AASE - Actor-Actor Systems Engineering Theory & Applications Micro-Edition (Vers.9)

eJournal: uffmm.org, ISSN 2567-6458
13.June 2018
Email: info@uffmm.org

Gerd Doeben-Henisch
Louwrence Erasmus
Zeynep Tuncer

CONTENTS

Frankfurt University of Applied Sciences (FRA-UAS) and Institut for New Media (Frankfurt)(INM)
Council for Scientific and Industrial Research (CSIR) of South-Africa
Technical University Darmstadt (TUD)

Abstract

This text is based on the the paper ”AAI - Actor-Actor Interaction. A Philosophy of Science View” from 3.Oct.2017 and version 11 of the paper ”AAI - Actor-Actor Interaction. An Example Template” and it transforms these views in the new paradigm ’Actor- Actor Systems Engineering’ understood as a theory as well as a paradigm for and infinite set of applications. In analogy to the slogan ’Object-Oriented Software Engineering (OO SWE)’ one can understand the new acronym AASE as a systems engineering approach where the actor-actor interactions are the base concepts for the whole engineering process. Furthermore it is a clear intention to view the topic AASE explicitly from the point of view of a theory (as understood in Philosophy of Science) as well as from the point of view of possible applications (as understood in systems engineering). Thus the classical term of Human-Machine Interaction (AAI) or even the older Human-Computer Interaction (HCI) is now embedded within the new AASE approach. The same holds for the fuzzy discipline of Artificial Intelligence (AI) or the subset of AI called Machine Learning (ML). Although the AASE-approach is completely in its beginning one can already see how powerful this new conceptual framework is. [1]

1 History: From HCI to AAI

To speak of ’Actor-Actor Interaction (AAI)’ instead of ’Human-Computer Interaction (HCI)’ is rooted in the course of history. When the World War II ended several advances in technology and software gave raise to great expectations and visions what the future can bring mankind to improve life.[2]

Looking to the course of events between 1945 and about 2000 one can observe a steady development of the hardware and the software in many directions. This caused an explosion in many variants of new applications and usages of computer. This continuous challenge of how human persons can interact with this new technology provoked a rapid development what has been called in the beginning ’Human Computer Interaction (HCI)’. But with the extension of the applications in nearly all areas of daily live from workplace, factory, to education, health, arts and much more the interaction was no longer restricted to the ’traditional’ computer but interaction happened with all kinds of devices which internally or in the background used computer hardware and software. Thus a ’normal’ room, a ’normal’ street, a ’normal’ building, a toy, some furniture, cars, and much more turned into computerized devices with sensors and actuators. At the same time the collaborators of human persons were not only other human persons

1. This text has a long ’conceptual history’ leading back to the Philosophy-of-Science studies of Doeben-Henisch 1983 - 1989 in Munich under the guidance of Peter Hinst, many intensive discussions between Doeben-Henisch and Erasmus about Systems engineering since 1999, a paper written by Doeben-Henisch and Wagner 2007 [1] with ongoing discussions since then, a lecture by Doeben-Henisch about formal specification and verification in 2010 [2], two papers by Erasmus and Doeben Henisch in 2011 [3], [4], about 20 regular semesters with the topic Human-Machine Interaction by Doeben-Henisch at the Frankfurt University of Applied Sciences (Frankfurt, Germany)(unpublished) in the timespan 2005 - 2015, two regular semesters with the topic AAI together with Tuncer in SS2016 and WS2016 at the Frankfurt University of Applied Sciences (Frankfurt, Germany) (unpublished), and two workshops with Erasmus in summer 2016 and Spring 2017 (unpublished). Additionally many discussions between Doeben-Henisch and Idrissi about AI and AAI since 2015.

2. For some ’bits of history’ see Doeben-Henisch (2018) [5]

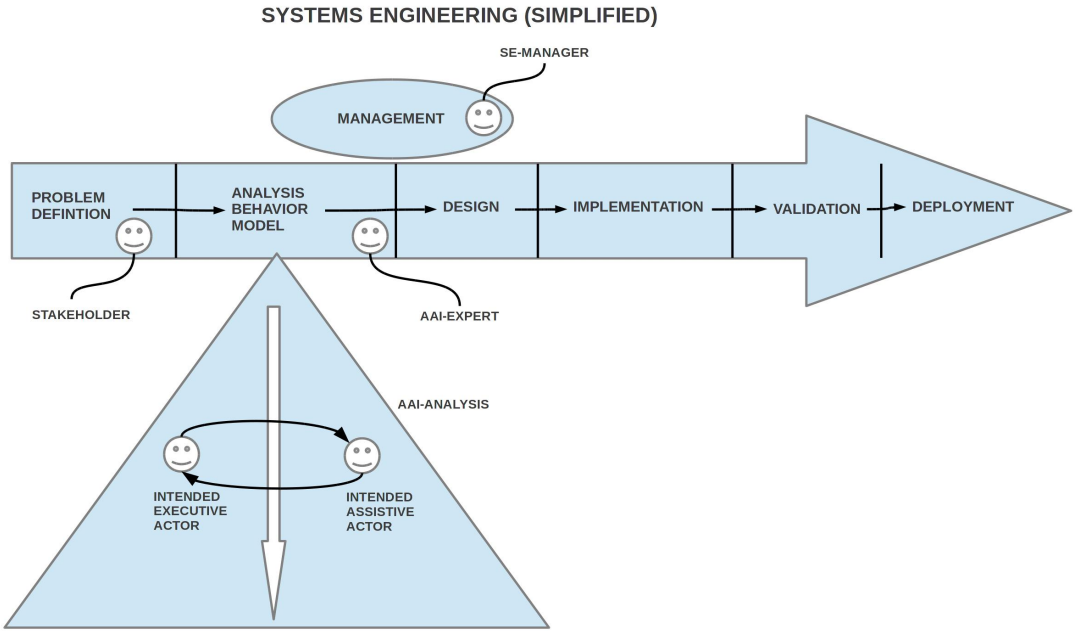

Fig. 1. Overview Systems Engineering Process - participating actors

or certain animals but more and more 'intelligent' machines, robots, smart interfaces. Thus to speak of a 'human user' interacting with a 'technical interface' was no longer appropriate. A more appropriate language game is the new talk of 'interacting actors', which can be sets of different groups of actors interacting in some environment to fulfill a task. Actors are then biological systems (man as well as animals) and non-biological systems.

This new perspective is guiding the following considerations.

2 Different Views

If one wants to deal with the development of optimal interfaces within certain tasks for executing actors[3] one can distinguish different *views* onto this problem (see figure 1).

The common *work view* in systems engineering is an *expert (EXP)* as part of a *systems engineering process (SEP)* who takes a *problem description* D_p and does some analysis work to find an *optimal solution candidate (OSC)*.

3. Today still mostly human persons.

One *level above* we have the *manager (MNG)* of the systems engineering process, who is setting the *framework* for the process and has to *monitor* its working.

Another upper level is the *philosopher of science (POS)* who is looking onto the managers, processes, and their environments and who delivers *theoretical models* to *describe* these processes, to *simulate* and to *evaluate* these.

In this text the *Actor-Actor Interaction (AAI)* is the main focus, embedded in a *Systems Engineering Process (SEP)*, all embedded in a minimal *Philosophy of Science (PoS)* point of view.

