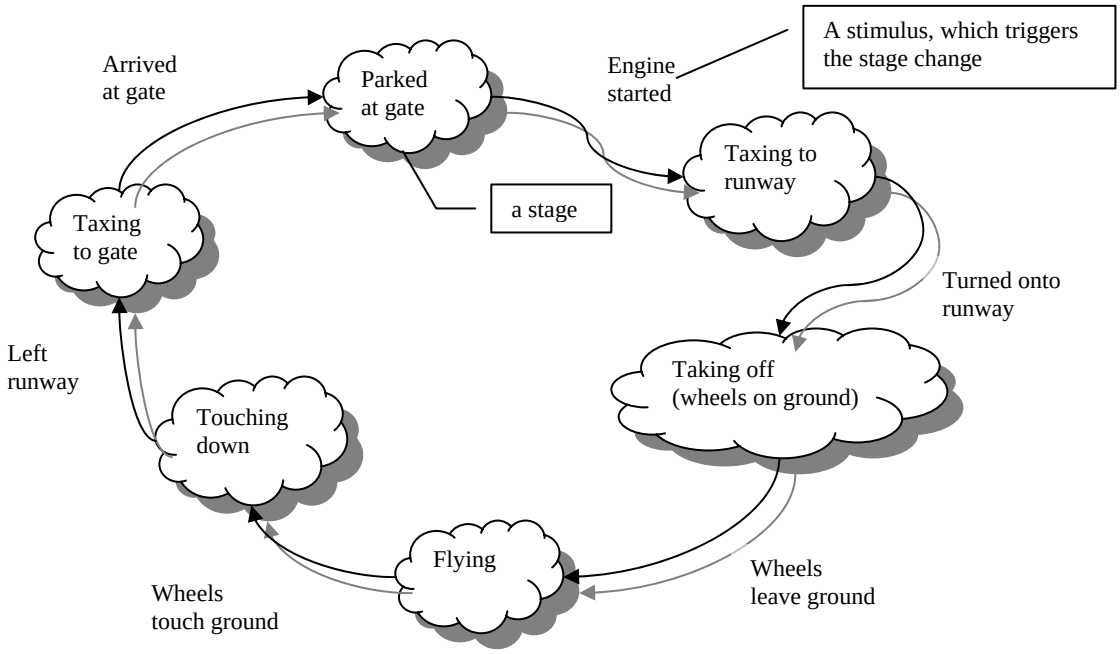


Figure 2. Behaviour pattern of an airplane [2].

A real-world thing is in exactly one stage of its behaviour pattern at any given time. An airplane cannot taxing to runway and flying at the same time. When a thing gets a stimulus, it changes the stage. A stimulus is caused by a user of a thing or by an other system. It can be caused by the thing itself or it can be caused by an active environment in which the thing is used. If an airplane looses a wheel, when it is touching down, the behaviour pattern may be changed dramatically. A technical process, introduced in the previous chapter, is able to be modeled by this kind of behaviour pattern.
Yet this model is too simple for modeling the real behaviour of systems and technical processes. The idea has been developed further as a modeling tool for the object-oriented analysis (OOA) methods used in computer science. The foundation for this developing process is the finite-state machines, the theory of which belongs into the discrete mathematics.

3.1 Finite-state machines with output

A finite-state machine, as it is presented in the book: “Discrete mathematics and its applications”, written by K.,H., Rosen [3], includes a finite set of states, with a designated starting state, an input alphabet, and a transition function that assigns a next state to every state and input pair. There exists also an output function. If an output function corresponds to a transition between states, machines of this type are known as *Mealy machines*, according to G., H., Mealy, who was first studied them in 1955. If the output is determined only by the state, the finite-state machine is known as a *Moore machine*, since it was introduced by E., F., Moore in 1956 in the article: “Gedanken-Experiments on Sequential Machines, which was published in “Automata Studies” [4].

There are two basic styles to illustrate finite-state machines. A vending machine for coffee and tea automate is illustrated by a directed graph in Figure 3. The vending machine accepts 10 cent, 20 cent, and 50 cent coins, which are inputs of the state machine. A cup of coffee or tea costs 30 cents. States are the amounts of deposited coins (s0=0 cents, s1=10 cents, s2=20 cents, and s3=30 cents). When a total of 30 cents has been deposited, the machine returns the amount in excess of 30 cents, and a cup of coffee or tea can be selected by pushing the C (coffee) or T (tea) button. These selection buttons are also the inputs of the machine. The finite-state machine of the vending machine is a Mealy form, because the outputs (returned money, a cup of coffee or tea) correspond to transitions between states.

