PROCEEDINGS

55. IWK

Internationales Wissenschaftliches Kolloquium
International Scientific Colloquium

13 - 17 September 2010

Crossing Borders within the ABC

Automation,

Biomedical Engineering and

Computer Science

Faculty of
Computer Science and Automation

www.tu-ilmenau.de

TECHNISCHE UNIVERSITÄT
ILMENAU

Impressum
Published by

Publisher:	Rector of the Ilmenau University of Technology Univ.-Prof. Dr. rer. nat. habil. Dr. h. c. Prof. h. c. Peter Scharff
Editor:	Marketing Department (Phone: +49 3677 69-2520) Andrea Schneider (conferences@tu-ilmenau.de) Faculty of Computer Science and Automation (Phone: +49 3677 69-2860) Univ.-Prof. Dr.-Ing. habil. Jens Haueisen
Editorial Deadline:	20. August 2010
Implementation:	Ilmenau University of Technology Felix Böckelmann Philipp Schmidt

USB-Flash-Version.

Publishing House:	Verlag ISLE, Betriebsstätte des ISLE e.V. Werner-von-Siemens-Str. 16 98693 Ilmenau
Production:	CDA Datenträger Albrechts GmbH, 98529 Suhl/Albrechts
Order trough:	Marketing Department (+49 3677 69-2520) Andrea Schneider (conferences@tu-ilmenau.de)

ISBN: 978-3-938843-53-6 (USB-Flash Version)

Online-Version:

Publisher:	Universitätsbibliothek Ilmenau ilmedia Postfach 10 05 65 98684 Ilmenau

Home / Index:
http://www.db-thueringen.de/servlets/DocumentServlet?id=16739

SEQUENCE-BASED SPECIFICATION AS A NEW KIND OF MODELING EMBEDDED SYSTEMS

Prof. Dr. Reinhold Schönefeld

Schulung & Beratung für Software, Ilmenau

ABSTRACT

Sequence-based Specification is a long existing, but not so well-known approach, which is suited to specify the software of embedded systems formally. To reduce the reservation against a formal specification, the focus in the paper is directed to a more ”physical view” of the specification process. The paper shows some interesting principles in software engineering, which are analogous to other principles in physics supporting a deeper insight into the “movement” of a software system.

1. INTRODUCTION

An embedded or interactive software system is an open system interacting with its environment. To specify the functional requirements for such a system, there is the so-called sequence-based specification (SbS) available. The first paper about specifying a software system by sequences of input variables goes back to 1978 and was published by David L. Parnas and W. Bartoussek [1]. Parnas and his coworkers have spent a lot of research since this first paper to develop a specification methodology. In 2003 Stacy J. Prowell and Jesse H. Poor [2] published their paper about the sequence-based specification.

A specification of an open software system is an abstract conceptual model of the behavior of the not yet existing software system, here called an MSS. The outside behavior of the MSS is how the MSS reacts to a specific sequence of inputs. This view on the MSS is the well-known black box view

BB: $S^* \rightarrow R$ (1)

It is a function where S^* is the set of all input sequences and R the set of all reactions. In this view no internal variables like the state have to be considered. The models in [1] and [2] and other well-known models like a deterministic finite automaton are except the black box view dealing with the state concept and state transitions caused by certain inputs to the MSS. The outside behavior of the MSS can also be described by a state box view. It is determined by the state box function SB.

SB: $(Z \times S) \rightarrow (Z \times R)$ (2)

Z is the set of all state variables and S the set of all inputs. In (2) we need an initial state. The dissertation of St. J. Prowell in 1996 [3] shows how to construct the specification functions (1) and (2) out of the given informal functional requirements by developing an enumeration schema.

Both functions are part of the box structure methodology which belongs to the unfortunately not so well-known research results of Harlan D. Mills and his coworkers [4]. The box structure methodology is completed by a third box, the so-called clear box view. Here the MSS will be instantiated in form of the architecture and the implementation of the realized software system.