For this the following minimal SEP-structure is assumed[4]:

$$\begin{aligned} SEP(x) \quad & iff \quad x = \langle P, S, Sep \rangle \\ Sep \quad & : \quad P \longrightarrow S \\ Sep \quad & = \quad \alpha \otimes \delta \otimes \mu \otimes \upsilon \otimes o \\ \alpha \quad & := \quad Analysis\ of\ the\ problem\ P \\ \delta \quad & := \quad Logical\ design \\ \sigma \quad & := \quad Implementation\ of\ S \\ \upsilon \quad & := \quad Validation \\ o \quad & := \quad Deployment \end{aligned} \tag{1}$$

The outcome of the analysis of an AAI-expert is an *optimal solution candidate (OSC)* for an interface of an assisting actor embedded in a complete *behavior model* M_{SR} given as an *actor story (AS)* combined with possible *actor models (AMs)*. This output provides all informations needed for a following *logical design*. The logical design provides the blue-print for a possible *implementation* of a concrete working system whose behavior should be in agreement (checked through a *validation* phase) with the behavior model provided by the AAI-analysis.

3 Philosophy of the AAI-Expert

Before digging into the details of the following actor-actor interaction (AAI) analysis done by an AAI-expert one has to consider the conditions under which the AAI-expert is doing his job. These considerations are done in a separate paper called 'Philosophy of the AAI-Expert' (see Doeben-Henisch (2018) [6]).

The main topic in the philosophy paper is centered around the findings of modern biology and psychology that the ability of human persons to use a set theoretical language L_ϵ to talk about the experiences with the world is grounded in the cognitive machinery of human persons including complex processes related to perception, memory, spatial and temporal thinking, embedding of languages and others. Because the human brain in the body is not directly interacting with the outside world but mediated by sensors and actuators it is this complex cognitive machinery which constructs an inner model of the outside world. And it are exactly the properties of this 'inner model' which provide a 'point of reference' for all our thinking and talking.

One conclusion from these considerations is that reality is basically perceived as a stream of events, which can be divided in distinguishable situations, called states. A state is understood as a set of properties embedded in a three-dimensional space. If at least one property changes a state changes. Subsets of properties can be understood as objects, which in turn can be subdivided into 'actors' and 'non-actors'.

4. For the first paper of Erasmus together with Doeben-Henisch about this subject see [3]

Actors can 'sense' their environment and they can 'respond'. More distinctions are possible as needed.

To understand how an AAI-expert perceives his world, generates internal models, and how he is communicating with others, one has to clarify these philosophical groundings.

4 Problem (Document)

1) The *problem document* D_P is the result of a communication between some *stakeholder (SH)* and some *experts*, which have discussed a *problem* P which the stakeholder wants to be solved. In this context it suffices to describe shortly in the introduction of the problem document which persons have been participating in the communication with their communication addresses for further questions.

2) Due to the fuzziness of human communication one has to assume to a certain degree a *semantic gap* with regard to the participants of the communication which generated the problem document as well as for potential readers of the problem document.[5]

3) Additionally to the general problem a finite set of special *constraints (C)* can be given, which correspond to the traditional 'non-functional requirements'. To do this in the right way one has to describe the 'intended meaning' of these constraints in a way that it is possible either to decide, whether this intended meaning is fulfilled by the following actor story and actor models or that these constraints pointing to the follow up phases of the systems engineering process.

5 Check for Analysis

Within the general analysis phase of systems engineering the AAI-perspective constitutes a special view. This implies a check of the *occurrence* of the following aspects:

1) At least one *task (T)* and
2) an *environment (ENV)* for the task and
3) an *executive actor (ExecA)* as the intended user.

6 AAI-Analysis

The goal of the AAI-analysis is to find an optimal *assistive actor (AssA)*[6] to support the *executive Actor (ExecA)*[7] in his task. For this to achieve one needs an iterative application of the whole AAI-analysis process whose results are evaluated for an *optimal solution*.

To analyze the problem P one has to dig into the problem P so far that one is able to *tell a complete story*, how to *understand* and later to *realize* the task.

It can be some work to *investigate* the details of such a story. The investigation is complete if the resulting story is *sound*, that means all participants agree that they *understand* the story and that they *accept* it.

To *communicate a story* we assume the following main modes: *textual*, *pictorial*, *mathematical*, as well as *simulated*.[8]

.

5. For an early discussion of one of the authors about the semantic-gap problem see Doeben-Henisch & Wagner (2007) [1].
6. Traditionally understood as the technical interface.
7. Traditionally understood as the human user.
8. For an extended explanation of the formalisms used in this document see the web-page https://uffmm.org/2017/12/27/formal-appendix-for-the-aai-case-studies/

6.1 Actor Story (AS)

To *communicate* a story in the main modes textual, pictorial, mathematical as well as simulated one has to consider the above mentioned epistemological situation of the AAI-expert.

The *point of view* underlying the description of an actor story AS is the so-called 3rd-person view. This means that all participating objects and actors are described from their *outside*. If an actor *acts* and changes some property through it's action it is not possible in a 3rd-person view to describe the inner states and inner processes, that enabled the actor to act and why he acts in this way. To overcome the limits of a 3rd-person view one has to construct additional models called *Actor Models (AMs)*. For more details have a look to the section 6.2.1.

The relationship between the traditional '*functional requirements (FR)*' and the 'actor story' is such, that all necessary functional requirements have to be part of the actor story. The '*non-functional requirements (NFR)*' have to be defined in their intended meaning before the actor story and then it must be shown, how the structure of the actor story 'satisfies' these criteria. In this sense are the 'non-functional requirements' presented as 'constraints' which have the status of 'meta-predicates', which have to be designed in an appropriate 'control logic' for actor stories.[9]

The *Philosophy behind the Actor-Story concept* – as pointed out in the figure 2 – is given in a draft paper by Doeben-Henisch (2018) [6] describing the basic relationships between the empirical external world with the body as a part and the internal, mental, structures and processes enabling things like concepts, memories, languages with meaning etc.

From this one can derive that different modes to represent empirical matters with symbolic expressions like a language L have as primary point of reference the '*mental ontology*' DAT_{ontol} of the AAI experts. While the mental ontology is assumed to be 'the same' for all different modes of symbolic articulation[10], the different modes of articulation can express different aspects of the same mental ontology more highlighted than in other modes of symbolic articulation.

In the case of expressions of some '*everyday language*' L_0 like German or English we have only symbols of some alphabet, concatenated to strings of symbols or articulated as a stream of sounds. Thus an understanding of the intended meaning is completely bound to the mental encoding of these expressions, eventually associated with some other clues by body-expressions, mimics, special contexts, and the like.

If we would use a '*pictorial language*' L_{pict} as in a comic strip, we would have again some strings of symbols but mostly we would have sequences of two-dimensional drawings with the symbols embedded. These drawings can be very similar to th perceptual experience of spaces, objects, spatial relations, timely successes, and more properties which somehow 'directly' encode real situations. Thus the de-coding of the symbol expressions is associated with a strong 'interpretation' of the intended situations by 'world-like pictures'. In this sense one could use such a pictorial language as a 'second hand ontology' for the encoding of symbolic expressions into their intended meaning.