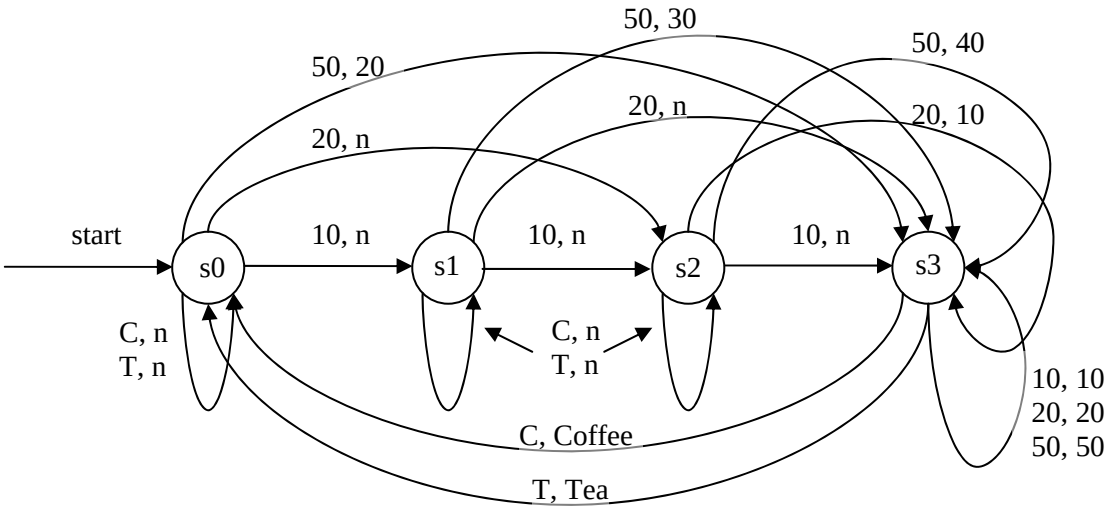


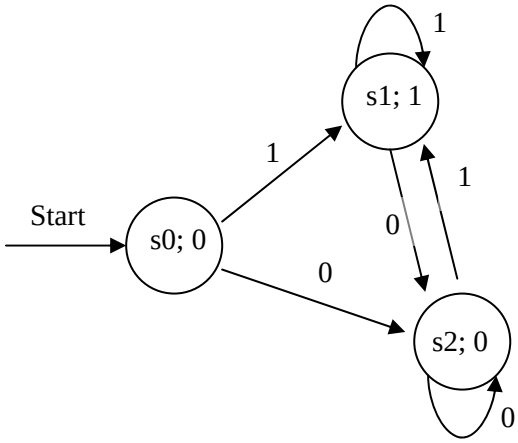
Figure 3. A Vending Machine.

Another way to illustrate a finite-state machine is a state table. The state table of a vending machine of the previous example is presented in Figure 4.

State	Next State					Output				
	Input					Input				
	10	20	50	C	T	10	20	50	C	T
s0	s1	s2	s3	s0	s0	n	n	20	n	n
s1	s2	s3	s3	s1	s1	n	n	30	n	n
s2	s3	s3	s3	s2	s2	n	10	40	n	n
s3	s3	s3	s3	s0	s0	10	20	50	Co	Te
						(Co=coffee, Te=tea, n=nothing)				

Figure 4. A State table of a Vending Machine.

The Moore form differs from the Mealy form by the correspondences of the outputs. In the Moore form the output is determined only by the states. In electronic devices a unit-delay machine produces as output the input string delayed by a specified amount of time. If the input string is x_1, x_2, x_3, x_4,…x_k, the finite-state machine produces as the output string 0, x_1, x_2, x_3, …x_{k-1}. A Moore form of a unit-delay machine is presented in Figure 5.



State	f: Input 0	f: Input 1	g
s0	s2	s1	0
s1	s2	s1	1
s2	s2	s1	0

A state transition function f assigns a next state to every pair of a state and an input.
An output function g assigns an output to every state.

Figure 5. A directed graph and a state table of a unit-delay machine.

These finite-state machines are used in computer programs for spell checking, grammer checking, recognizing speech, and network protocols, for example [3]. The states and input parameters and outputs in these applications are more like quantitative (strings of bits or numbers) than qualitative. Usually they are binary codes, which are transformed into the other binary codes. When a phenomenon is modelled by a technical process, states and transitions are qualitative aspects of involving things in this process. Mealy and Moore machines gives the basic rules for modelling the behaviour of things, but these models have had to be refined so that they are able to capture the better and more realistic model of the real phenomena. Refining efforts have been done in developing the third generation object-oriented modelling language UML for modelling the information systems.