The focus in this paper is directed to a more “physical view” to the SbS as a whole and the enumeration schema, differently from being demonstrated in [3]. A physical view should look at an MSS as an open system which is stimulated by some cause and which answers by some reaction. We will show that an MSS and its correct implementation behave analogue to a mechanical system. In mechanical systems well-known fundamental relations exist between the describing variables, leading to axioms and equations. The following pages are a first attempt to demonstrate that there are also some fundamental relations between the describing variables in an MSS and of course in its implementation respectively if it is carried out correctly.

2. MESSAGES CARRYING SEMANTICS

The cause of the movement of an open mechanical system is a force. In an open electrical system a voltage is acting as a cause. In an MSS we usually speak about an input. Here we will use the terminus stimulus from the box structure methodology, more typically expressing the character of a cause.

But what informational substance belongs to a stimulus? The extension principle of semantics gives us an answer [5]:

- The semantics of an expression of a natural or artificial language is sufficiently defined by its

extension that means what real or abstract objects are named by that expression.

- The extension of a composed expression in this language is consistently defined by the extensions of the parts of that expression and the kind of their composition.

The extension of a concept or a notion is its volume, which means the set of real or abstract named objects belonging to this concept or notion respectively. The second point is also known as the Frege principle or principle of compositionality and is the fundamental assumption for all standard approaches in the field of semantics. We will use it later on.

According to the first point each stimulus that we can find and distinguish from each other in the informal functional requirements carries semantics. This semantics is derivable from the extension of each stimulus found. The extension by itself is mostly a set of equivalent stimuli. Using the concept of forming equivalence classes reduces the set of equivalent stimuli to one single representing stimulus, carrying the semantics of the whole equivalence class. This is the way to find the semantics of each extractable stimulus from the functional requirements. Note that the extension set could also contain only one element.

Stimulus is a notion coming more from systems theory and, as already mentioned, is used in box structure methodology [4] and SbS [3]. The usual terminus used in software engineering is message. A message from the environment of the MSS causes a "movement" or process in the MSS like a force or a voltage does in mechanical or electrical systems. As we can see so far is that a message carries semantics into the MSS. Semantics of what happened in the environment.

The outside source of a message to the MSS is an event in the environment at a certain time. But acting as a message are only these events which can pass the interface or the boundary of the MSS. Therefore it is recommended to design a graphical user interface at first in order to support the extraction of the messages from the verbally formulated informal functional requirements.

3. CONTINUOUS VERSUS DISCRETE SYSTEMS

There is nevertheless an important difference between a force or a voltage and a message. Force and voltage are continuous quantities over time and

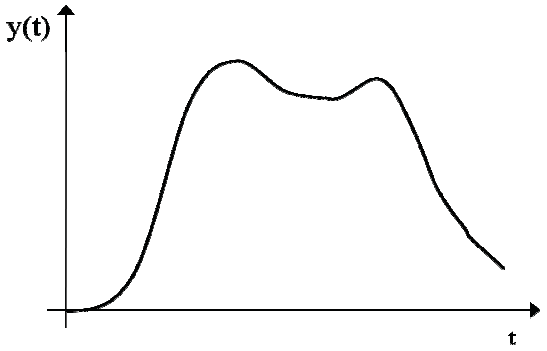

Figure 1 Continuous variables

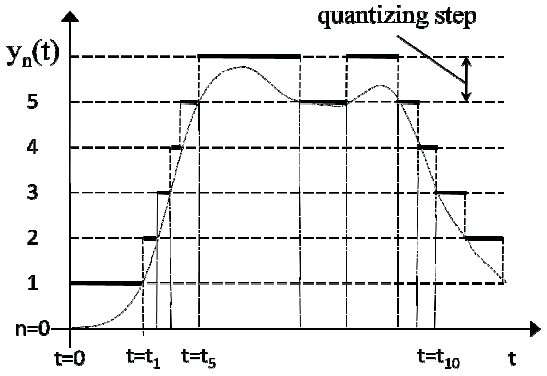

Figure 2 Event-discrete variables

in its values, whereas a message is a discrete quantity in time and its values. To find out the differences somewhat more precise let's look at the typical features of continuous and discrete systems.