9. This topic of 'Non-Functional Requirements (NFRs)' as well as 'Functional Requirements (FRs)' and their relationship is a hot topic in systems engineering and has not yet a complete solution. The general problem is how to 'represent' the NFRs in a way, that these can be handled in the overall system. The following selected papers (only a subset of thematic related papers) can illustrate the discussion: dealing mainly with NFRs see Khalique et.al. (2017) [7], Fellir et.al. (2015) [8], Mairiza et.al. (2013) [9], Suhr et.al. (2013) [10], Yin et.al. (2013) [11], Zhang et.al.(2013) [12], Menzel et.al. (2010) [13], Liu et.al. (2012) [14], Kassab et.al. (2009) [15]. Dealing mainly with FRs see Lian et.al. (2017) [16], Abrahão et.al (2013) [17]. The big advantage of the AASE paradigm in this context is that the mathematical version of the actor story provides a formal structure which allows to describe all functional requirements (FRs) in a formal way which allows the annotation of non-functional requirements (NFRs) easily.

10. Which is a highly idealistic assumption in case of learning systems

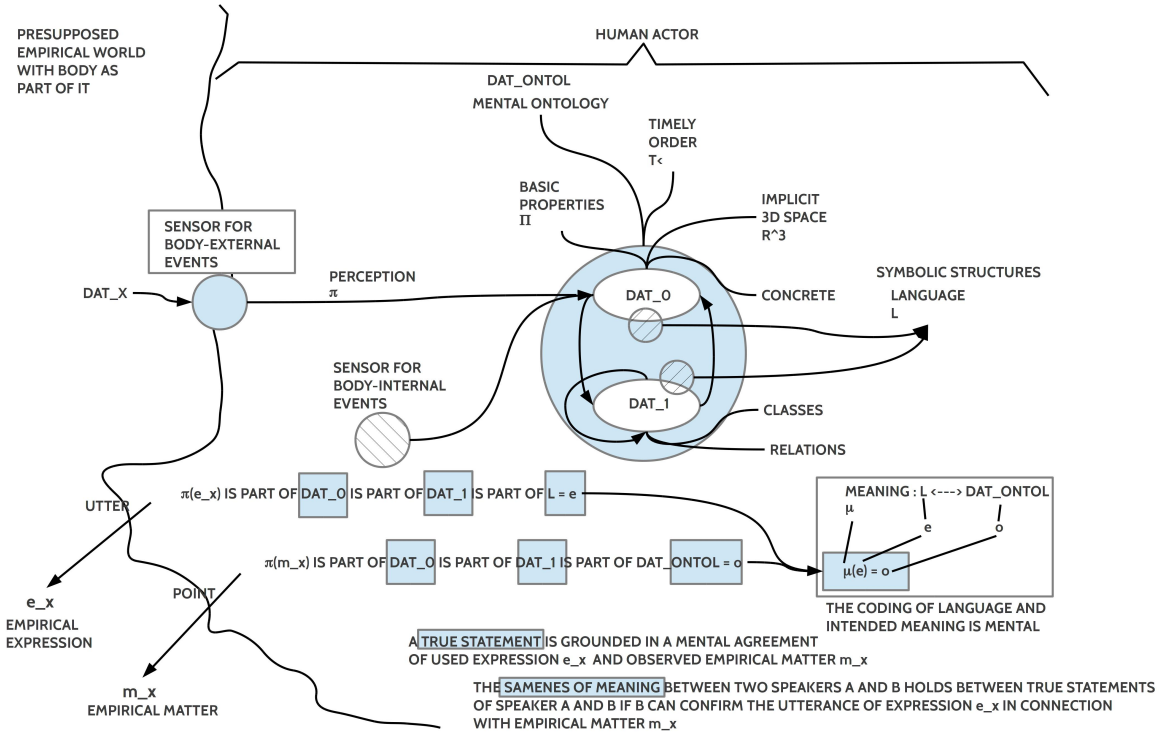

Fig. 2. Basic Mappings between empirical reality with body as a part and internal mental structures

But for the intended engineering of the results of an AAI analysis neither the everyday language mode L_0 nor the pictorial language mode L_{pict} is sufficient. What is needed is a '*formal language*' L_ϵ which can easily be used for logical proofs, for automated computations, as well as for computer simulations. One good candidate for such a formal language is a language using mathematical graphs which are additional enriched with formal expressions for properties and changes between states. This allows an automatic conversion into automata which can simulate all these processes. Additional one can apply automatic verification for selected properties, e.g. for non-functional requirements!

From this we derive the following main modes of an actor story in this text: (i) Everyday language L_0(here English), (ii) Pictorial language L_{pict} (in this version of the text not yet defined), (iii) Formal langauge L_ϵ, (iv) Converted automata α_{L_ϵ} out of the formal language, which can simulate the actor story.

The additional actor models described after the actor story can be seen as special extensions of the actor story and have to be included in the simulation mode. This is straightforward but has also not yet been included in this version of the text.

6.1.1 Textual Actor Story (TAS)

An actor story AS in the *textual mode* is a *text* composed by expressions of some *everyday language* L_0 – default here is English L_{EN} –. This text describes as his *content* a sequence of distinguishable *states*. Each state s – but not an *end-state* – is connected to at least one other follow-up state s' caused by the *change* of at least one *property* p which in the follow up state s' either is *deleted* or has been *newly created*.

Every described state s is a set of properties which can be sub-distinguished as *objects (OBJ)* which are occurring in some *environment (ENV)*. A special kind of objects are *actors (As)*. Actors are assumed to be able to *sense* properties of other actors as well as of the environment. Actors are also assumed to be able to *respond* to the environment without or with taking into account what *happened before*.

Actors are further sub-divided into *executive* actors as well as *assistive* actors. Assistive actors A_{assist} are those who are expected to support the executive actors A_{exec} in fulfilling some *task (t)* (with $t \in T$).

A *task* is assumed to be a sequence of states with a *start* state s_{start} and a *goal* state s_{goal}, where the goal-state is an end state. The set of states connecting the start and the goal state is finite and constitutes a *path* $p \in P$. There can be more than one path leading from the start state to the goal state. The states between the start and the goal state are called *intermediate* states.

Every finished actor story has a least one path.[11]

6.1.2 Pictorial Actor Story (PAT)

In case of an textual actor story (TAS) – as before explained – one has a set of expressions of some common language L_0. These expressions *encode a possible meaning* which is rooted in the *inner states (IS)* of the participating experts. Only the communicating experts *know* which meaning is encoded by the expressions.

This situation – labeled as semantic gap – can cause lots of misunderstandings and thereby *errors* and *faults*.

To minimize such kinds of misunderstandings it is a possible strategy to *map* these intended meanings in a *pictorial language* L_{pict} which has sufficient resemblances with the intended meaning. Replacing the textual mode by a story written with a pictorial language L_{pict} can *show* parts of the encoded meaning more directly.

As one can read in the section 3 'Philosophy of the View-Point' the world of objects for a *standard user* is mapped into a spatial structure filled with properties, objects, actors and changes. This structure gives a blue-print for the *structure of the possible meaning* in an observer looking to the world with a 3rd-person view. Therefore a pictorial language can *substitute* the intended meaning to some degree if the pictorial language provides real pictures which are structurally sufficient similar to the perceived visual structure of the observer.