3.2 Lifecycles as state models in object-oriented analysis

Pioneers in the developing process of the state machines for modelling the information systems are, among many other researchers, Sally Shlaer and Steven J. Mellor from Berkley California. In their book [2] a common behaviour pattern is abstracted into a lifecycle, which is formally modeled by a state model. Their model, which bases on Moore form, consists of a set of states, each of which represents a stage in the lifecycle of a typical instance of an object. Events, in this notation, represent incidents or indication that a progression is happening. A transition rule specifies what new state is achieved when an instance in a given state receives a particular event (function f in Figure 5). An output function (called action) is an activity or an operation that must be accomplished when an instance arrives in a state. Actions are assigned to states, which is a basic rule of the Moore form.
In this book [2] there has been modeled as an example a state model for a compact and inexpensive microwave oven, the one-minute microwave oven. In this oven there is a single control button for the user. If the door is closed and the control button is pushed, the oven will cook for 1 minute. Whenever the button is pushed during the cooking, the cooking time is added by 1 minute. Pushing the button when the door is open has no effect. Cooking is stopped by opening the door. There is a light inside the oven at any time when the cooking is on or when the door is open. If the door is closed and the cooking is not on (the control button is not pushed) the light goes out. If the cooking times out the power tube and light is turned off and a warning beep tells that the food is ready. A state model for the one-minute microwave oven is presented in Figure 6.

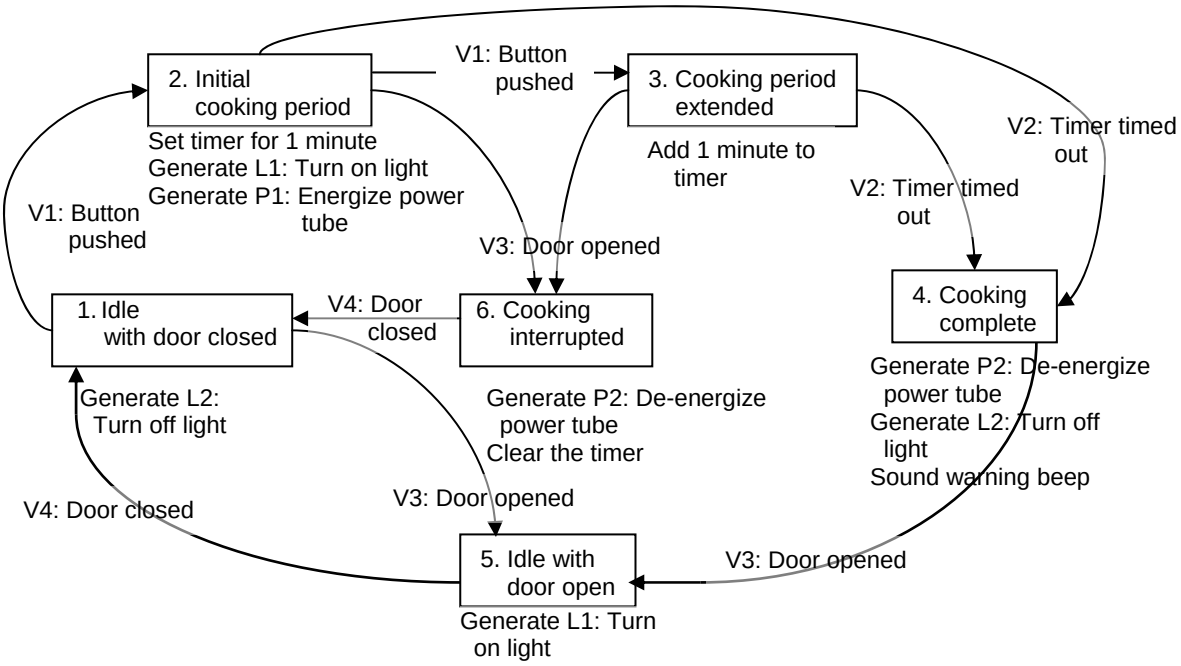


Figure 6. State model for the one-minute microwave oven [2].

The state model by Sally Shlaer & Steven J. Mellor belongs to first generation OOA methods. The numbered and textually phrased states are in rectangles. The transitions are shown by arcs connecting two states. Each transition is labelled with the event that causes the transition. The actions associated with states are described under the state boxes. A state represents a condition of the object in which a defined rules, policies, regulations, and physical laws applies. An event is the abstraction of an incident or signal in the real world that tells us that something is moving to a new state. As it is in the finite-state machines of the Mealy or the Moore form, an event is an input to the system. An input can come outside (active environment, user interface) or inside the system created by an action. A state transition function assigns a next state to every pair of a state and an input. An action is an activity or an operation that must be done upon arrival in a state. It is an output function that can [2]:

1. Do any calculation.
2. Generate an event to any instance of any object
3. Generate an event to outside environment (hardware, actors, sensors, other subsystems).
4. Create, delete, read, set, reset a timer
5. Read and write attributes inside or outside the modelled object.