In continuous systems as classical mechanical systems and electrical systems the time and the values are real-valued variables. That means the independent and the dependent variables are infinitely tightly located at its axes (Figure 1). As mathematical models of mechanical or electrical systems are used differential equations. It is important to recognize that these equations express an instantaneous equilibrium of the involved mechanical or electrical variables. That means even the smallest change of the causing force or voltage in the environment leads to an instantaneous change of quantities inside the considered system.

As usually known, digital information processing systems are discrete systems. Because of the finite length of a computer word, only rational numbers can be processed. This already leads to small quantization steps. In embedded systems are often introduced greater quantization steps intentionally, to reduce the amount of data which has to be processed. Very often we can find binary variables as messages to the discrete system. Such quantization steps lead to an event-discrete variable, as we can see in Figure 2. As soon as the values

inside a quantization step are leaving from the lower or upper step, the variable is changed discontinuously. Just at this point of time the event takes place, which is acting as a message to the MSS. All the event-time points in Figure 2 could be replaced by integer numbers, numbering the sequence of discrete events of one variable in the environment of the MSS.

It is important to realize that the quantization implies only finitely small changes of the values of a variable. Note that the quantizing step in Figure 2 does not have to be an integer but could be any symbolic range limit e.g. likes "low", "normal" and "high" instead. The quantization of the involved variables is the characterizing difference to a continuous system. The arrival of a causing message always carries a finite quantum of information into an MSS and as we know from chapter 2 it is a quantum of semantics.

4. MESSAGE CAUSES STATE CHANGE

In the following we will see that despite the demonstrated difference between continuous and event-discrete systems there is an analogue between both kinds of systems concerning the relationship of external cause and internal effect.

In case of a continuous mechanical system (Figure 3) a force is the external cause and the internal effect is a potential energy stored in the spring k. If we put a force at the system as in Figure 3 at times $t \leq 0$ we can see that the causing force is continuously stretching the spring and instantaneously storing a potential energy in it. In mechanics it is shortly called a potential. From this follows a force is equal to a negative change of a potential.

$$m\ddot{x} = -\frac{\partial}{\partial x}\Phi(x) \qquad (3)$$

$$m\ddot{r} = -\nabla\Phi(x,y,z) = -\left(\frac{\partial\Phi}{\partial x}e_x + \frac{\partial\Phi}{\partial y}e_y + \frac{\partial\Phi}{\partial z}e_z\right) \qquad (4)$$

This equality is a different approach to a force in Lagrangian mechanics in comparison to Newtonian mechanics, in which a force is considered the second timely derivation of the spatial position. Equation (3) holds for the one-dimensional case like in Figure 3. $\Phi(x)$ is the scalar potential function. Equation (4) holds for the three-dimensional case with *r* as the position vector and the ∇-operator. Fundamental forces are generally conservative in the universe, only frictional forces, here neglected, are not conservative. In a continuous electrical field we can find the analogue equations between the electric field strength and an electrical potential, which underlines the generality of the equations (3) and (4).

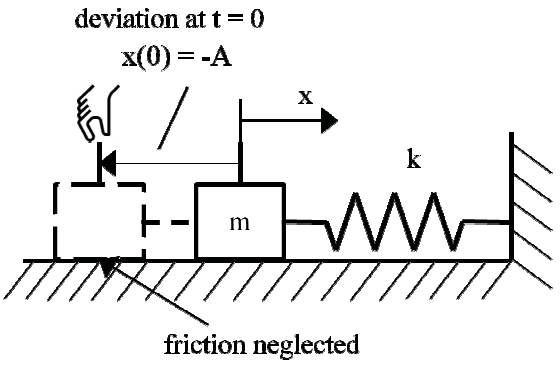

Figure 3 Mechanical oscillator as conservative system

To study the relationship between external cause and internal effect in case of an event-discrete MSS it is sufficient to consider a deterministic finite automaton (DFA):