To construct a *pictorial actor story (PAS)* one needs therefore a mapping of the 'content' of the textual actor story into an n-dimensional space embedded in a time line. Every time-depended space is filled with objects. The objects show relations within the space and to each other. Objects in space, the space itself, and the changes in time are based on distinguishable properties. To conserve a consistency between the textual and the pictorial mode one needs a *mapping* between these both languages: $\pi : L_0 \longleftrightarrow L_{pict}$.

6.1.3 Mathematical Actor Story (MAS)

The Graph as Backbone

To translate a story with *spatial structures* and *timely changes* into a mathematical structure one can use a *mathematical graph* γ extended with *properties* Π and *changes* Ξ for encoding.

11. To turn a textual actor story into an *audio* actor story (AAS) one can feed the text into a *speech-synthesis* program which delivers spoken text as output.

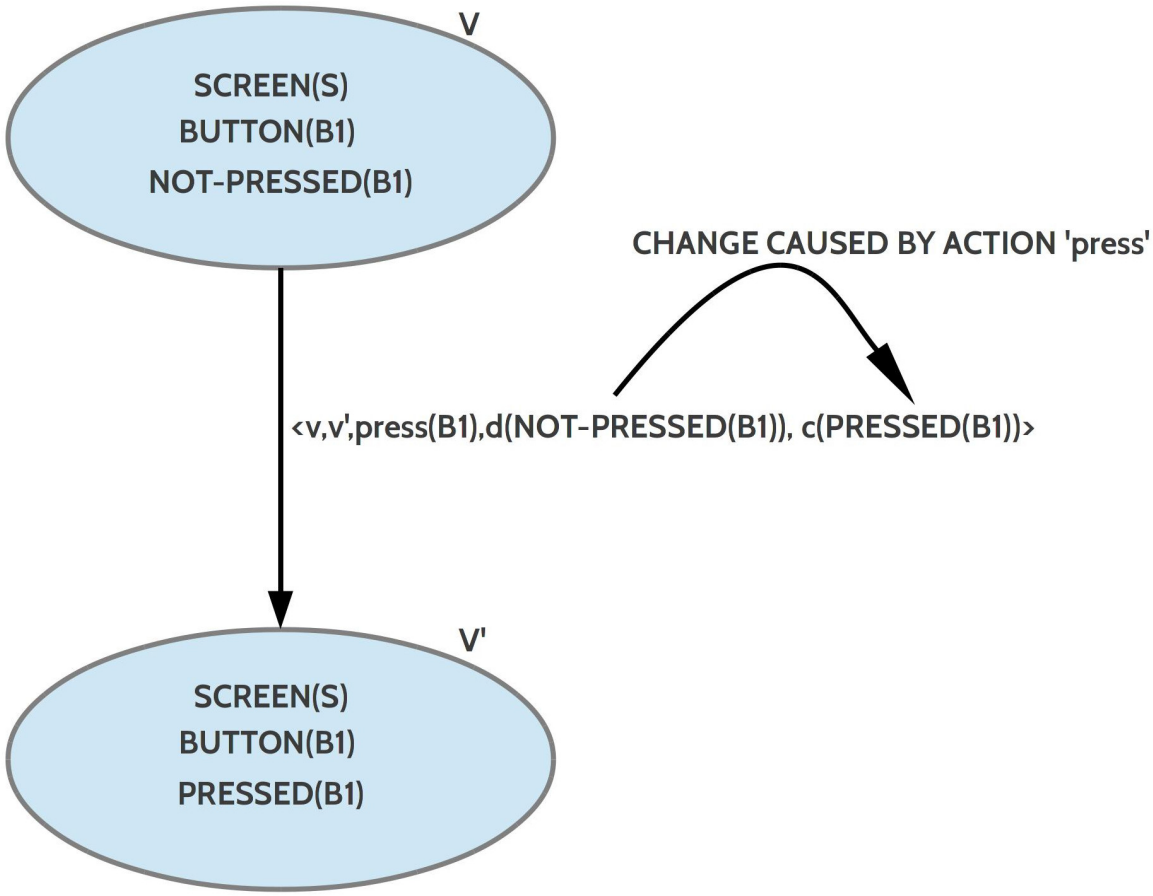

Fig. 3. Change event between two states

A *situation* or *state* $q \in Q$ given as a spatial structure corresponds in a graph γ to a *vertex* (also called 'node') v, and a *change* $\xi \in \Xi$ corresponds to a pair of vertices (v, v') (also called an 'edge' $e \in E$).

If one maps every vertex $v \in V$ into a set of property-expressions $\pi \in 2^{L_\Pi}$ with $\lambda : V \longmapsto 2^{L_\Pi}$ and every edge $e \in E$ into a set of change-expressions L_Ξ with $\epsilon : E \longmapsto 2^{L_\Xi}$ then a vertex in the graph γ with the associated property-expressions can represent a state with all its properties and an edge e followed by another vertex v' labeled with a change-expression can represent a change from one state to its follow-up state.

A graph γ extended with properties and changes is called an *extended graph* γ^+.

Thus we have the extended graph γ^+ given as:

$$\gamma^+(g) \quad iff \quad g = \langle V, E, L_\Pi, L_\chi, \lambda, \epsilon \rangle \tag{2}$$
$$E \quad \subseteq \quad V \times V \tag{3}$$
$$\lambda \quad : \quad V \longrightarrow 2^{L_\Pi} \tag{4}$$
$$\epsilon \quad : \quad E \longrightarrow 2^{L_\Xi} \tag{5}$$

The occurrence of a *change* is represented by two vertices v, v' connected by an edge e as $e : \{v\} \longmapsto \{v'\}$. The follow-up vertex v' has at least one property-expression less as the vertex v or at least one property-expression more. This change will be represented in a formal *change-expression*

$\epsilon \in L_\chi$ containing a list of properties to be *deleted* as $d : \{p_1, p_3, ...\}$ and properties to be newly *created* as $c : \{p_2, p_4, ...\}$.

The *deletion-operation* is shorthand for a mapping of subtracting property-expressions like $d : \{s\} \longmapsto s - \{p_1, p_3, ...\}$ and the *creation-operation* is shorthand for a mapping of adding property-expressions like $c : \{s\} \longmapsto s \cup \{p_2, p_4, ...\}$. Both operations are processed in a certain order: first deletion and then addition, $change = d \otimes c$.

Objects and Actors

Every assumed *object* $o \in OBJ$ attached to a vertex represents a sub-set of the associated properties. An *actor* $a \in A$ is a special kind of object by $A \subseteq OBJ$.

Enriched by Properties

1) Generally it is assumed that there exists some 'domain of reference' D^R which corresponds to a situation/ state of an actor story.