By using this state modelling method the real behaviour pattern of a phenomenon can be captured into a technical process that is a reactive state transition model of an event chain from an initial unwanted state to a final satisfactory state in an active environment. Other famous first generation state models have been developed by James Rumbaugh, Ivar Jacobson and Grady Booch in their own object-oriented modelling methods, which are: the Booch method, published in 1991, the Object Modeling Technique (OMT) by Rumbaugh et al., in 1991, and the Objectory (OOSE) by Jacobson et al., in 1992.

It is obvious that the former presented state model grows to an enormous jungle of boxes and arcs when trying to capture the real behaviour of a little more complicated reactive systems. David Harel from the Department of Applied Mathematics, the Weizmann Institute of Science, Rehovot, Israel, has developed the theory of state machines and he has presented a broad extension of the conventional formalism of state machines in his article: “Statecharts: A Visual Formalisms For Complex Systems”, which was published in the scientific paper: “Science of Computer Programming 8 (1987)” [6]. The theory of David Harel is a foundation of modelling the behaviour of the systems by using the third generation object oriented modelling language UML.

3.3 State machines in the third generation UML

Both Mealy and Moore form are possible in a same state machine of the third generation UML. Actions, which are assigned to states, are divided into an entry (Moore form) and an exit of a state transition. A function that takes place during a state existence is called an activity in the UML. The most noticeable departure of earlier state models is the nesting of states within state. A simple example, presented in the book: "Doing hard time: developing real-time systems with UML, objects, frameworks, and patterns", written by Bruce Powel Douglass [5], is a dual speed multiheat blower.

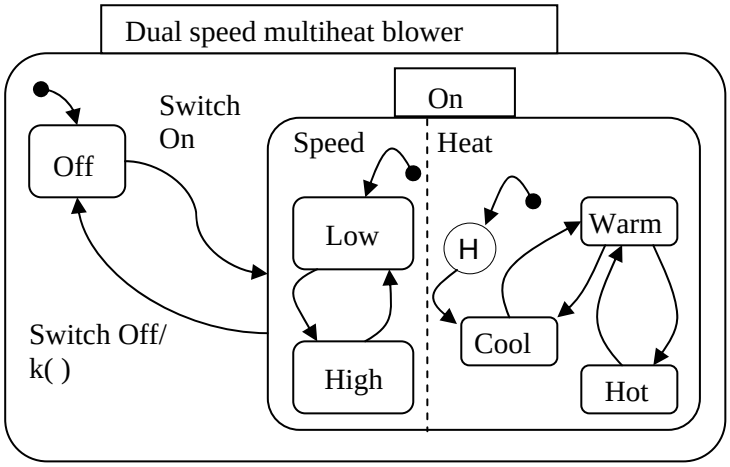


Figure 7. State model for a dual speed multiheat blower [5].

The blower can be *On* or *Off*. When the blower is on, the speed of the blow can be in one of the two nested substates: *Low* or *High*. The *On* state is divided by a dotted line into two orthogonal states, which are the speed and the heat of the blower. The speed and the heat represent independent states that are concurrently active. The heat of the blower may be *Cool*, *Warm* or *Hot* in the same time the speed of the blower is high or low. The speed and the heat are *and*-states. In each level of nested and orthogonal states there is a start transition marked with an arc from a filled circle to an initial state. When the blower is turned on the initial speed is low and the initial heat is cool when entered at first time. In *Heat* state the start transition arc ends to a circle inside which is a letter H. This means the system's *history* in a state. The most simplest kind of this "enter by history" is entering the state most recently visited. This means that whenever the blower is turned on the heat is in the state it was when the blower was turned off in the previous time. Events (*Switch On*, and *Switch Off/k( )*) trigger the state transitions. According to Mealy form the action k( ) is assigned to the state transition that switches the blower off.

The UML defines four kinds of events [5]:

1. *Signal Event* – an occurrence of interest arising asynchronously from outside the scope of the state
2. *Call Event* – an explicit synchronous notification of an object by another
3. *Change Event* – an event based on the changing of an attribute value
4. *Time Event* – either the elapse of a specific duration or the arrival of an absolute time

Events have neither duration nor persistence they just trigger the state transition. In modelling a technical process, *signal events* are interesting because they are sent asynchronously from outside the scope of the state. For example properties of an active environment are changed suddenly. There may be happened collisions, the quality of sliding surface may be changed dramatically (geometry, friction). Our system should be prepared to handle these events.

An event that triggers the state transition may have different kinds of transition parameters. The complete syntax for transitions is [5]:

event-name(parameters) [guard] / action list ^ event list

Parameters form a comma separated list containing the names of data parameters passed with the event signal. *Guard* is a Boolean expression that must evaluate to TRUE for the transition to be taken.