$$M = (Z, S, z_0, Z_F, \delta) \qquad (5)$$

With Z a finite set of states, S a finite set of messages also called the alphabet, z_0 an initial state, $Z_F \subseteq Z$ a finite set of accepting final states, and $\delta: Z \times S \rightarrow Z$ the state-transition function. It is well-known that the DFA is stimulated from the environment by a sequence of messages $x = s_1 s_2 \ldots s_m \in S^*$. With each message the DFA instantaneously transfers from its actual state in a deterministic new state, which is given by its state transition function δ. The initial state z_0 has to be known. Each sequence of external messages $x \in S^*$ generates a sequence of internal states. When the last message s_m is sent, the DFA can decide by the final state reached, whether $x = s_1 s_2 \ldots s_m$ is an acceptable sequence of messages $x \in Z_F$ or not.

It is obviously the finale state which contains the history of all previous states and messages starting with z_0. Figure 4 illustrates the DFA as a simple box structure. Here $s_0 = \lambda$ is the empty sequence and z_0 is the initial state. After each incoming message s_i the state instantaneously transfers from z_{i-1} to z_i. As we can see each new state z_i is the result of a superposition of two different informational quantities - the external message s_i and the internal old state z_{i-1}. From this follows an

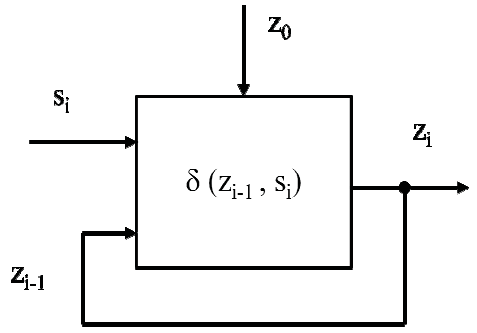

Figure 4 State-transition function of a DFA as a box structure

external message has to be equal to a change of the internal state.

This is in principle a well-known fact, but the value of this fact is the analogue between a continuous mechanical system and an event-discrete DFA. From this analogue we can conclude to more interesting facts in an MSS already known in mechanical systems, as later will be shown. The equality between message and change of state also holds in all other models where the state concept is used. In the continuous system the change is represented by the partial derivative in space of the potential function, see (3) and (4). In an event-discrete system a derivative with respect to time and space is not defined because of the discontinuous properties of the state-transition function.

In spite of this we know precisely at what amount the state is changed after a message stimulates the DFA. The change is equal to the quantum of semantics that this message carries into the DFA or e.g. into the state box function (2) as another model, based on the state concept. As a result each message is accumulated by the DFA to a new state. As soon as the state changes, a new coordinate is added to the state space of the DFA. This leads to an m-dimensional state space or state vector, if m messages arrive in the DFA. State space could even be better called state transition space, because each message causes a state transition.

With a view on semantics it means that each new state carries a semantics which is composed from the parts of semantics caused by each single message, in accordance with the Frege principle or principle of compositionality above cited. Thereby the parts are composed as a sequence of messages. Both the m-dimensional state space and state vector represent the whole formal semantics of the sequence of messages $x = s_1\ s_2\ \ldots\ s_m$.

5. ESSENTIALS OF SBS UNDER A NEW VIEW

A specification as a BB- (1) or a SB-function (2) has to describe the functional behavior of a software system in a general manner, which means it has to abstract from its different possible instances. In the box structure methodology the different possible clear boxes are instances of the software system. In other words a specification should always be a representative of a class of equivalent instances of a software system. An instance is e.g. a concrete operational architecture or a concrete implementation of the software system. From this follows, the creation of a specification is the development of a universal conceptual model of the not yet existing software system – shortly called an MSS.