2) For every 'object' in D^R one can introduce a 'name' realized as a string of 'small alphanumeric letters' beginning with a 'capital letter'. Names are a subset of terms. Examples: *'Hobbes', 'U2', 'Moon', ...*

3) Mappings from distinct objects into other distinct objects which have all to be objects of D^R are called 'functions' realized as a string of 'small alphanumeric letters' followed by n-many terms enclosed in brackets. Functions are as well a subset of terms. Examples: *'add(3,4)', 'push(Button1)', ...'*

4) 'Properties' Π are relations between objects in an assumed 'domain of reference' D^R. The properties are symbolically represented by property expressions L_Π which are realized by n-many terms functioning as 'arguments' of n-ary 'predicates'. Thus a property-expression is a sequence of an n-ary 'predicate' as a string of 'big alphanumeric letters' enriched with the '-'-sign followed by n-many terms as arguments enclosed in brackets. Example: 'USER(U1)', 'SCREEN(S)', BUTTON(B1)', 'IS-PART-OF(B1,S)', 'ON(push(B1))', ...

5) As stated above there exists a mapping from states into sets of property expressions written as $\lambda : V \longrightarrow 2^{L_\Pi}$

Enriched by Changes

1) A change in the domain D^R happens when at least one property disappears or emerges. To express this symbolically one has to assume (as stated above) that there are two formal states v, v' each with property expressions $L^v_\Pi, L^{v'}_\Pi$ and the property expressions from follow-up state v' are generated by applying a 'change-action' realized as a function $\alpha \in ACT$ to the preceding state v. The change action has a 'name' realized by a string of 'small alphanumeric values' followed by a 'delete function' named 'delete' (or short 'd') and then by a 'creation function' named 'create' (or short 'c'). Thus the change action α is a concatenated operation $\alpha = d() \otimes c()$. The arguments of the delete- and create-function are property expressions.

2) Example: if there is a set of property expressions $L^v_\Pi = \{SCREEN(S), BUTTON(B1), NOT - PRESSED(B1)\}$ and a change action $\alpha(L^v_\Pi)$ with the sub-functions $d(NOT - PRESSED(B1))$ and $c(PRESSED(B1))$ then the resulting follow-up property set looks like $L^{v'}_\Pi =$

$\{SCREEN(S), BUTTON(B1), PRESSED(B1)\}$

3) The complete change expression will be realized as a 'list': $\langle v, v', \alpha, d(p_1, ..., p_n), c(p_1, ..., p_m)\rangle$. This reads: a change action with name α has been applied to state v and generates a new state v' by (i) copying the properties from state v to state v', then (ii) deletes the properties $(p_1, ..., p_n)$ in v', and then (iii) creates the properties $(p_1, ..., p_m)$ in v'. The result of applying (i) - (iii) to the old state v generates the new state v'.

4) Thus change statements are terms derived as a subset as follows: $\epsilon \subseteq V \times V \times ACT \times \Pi^{Nat} \times \Pi^{Nat}$ (with Nat as the natural numbers including 0).[12]

5) If there is in one state v more than one action possible than more than one change statement is possible. This results in more than one edge leading from state v to n-many follow-up states $v'_1, ..., v'_n$.

6) Additional to the names of possible objects we assume a special operator *'not(n)'* applied to a name 'n'. The meaning of the operator is, that in this case not the name 'n' is valid, but the 'absence' of the object signified by the name n'. This is important because otherwise in case of many alternative options one has to enumerate all alternatives to an object named 'n'.

Correspondence between mathematical and pictorial modes

To keep the consistency between a mathematical and a pictorial actor story one needs a mapping from the pictorial actor story into the mathematical actor story and vice versa, $m_{p.m} : L_{pict} \longleftrightarrow L_{math}$.

6.1.4 Simulated Actor Story (SAS)

A *simulated actor story (SAS)* corresponds to a given *extended graph* γ^+ by mapping the extended graph into an *extended automaton* α^+.

The usual definition of a finite automaton is as follows: $\langle Q, I, F, \Sigma, \Delta\rangle$ with

1) Q as a finite set of *states*
2) I $\subseteq$ Q as the set of *initial states*
3) F $\subseteq$ Q as the set of *final states*
4) Σ as a finite input *alphabet*
5) $\Delta \subseteq$ Q $\times$ Σ^* $\times$ Q as the set of *transitions*

If one *replaces/ substitutes* the *states* by *vertices*, the *input expressions* by *change-expressions* and the *transitions* by *edges* then one gets: $\langle V, I, F, L_\chi, E\rangle$ with

1) V as a finite set of *states*
2) I $\subseteq$ V as the set of *initial states*
3) F $\subseteq$ V as the set of *final states*
4) L_χ as a finite set of *input expressions*
5) $E \subseteq V \times L_\chi \times V$ as the set of *transitions*

Finally one extends the structure of the automaton by the set of property-expressions L_Π as follows: $\langle V, I, F, L_\chi, L_\Pi, E, \lambda\rangle$ with $\lambda : V \longrightarrow 2^{L_\Pi}$.

12. The default assumption is that either the delete or the create function has to have at least one property argument.

With this definition one has an extended automaton α^+ as an automaton who being in state v *recognizes* a change-expression $\epsilon \in L_\chi$ and generates as follow-up state v' that state, which is constructed out of state v by the encoded deletions and/ or creations of properties given as property-expressions from L_Π. All state-transitions of the automaton α^+ from a start-state to a goal-state are called a *run* ρ of the automaton. The set of all possible runs of the automaton is called the *execution graph* γ_{exec} of the automaton α^+ or $\gamma_{exec}(\alpha^+)$.

Thus the *simulation* of an actor story corresponds to a certain run ρ of that automaton α^+ which can be generated out of a mathematical actor story by simple replacement of the variables in the graph γ^+.

6.1.5 Task Induced Actor Requirements (TAR)

Working out an actor story in the before mentioned different modes gives an outline of *when* and *what* participating actors *should do* in order to *realize* a *planned task*.

But there is a difference in saying *what* an actor *should do* and in stating *which kinds of properties* an actor *needs* to be able to show this required behavior. The set of required properties of an actor is called here the *required profile* of the actor A $RProf_A$. Because the required profile is depending from the required task, the required profile is not a fixed value.

In the general case there are at least two different kinds of actors: (i) the *executing* actor A_{exec} and (ii) the *assistive* actor A_{assis}. In this text we limit the analysis to the case where executing actors are *humans* and assistive actors *machines*.

6.1.6 Actor Induced Actor Requirements (UAR)

Because the required profile $RProf_{requ}$ of an executive actor realizing a task described in an actor story can be of a great variety one has always to examine whether the *available executing actor* A_{exec} with its *available profile* $RProf_{avail}$ is either in a *sufficient agreement* with the required profile or not, $\sigma : RProf_{requ} \times RProf_{avail} \longmapsto [0, 1]$.

If there is a *significant dis-similarity* between the required and the available profile then one has to *improve* the available executive actor to approach the required profile in a finite amount of time $\chi : A_{avail,exec} \times RProf_{requ} \longmapsto A_{requ,exec}$. If such an improvement is not possible then the planned task cannot be realized with the available executing actors.

6.1.7 Interface-Requirements and Interface-Design

If the available executing actors have an available profile which is in sufficient agreement with the required profile then one has to *analyze the interaction* between the executing and the assistive actor in more detail.

Logically the assistive actor shall assist the executing actor in realizing the required task as good as possible.

From this follows that the executing actor has to be able to *perceive* all necessary properties in a given situation, has to *process* these perceptions, and has to *react* appropriately.