Actions executed as a result of the transition to be taken are in a comma-separated *Action list (Mealy form)*. An event can generate other events on orthogonal regions of the state or on other objects. These events are in a comma-separated *Event list* in the syntax for transitions.
Structural things as markings for an initial and terminal states are called *pseudostates* in the UML. *A Conditional connector* and an earlier mentioned and defined *History* are also these pseudostates. The conditional connector is used for branching based on guards. It is equivalent to the statement: "if-then-else".

4 MODELLING A TECHNICAL PROCESS BY USING STATE MACHINES

If an electric shaver is taken as an example for the product definition, the primary problem is: "How the beard on the face is cut as short as possible without damaging the skin under the beard?" As an active environment the electric power is available from the plug or from included loadable batteries. The body of the device can be manufactured handy and light. The electric power is able to transform easily to reciprocating or rotating mechanical movement. The principal idea how the technical process captures the needed functionality and how it manages the problem solving is presented in Figure 8.

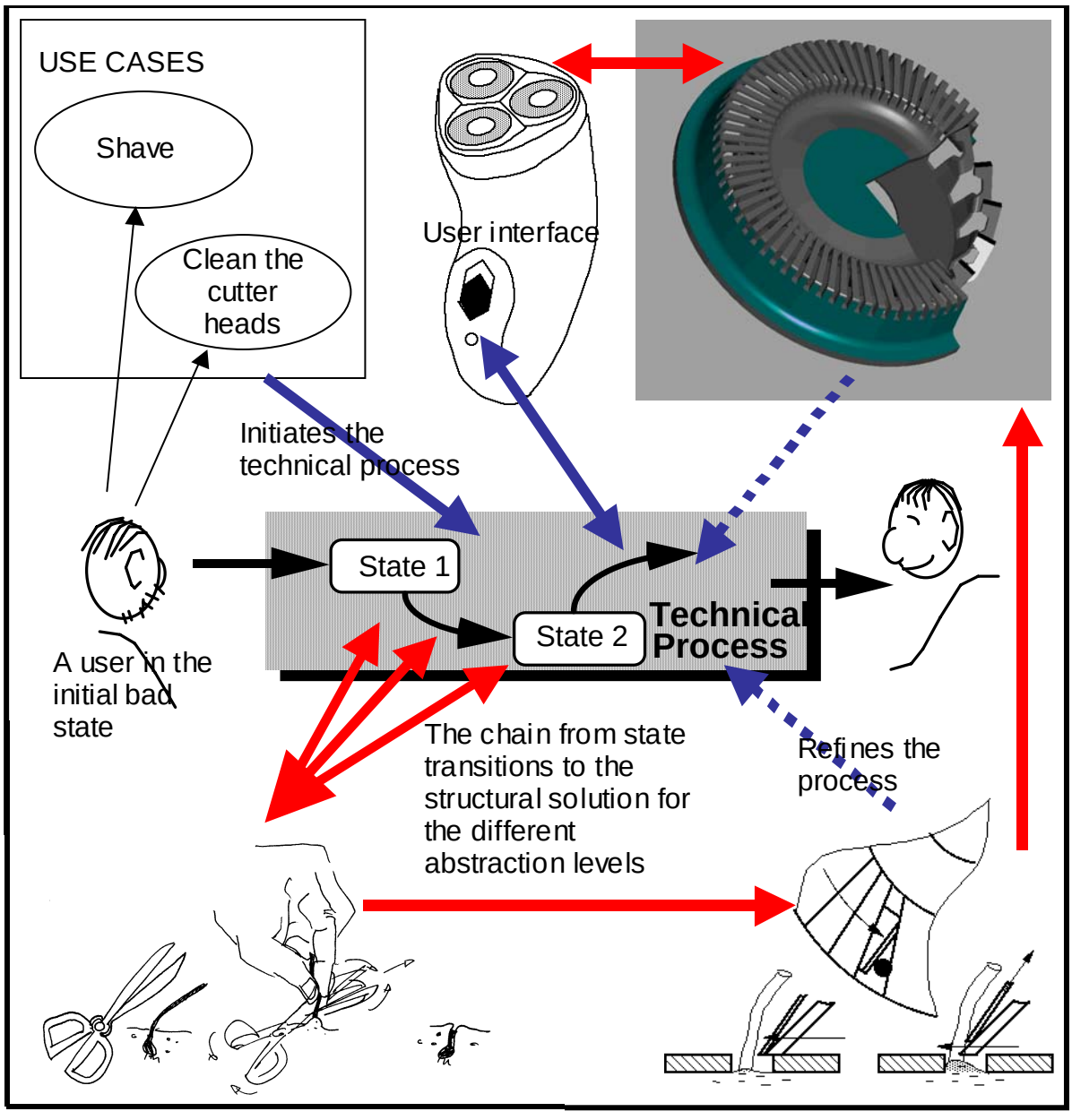


Figure 8. The technical process manages the design process of a shaver.