Preparing the SbS requires to find the sets S and R. The two once contain all the information crossing the interface or systems boundary of the MSS in both directions. The easiest way to gather all elements of S is to document all the interfaces of the MSS in form of a GUI. To find the elements of R it needs a natural language specification, which describes the intended software reactions for all cases of use. It is useful to formulate this verbal specification as a rule and represent it as a table by tagging each entrance for traceability reasons. An example of an entrance of a required reaction could be formulated as follows: “If a true three-digit code is given, the alarm tone will be stopped.” Neither GUI nor verbal specification has to be complete or consistent. They can be corrected during developing the SbS.

Wherein the new view does exist to extract the messages out of the GUI? Note that S is a set of messages from different sources, e.g. any sensors or input fields. Each single message could be recognized considering the extension of this message. As example look at the input via GUI of the first true digit of a three-digit code, assumed it be “3”. To find the message we have to abstract from the “3” and have to express the message more abstract, in the sense what the “3” means to the MSS, as e.g. “first true digit crossed the interface”, shortened with the name “d1”. The semantics of d1 is shortly spoken “first true digit”. Note that each message of S should be found by abstraction.

What kind of an abstraction have we used? In the extension of the first true digit in the code

Sequence	Reaction	Equivalent. Sequence	Trace	Compositional State		
				State Var. 1	State Var. 2	…
λ	Null			none	none	…
…	…	…	…			…
a reducible sequence	r_i	equivalent sequence	tag- no k			…
…	…	…	…	initial state value		…
a canonical sequence	r_j	no equivalent sequence	tag- no l	new state value		…
…	…	…	…			…

Figure 5 Fragment of the enumeration schema

we can find anyone of all digits 0 … 9. The extension by itself is generally a set of concrete real ore conceptual objects, which a human being can comprehend. Characteristic for an extension set is that it is always an equivalence class. A chosen representative from this class is created by a classification abstraction. An abstract message is then always the representative carrying the semantics of the extension. From this we can conclude finding a message with a related semantics requires a classification abstraction of the extension set of this message.

An enumeration is the systematic construction of a schema by listing sequences from S* by length and any but fixed order within a given length and for each sequence assigning a reaction (Figure 5). The enumeration starts with λ, the empty sequence. The connected reaction is Null because the MSS is not yet stimulated by any message. After λ follow all sequences of length one, which are all messages of the set S. Each sequence is mapped to its appropriate reaction out of R. Thereafter all sequences of length two are established by combination and each sequence is mapped to its reaction. Then we combine sequences of length three and so on.

A combination is carried out by extending all appropriate sequences of the actual length with all the messages belonging to S. In forming the combination we don't have to extend all actual sequences because of two reasons:

- Illegal sequences are without semantics, because they cannot be generated in the environment or cannot be observed by the MSS. Illegal sequences do not have to be extended to a sequence of the next length. They are registered with ω.
- Equivalent sequences do not have to be extended either, because they lead to the same state in the MSS (see chapter 4) as the state generated by a previously mapped sequence.

All equivalent sequences carry the same composed semantics and form an equivalence class. Therefore they can be reduced to one representative in form of a shorter or equally long equivalent sequence. This entire means, there are only a few sequences of the actual length, which can be extended. These sequences change the state and consequently build a new composed semantics and can be extended. Such special sequences are the so-called canonical sequences.

The history or prefix "p" of each sequence u = p s therefore is a canonical sequence. "s" is the actual message. That holds also for the canonical sequences themselves. Hence λ is a canonical sequence. All fond canonical sequences are growing stepwise by length up to the longest length the enumeration unfolds. Since each equivalent sequence in the third column has the same state as a previously sequence it must be a canonical sequence. From these facts we can conclude that the history "p" represents the old state and the equivalent sequence the new state. So we can conclude the enumeration contains all information to build the functions (1) and (2). The enumeration is continued until all sequences of a given length are illegal or equivalent to previous sequences. Figure 5 gives a certain impression how the enumeration schema is built.

If during the enumeration the analysis of the canonical sequence just found takes place immediately, the enumeration schema has to be enlarged in comparison to [3], with the advantage that the decision about equivalence is based on state or semantics and not on reaction. λ as the first canonical sequence determines the initial state in the state space.