If one calls the sum of all possible perceptions and reactions the *interface of the executing actor* $Intf_{A,exec}$ and similarly the sum of all possible perceptions and reactions of the assistive actor the *interface of the assistive actor* $Intf_{A,assis}$,then the interface of the assistive actor should be optimized with regard to the executing actor.

To be able to know more clearly how the interface of the assistive actor $Intf_{assis}$ should look like that the executive actor can optimally perceive and react to the assistive interface one has to have sufficient knowledge about how the executive actor *internally processes* its perceptions and computes its reactions. This knowledge is not provided by the actor story but calls for an additional model called *actor model*.

6.2 Actor

A main element within the AAI-perspective is the intended executive actor, often called *user U*. In this AAI-version of HMI the assistive actor can also be seen as a user. The processing of the task requires that the user U can *perceive* all necessary aspects of the task processing as well as he can *act* as needed. Besides this one expects that the user U is able to *process* the *perceptual information* – here called *input I* – in a way that the user is able to *generate* the right *actions* – here called *output O* –. One can break down the *required behavior* to a series of necessary inputs I for the user followed by necessary responses O of the user. This results in a series of input-output pairs pairs $\{(i,o),\cdots,(i,o)\}$ defining implicitly a required empirical behavior function ϕ_e = $\{(i,o),\cdots,(i,o)\}$. Because any such empirical behavior function is finite and based on single, individual events, it is difficult to use this empirical finite function as the function of an explicit model. What one needs is an explicit general theoretical behavior function like:

$$\phi \quad : \quad I \longmapsto O \tag{6}$$

Although an empirical behavior function ϕ_e is not a full behavior function, one can use such an empirical function as a *heuristic guide* to construct a more general theoretical function as part of a complete hypothetical model of the user.

6.2.1 Actor and Actor Story

While one can describe in an actor story (AS) possible changes seen from a 3rd-person view one can not describe *why* such changes happen. To overcome these limits one has to construct additional models which describe the internal states of an actor which can explain why a certain behavior occurs.

The general idea of this interaction between actor story and actor model can be seen in figure 4.

1) In a simple actor story with only two states v, v' we have an actor called 'USER(U1)' which has 'visual perception' and which can act with 'motor activities'.

2) Therefore the actor can 'see' properties like 'SCREEN', 'BUTTON', and 'NOT-PRESSED'. Based on its 'behavior function' Φ the actor can compute a possible output as a motor-action, described as an event expression $\langle v, v', press(BUTTON(B1)), d(not-pressed(B1)), C:(pressed(B1))\rangle$.

3) This results in a change leading to v'. The actor $U1$ is left out in v', also it is still part of v'.

6.2.2 Actor Model

It is an interesting task, to *elaborate* a *hypothetical model* of the internal processes of an user which defines the*theoretical behavior function* ϕ. To do this broadly with all details is beyond the scope of this text. Instead we will work out a first basic model which can be understood as a kind of a *theoretical template* which can be further extended in the future.

The task of modeling a possible user U is twofold: first (i) one has to define a complete formal model of a possible structure and it's dynamic, second (ii) it must be possible to predict the behavior of the model in a way that it is possible to *observe* and *measure* this behavior. If the observable behavior of the model is

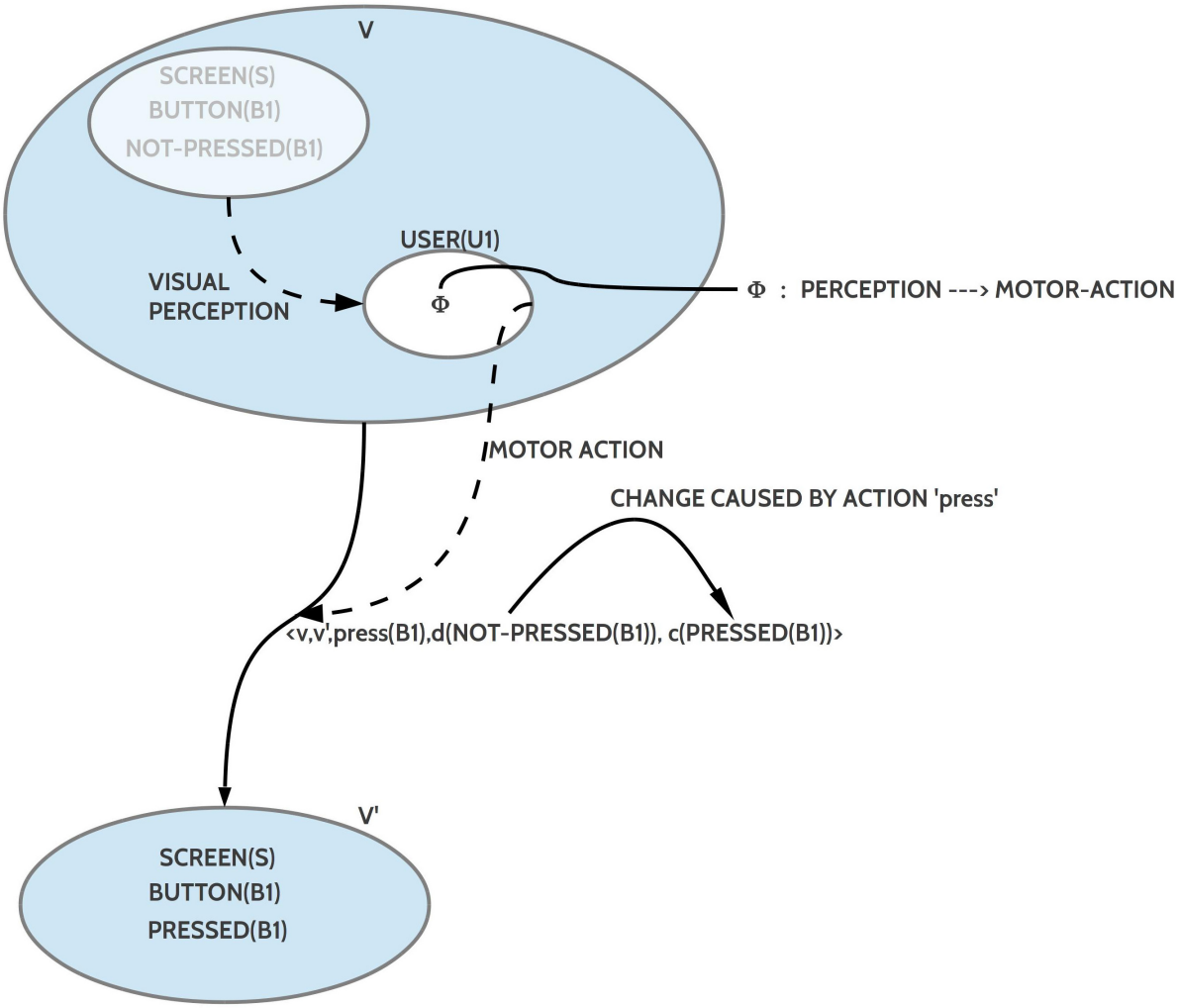

Fig. 4. Change event with an embedded actor

including the *empirical behavior function* ϕ_e, then the hypothetical model is *empirical sound in a weak sense*.

$$\phi_e \quad \subseteq \quad \phi \tag{7}$$

We understand here a *model* as a mere collection of rules, while an *algebraic structure* is an extension to models by including additional sets as well as axioms. But we use the term 'model' here equivalently to the term 'algebraic structure'.