The use cases in the upper left corner of the Figure 8 are the using functions of the outside world of the system. They describe the needed using scenarios of some actors of the system. These actors may be human, another systems, or an active environment, for example. An use case initiates a technical process. The use case "Shave" may include the next list of activities:

1. Take the shaver into your better hand so that you can easily slide the cutter head on your chin.
2. Turn the cutter on.
3. Lay the cutter head perpendicular into your hairy chin.
4. Slide the cutter head on your chin.
5. Lift the cutter head from the chin.
6. Slide the fingers of your free hand on the place of the chin just shaved.
7. If the result is not good enough go back to line number 3.
8. Or else if the result is acceptable and there are still hairy places on the chin left, move to the hairy place and return to line number 3.
9. Or else, turn the cutter off and put the shaver in the place from where you took it.

If this shaving operation is modeled as a state model according to the state modeling method of Shlaer & Mellor presented in Figure 2, it might look as in the Figure 9.

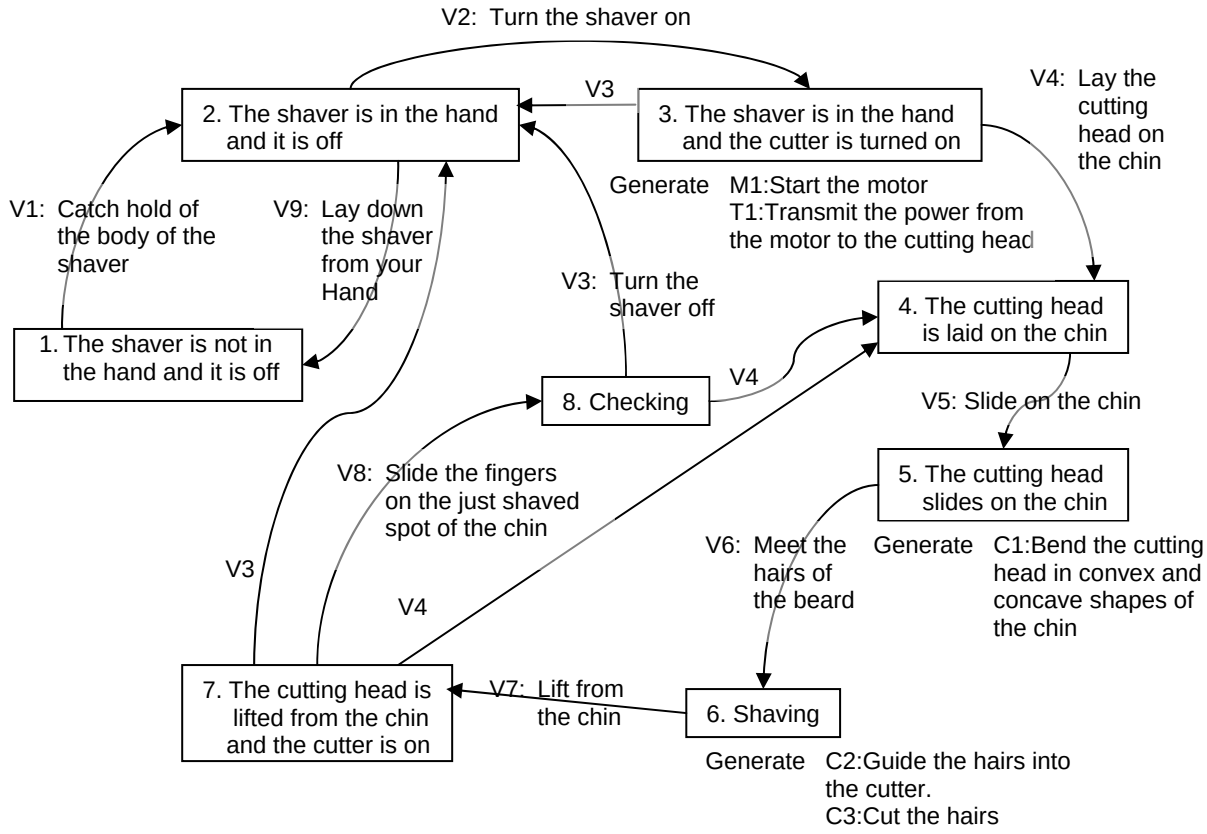


Figure 9. The state model of a shaving process according to the state model by Shlaer&Mellor.