Each next found canonical sequence creates a new state variable the name of it is chosen following the semantics that carries the actual message. The value of that state variable is equal to the amount of the semantics the state variable has changed at the event-time. The value each state variable has before the state step happens is its initial value (z_0). The initial values of all state variables determine the origin of the state space. Each line of the compositional state in Figure 5 should be disjoint from each other. If not a mistake occurred with an equivalence decision.

Throughout the development of the enumeration it often happens that there is no given reaction for a combined sequence. Here we have to derive the required verbal specification supported by the knowledge of domain experts. The enumeration is a mathematical function and as such complete, consistent and in respect to the verbal specifications correct. Correctness is supervised by traces to the tagged verbal specifications (see column four). Moreover enumeration is a systematic process for discovering omissions and ambiguities.

6. CANONICAL SEQUENCES AS PATHS OF COMPACT SEMANTICS

The movement of a continuous mechanical system obeys a fundamental law – the principle of least action [6]. If we look at the curve of a ball thrown between two persons, then this movement complies with that principle in the gravitational field. The least action says simply spoken, the curve or path the ball will take needs a minimum of the physical quantity action. The action is the product of energy and time. If the time should be shorter, the ball arrives at the other person; the energy should be higher and vice versa. Strictly speaking the action is the integration of the Lagrangian function over the time.

The scientific history of this fundamental law started with the observations about the movement of light by Fermat (1607-1665), known as Fermat's principle and ended with W. R. Hamilton (1805-1865) who completed this theory. The basic mathematical apparatus, belonging to this principle is well-known as the Lagrangian- or Hamiltonian mechanics [7], [8]. The starting point of this approach is given by equations (3) and (4), the equality of force and change of potential.

As demonstrated in Chapter 4 there is an analogue between mechanics and informatics despite the different system properties. Therefore one could look for if there is an analogue for the principle of least action in informatics, especially in an MSS of an embedded software system. The latter is best described by equations (1) and (2), which can be derived from the enumeration schema, as shown in [2] and [3].

The analogue principle in an MSS really exists in form of the canonical sequences. Each legally combined message sequence creates a path in the state space according to the equality of a message and a state change. This path is not continuous in three dimensions like in mechanics, but is event-discrete. Each path starts in the origin of ordinates at the event-time t_0 and goes in steps at each incoming message to a final point in the m-dimensional state space at the event-time t_m, if the sequence has m messages. All equivalent sequences are always longer or equal to the canonical sequences. Therefore only the canonical sequences build a path of least length or with the least number of state steps. Examples of state transition diagrams have prove the hypothesis of existence of an analogous principle in informatics. With the focus on semantics the principle says that only the canonical sequences carry the composed formal semantics of the MSS in the most compact way. All equivalent sequences carry redundancy that is why the can be ignored. "Principle of most compact semantics" may be is a name for it.

7. REFERENCES

[1] W. Bartoussek and D. L. Parnas, "Using Assertions about Traces to Write Abstract Specifications for Software Modules," Proc. Second Conf. European Cooperation in Informatics and Information Systems Methodology, pp. 211-236, 1978.

[2] St. J. Prowell and J. H. Poor, "Foundations of Sequence-Based Software Specification," IEEE Trans. on SE, vol. 29, pp. 1-13, 2003.

[3] St. J. Prowell, "Sequence-based Software Specification," Diss. Uni. of Tennessee, May 1996.

[4] A.R. Hevner and H.D. Mills, "Box-structured methods for systems development with objects," IBM Systems Journal, vol. 2, No.2, 1993.

[5]http://de.wikipedia.org/wiki/Extensionalitätsprinzip

[6]http://en.wikipedia.org/wiki/Principle_of_least_action

[7]http://en.wikipedia.org/wiki/Lagrangian_mechanics

[8]http://en.wikipedia.org/wiki/Hamiltonian_mechanics