6.2.3 Actor as Input-Output System

To enable a *transparent interaction* between actor and environment it will be assumed that an actor is generally an *input-output system (IOSYS)*, that means that an actor has (i) *inputs (I)* from the environment in a state, which are translated by some kind of a 'perceptional system' generating *perceptions* of this environment as well as (ii) *outputs (O)* which can cause changes in the environment. The sum of all inputs I and outputs O defines the *basic interface (BIntf)* of an input-output system S in an environment E, written $BIntf_{S,E} = \{x|x \in I \times O\}$. We can write this as follows:

Def: Input-Output System (IOSYS)

$$IOSYS(S) \quad iff \quad S = \langle I, O \rangle \tag{8}$$
$$I \quad := \quad Input$$
$$O \quad := \quad Output$$
$$\tag{9}$$

and:

Def: Basic Interface (BIntf)

$$E \quad := \quad Environment\ of\ system\ S \tag{10}$$
$$IOSYS(S) \quad iff \quad S = \langle I, O \rangle \tag{11}$$
$$I \quad := \quad Input \subseteq E$$
$$O \quad := \quad Output \subseteq E$$
$$BIntf_{S,E} \quad = \quad \{x | x \in I \times O\}$$

Def: Real Interface (RIntf)

The *basic interface (BInf)* has to be distinguished from that interface which represents a 'real' device interacting with an executive actor. The *real interface (RIntf)* of an assistive actor 'realizes' the 'basic interface' by providing some sensoric appearance of an assistive actor. Thus if the executive actor needs an *input* from the interface there can be visual or acoustic or haptic or other sensoric properties which are used to convey the input to the executive actor. As well, if the executive actor wants to produce an output to change some properties in the assistive actor there must be some sensor at the side of the assistive actor which can receive some 'action' from the executive actor. The concrete outlook of such a real interface is the task of the '*interface design*' given a 'basic interface'.

6.2.4 Learning Input-Output Systems

A '*learning input-output system (LIOSYS)*' is a special case of an input-output system because its 'basic interface can change'! This dynamic behavior is described by a '*learning behavior function*' ϕ. The learning behavior function assumes additionally to the inputs (I) and outputs (O) a non-empty set of 'internal states (IS)' which can change their properties. These dynamic internal states can 'represent' some properties of the inputs in a way, that the system can take these properties into account for the 'computation of the next output'. Depending on the 'quality' of these internal representations the system can pre-view 'possible states' of the environment and its own situation in it. This structure of internal states as part of a behavior function is usually called 'memory' μ. A most general definition of learning input-output systems can be given as follows:

Def: Learning Input-Output System (LIOSYS)

$$LIOSYS(S) \quad iff \quad S = \langle I, O, IS, \phi \rangle \tag{12}$$
$$I \quad := \quad Input$$
$$O \quad := \quad Output$$
$$IS \quad := \quad Internal\ states$$
$$\phi \quad : \quad I \times 2^{IS} \longmapsto 2^{IS} \times 2^{O}$$

From this it follows that the 'basic interface' is only a subset of the behavior function of a learning system. This means to 'understand a learning input-output system' it does not suffice to describe the behavior of a system once but 'very often', because a learning system can generally learn always. Thus to

'predict' the behavior of learning systems in an environment is be far not trivial.

The change of behavior which is a property of learning systems is usually driven by 'random' as well as 'non-random' properties. Thus to describe the 'dynamics' resulting from a learning behavior one needs a sufficient knowledge about these '*change-inducing properties*'. In real biological systems these change-inducing properties seem to be composed of bundles of different functions interacting in a non-trivial way with each other. Until today a complete and transparent description of these change-inducing properties seems by far to be a too complex task to do.

6.2.5 General AM

A first simple actor model (AM) for learning actors is given as follows:

$$AM(u) \quad iff \quad u = \langle I, O, IS, \phi \rangle \tag{13}$$
$$I \quad := \quad Input \tag{14}$$
$$O \quad := \quad Output \tag{15}$$
$$IS \quad := \quad Internal\ states \tag{16}$$
$$\phi \quad : \quad I \times 2^{IS} \longmapsto 2^{IS} \times O \tag{17}$$

The term IS represents some internal states and the behavior function is using the internal states as secondary input as well as secondary output. This means that the behavior function can *modify* the internal states, which implies some capability for *learning*.

6.2.6 Sound Functions

To check whether the behavior of an actor is '*sound*' does mean here that the 'observable behavior' (within a test or in the real working case) is in '*agreement*' with the actor story.

Thus, given an actor story with some input property $i \in I$ in a state S and having an actor model with the behavior function ϕ presupposing some internal states IS, then the behavior function ϕ of the actor model should generate an output $\phi(i) = o$ with $o \in O$ in accordance with the possible outcomes described in the actor story for this state S.

In a standard '*non-learning*' case this can be realized by a behavior function ϕ which is completely '*deterministic*'. In this case the basic interface is identical to the (deterministic) behavior function, $BIntf = \phi$.

If one assumes a '*learning actor*' then the actor story describes the '*expected behavior*' as a set of 'input-output pairs' for every state and an actor has to be 'trained to learn' the 'expected sets of input-output pairs'.

That structure in a learning actor, which enables the 'storage' of important perceptions as well as the 're-use' of these stored perceptions, is here called a '*memory structure*' or shortly a '*memory*'.

One can use therefor a 'task' as a sequence of expected states defining sequences of input-output tasks to 'measure' the '*intelligence*' of an actor. Running a task the first time one can use the percentage of correctly solved sub-tasks within a certain amount of time as a 'benchmark' indicating some measure of 'intelligence'.[13]

13. For an introduction into the topic of psychological intelligence measures see e.g. Eysenk (2004) [18], Rost (2009) [19], Rost (2013) [20]

To measure the '*learning capacity*' of an actor one can use a task to explore (i) how much time an actor needs to find a goal state and (ii) how many repetitions the actor will need until the error rate has reached some defined minimum.[14] Another measure could be the quality of the memory by first identifying a maximum of correctness and then (iii) one measures the duration until which the maximum correctness of the memory has again weakened below a certain threshold of accuracy.[15]

While some minimal amount of '*learning time*' is needed by all kinds of systems – biological as well as non-biological ones – only the non-biological systems can increase the time span for '*not-forgetting*' much, much wider than biological systems.

Today the biggest amount of executing actors are still biological systems represented by human persons (classified as 'homo sapiens'), therefore parameters as 'learning time', 'memory correctness', or 'memory forgetting time' are important to characterize the 'difficulty' of a task and ways to explore possible settings which make the task difficult. From such a 'learning analysis' one can eventually derive some ideas for possible 'improvements'. From this follows that the format of usability tests should be adapted to these newly identified behavior based properties.

On account of the unobservability of the *inner states (IS)* of every real system it follows that all assumptions about possible inner states as well as about the details of the behavior function ϕ represent nothing else as a *hypothesis* which is given in the format of a *formal model*. The formal space for such hypothetical models is *infinite*.