In the state model of a shaving process there are eight different states (numbered in boxes in Fig. 9), eight different events (V´n´ in Fig. 9)), and actions, which are assigned to states. For example, when the cutting head of a shaver is slid on the chin (state 5. in Fig. 9) and it meets hairs of the beard a state transition to *Shaving* takes place. Actions *C2:Guide the hairs into the cutter* and *C3:Cut the hairs* are assigned to the state *Shaving*. These actions are events, which are sent to an object or an actor marked with an upper case letter C. In the structure of a shaver the cutting head is marked by the letter C. A solution principle for this functionality is presented in the Figure 8. The state model of a shaving process modeled by using the third generation UML is presented in the Figure 10.

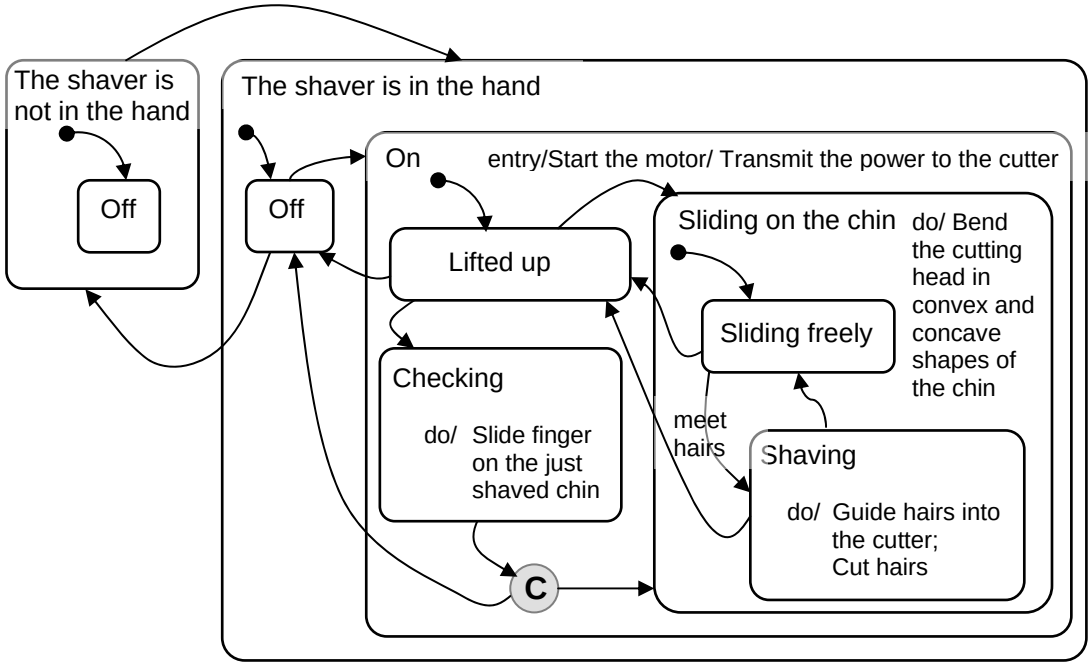


Figure 10. A state model of a shaving process according to third generation UML.

States in an UML state model are able to be nested. Actions (entry/…, exit/…) and activities (do/…) are included into the boxes of states which control and trigger them during the process. There is a conditional transition when checking the result of shaving marked with letter "C" in a shaded circle. If the result of shaving is good enough, the shaver is turned off, else the shaving goes on by sliding the shaver on the chin.

Professor Alexander Horowitz (1904-1982), Belgium to parents of Russian Jewish heritage, invented the first rotary electric shaver, using rotating cutters instead of the reciprocating cutters, which was invented earlier by Jacob Schick in United States of America. Philips company introduced the first Philishave rotary electric shaver to the European marketplace in 1939 just when World War II was beginning in Europe (http://iavbbs.com/gflinn/biohorow.htm). In the next figure there has been presented how some of the requirements found out in the previous state models, have been solved in the Philishave rotary electric shaver.

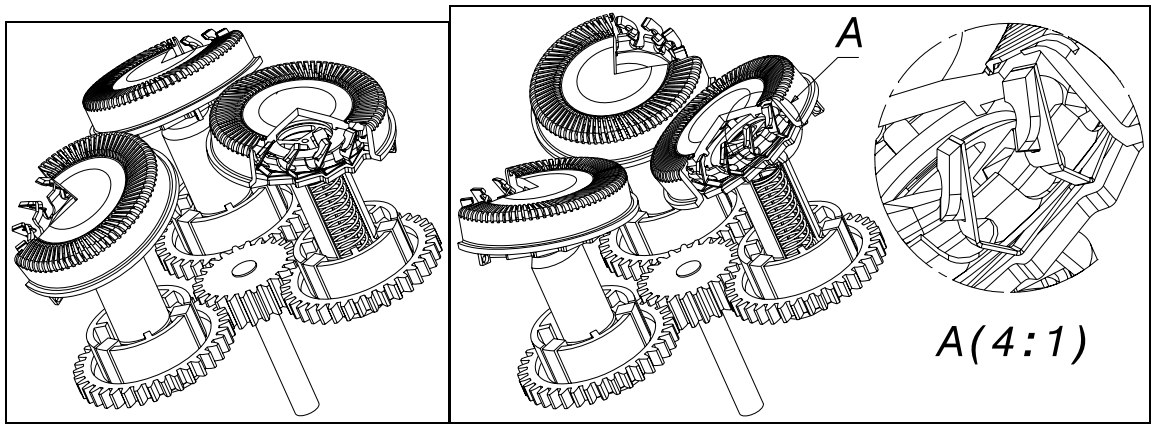


Figure 11. The Philishave rotary electric shaver.

The power is transmitted from the electric motor to the cutting heads by a planetary gear train. Flexible couplings of the shafts make possible the bending of the cutting heads in convex and concave shapes of the chin. The grille-slots in the circumference of the body cups of the cutting heads guide the hairs of the beard into the scissors based cutting process as it is illustrated in the Figure 8.

5 CONCLUSIONS

In our paper we have introduced ideas how the real world phenomenon can be captured into the state transition model, which is as realistic as possible and which can be used as a foundation of design process of a technical product. In the Theory of Technical Systems (TTS) by Hubka and Eder [1] the technical process is modelled by a state transformation process from an initial unsatisfactory state through intermediate states to a final satisfactory state. In this process the operands, which are classified into: biological objects, materials, energy and information are transformed by effects. The effects are produced by the users, or by the technical system, which is to be designed, or by the active environment. The effects can contain or consist of material, energy, and/or information. In the Use Case driven state modelling the starting point is the needed using scenario of some actors of the system. These actors may be human, another systems, or an active environment, for example.

There are a lot of similarities between the theory of technical systems by Hubka & Eder 1988 [1] and methods used in object-oriented analysis. The needed effect to achieve the transformation from an initial unsatisfactory state to a final satisfactory state is produced by the technical system, human or the active environment. Similar to this principle is to abstract a common event driven behaviour pattern to produce the lifecycle of an unspecified instance of an object by using state transitions diagrams used in the OOA [2], and state models in the system level used in the UML [5].

In the TTS functions are means to achieve effects for state transitions. Function carriers (physical laws, material behaviour, for example) are means to achieve the needed function in the TTS. These function carriers are called organs in the TTS and they are achieved by the part and assembly structures. This theory of four different views of a system has been developed as a genetic model for the result of the design task, the chromosome model [7]. When modelling a use case via a state model, a real functional behaviour of a using scenario of a system is captured into this state model. Functions which are needed during this process are allocated and located in exact points of the state transition chain they have to be triggered and to be sent inside or outside the scope of the use case.

In requirements modelling of a system, the external properties of a product are commonly clarified at first. Typical users, user interfaces, and the typical using scenarios of the different kind of users are these kinds of external properties. The UML offers formal methods for this definition process in carrying out using actors, use cases, use case statecharts, and sequence and collaboration diagrams. These implementation free models of the system are refined and updated all the time during the whole product development process. Besides defining the needed information and behaviour of the system modules, these models capture and document the real creating history of the final structures in the product.

REFERENCES

[1] Hubka, V. & Eder W.,E.: “Theory of Technical Systems”, Springer-Verlag, New York, 1988.

[2] Shlaer, S. & Mellor, S.,J.: “Object Lifecycles: Modeling the World in States”, Englewood Cliffs, New Jersey, Yourdon Press, 1992.

[3] Rosen, K., H.: “Discrete mathematics and its applications”, McGraw-Hill, Boston, 2003

[4] Moore, E., F.: “Gedanken-experiments on Sequential Machines.” In Automata Studies; Princeton University Press, Princeton N. J., 1956.

[5] Douglas, B., P.: “Real-time UML: developing efficient objects for embedded systems”, Addison-Wesley Longman Inc., United States of America, 2000.

[6] Harel, David,: “Statecharts: A Visual Formalisms For Complex Systems”, Science of Computer Programming 8, pp. 231-274, 1987.

[7] Andreasen, M.,M.,: “The Theory of Domains”, Workshop on Understanding, Function, and Function to Form Evaluation, Cambridge University, UK, 1991.

Contact: Tapio Korpela
University of Oulu
Department of Mechanical Engineering
P.O.Box 4200, FIN - 90014 University of Oulu
Finland
Phone +358 8 553 2052
Fax +358 8 553 2026
e-mail Tapio.Korpela@me.oulu.fi