6.2.7 Special AM

According to Card et al.(1983) [25] one has to assume at least three sub-functions within the general behavior function:

$$\phi = \phi_{perc} \otimes \phi_{cogn} \otimes \phi_{mot} \tag{18}$$
$$\phi_{perc} := Perception \tag{19}$$
$$\phi_{perc} : I \longmapsto (VB \cup AB) \tag{20}$$
$$VB := Visual\ buffer \tag{21}$$
$$AB := Auditory\ buffer \tag{22}$$
$$\phi_{cogn1} : (VB \cup AB) \times M_{STM} \longrightarrow M_{STM} \tag{23}$$
$$\phi_{cogn2} : M_{STM} \times M_{LTM} \longrightarrow M_{STM} \times M_{LTM} \tag{24}$$
$$\phi_{cogn1+2} := Cognition \tag{25}$$
$$\phi_{mot} : M_{LTM} \longrightarrow O \tag{26}$$
$$\phi_{mot} := Motor\ activity \tag{27}$$

Thus an input – visual or auditory – will be processed by the *perception function* ϕ_{perc} into an appropriate *sensory buffer* VB oder AB. The contents of the sensory buffers will then be processed by the partial *cognitive function* $cogn_1$ into the *short term memory (STM)*, which at the same time can give some input for this processing. Another cognitive function $cogn_2$ can map the contents of the short term memory into the *long term memory (LTM)* thereby using information of the long term memory as input too. From the long term memory the *motor function* can receive information to process some output O.

According to these assumptions we have to assume the following partitions:

14. The history of behavioral Psychology provides many examples for such experiments, see e.g. Hilgard et.al. (1979) [21] and a famous experiment with Tolman (1948) [22] using learning curves and error rates

15. The first scientist who did this in a pioneering work was the German Psychologist Ebbinghaus (1848) [23], English translation [24]

$$VB \cup AB \cup M_{STM} \cup M_{LTM} \quad \subseteq \quad IS \tag{28}$$

6.2.8 Hypothetical Model of a User - The GOMS Paradigm

The model thus far describes an overall structure with a general behavior which does not yet allow some concrete predictions. To determine a more concrete model several strategies are possible.

One old and popular strategy is labeled *GOMS* for *Goals, Methods, Operators* and *Selection rules*[16].

- **GOAL:** A *goal* is something to be achieved and will be represented by some *language expression*.
- **OPERATOR:** An *operator* is some *concrete action* which can be done.
- **METHOD:** A *method* is a composition of a goal and some operators following the goal to realize it.
- **SELECTION RULE:** A selection rule has an IF-THEN-ELSE structure: *IF* a certain condition is fulfilled, *THEN* some method will be selected, otherwise the method following the *ELSE* marker will be selected.

In case of the GOMS-strategy the cognitive function ϕ_{cogn} would be described by a finite set of methods which are organized around some goals.

6.2.9 Example: An Electronically Locked Door

For the following demonstration we use the simple example of an electronically locked door.[17]

With regard to this example we can construct an actor story as follows:

ACTORSTORY1 =

If we start with state Q1, then it will be followed by state Q2 if the output of the user is pushing the key with symbol *A*; otherwise, if the output is different, then we will will keep state Q1. Similar in the following states: If we are in state Q2 and the output of the user is pushing the key with symbol *B*, then the user story switches to state Q3; otherwise we are back in state Q1. Finally, if we are in state Q2 and the user pushes the key with symbol *A*, then we will reach the final state *Q4*, otherwise back again to state Q1.

The details of the different states are given here[18]:

1) Start = U1 $\cup$ S1 $\cup$ Env1
   U1 = $\{$USER(U1)$\}$
   S1 = $\{$SYSTEMINTF(S1),KEYPAD(K1), PART-OF(K1, S1), KEY(Ka), KEY(Kb),KEY(Kc),PART-OF($\langle Ka, Kb, Kc\rangle$, K1)$\}$
   Env1 = $\{$DOOR(D1), CLOSED(D1)$\}$
   Meaning: 'U1' is the name of a user, 'S1' the name of a system-interface, and 'Env1' is the name of an environment. All three 'U1, S1, C1' are names for subsets of properties of state Start.
2) CHANGE-AS:$\langle$ Start,Start,push(not(Ka),K1),d(),c()$\rangle$, $\langle$ Start,Q2,push(Ka,K1), d(), c(PRESSED(Ka))$\rangle$,
3) Q2 = U1 $\cup$ S1 $\cup$ Env1
   U1 = $\{$USER(U1)$\}$
   S1 = $\{$SYSTEMINTF(S1),KEYPAD(K1), PART-OF(K1, S1), KEY(Ka), KEY(Kb),KEY(Kc),PART-OF($\langle Ka, Kb, Kc\rangle$, K1),PRESSED(Ka)$\}$
   Env1 = $\{$DOOR(D1), CLOSED(D1)$\}$

16. A first extensive usage of a GOMS model can be found in Card et al. (1983) [25]:139ff
17. For a description of the example see: http://www.doeben-henisch.de/fh/fsv/node13.html in Doeben-Henisch (2010) [2].
18. The graphs are constructed with the DOT-Language using a normal editor under Linux and the KGraphViewer program based on the graphviz package of software tools developed since 1991 by a team at the ATT&Laboratories. For the theory see e.g. Ganser et.al (1993) [26], and Gansner et.al. (2004) [27]. For a tutorial see Ganser et.al (2015) [28]

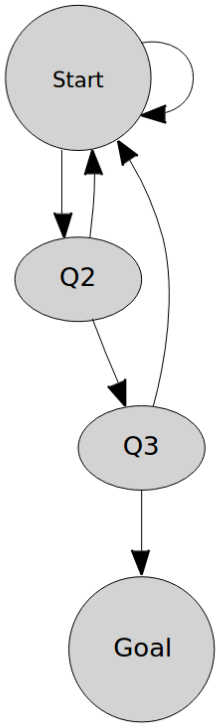

Fig. 5. Electronic door example - bare graph, only nodes

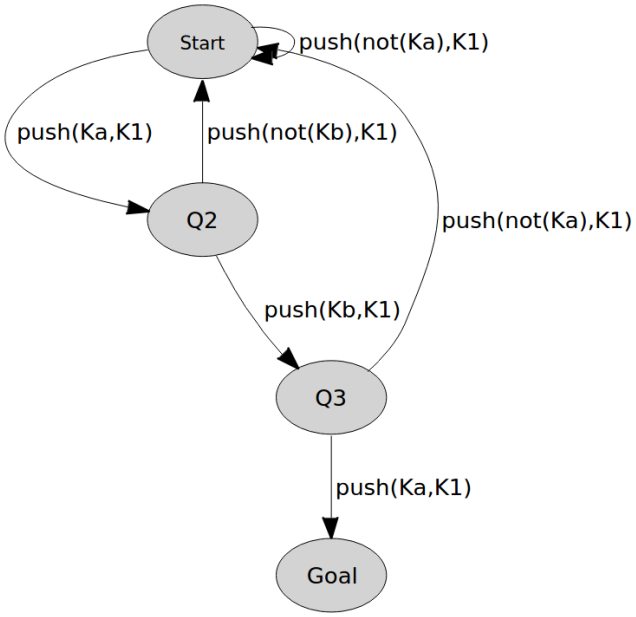

Fig. 6. aai-example electronic door: nodes and minimally labeled edges

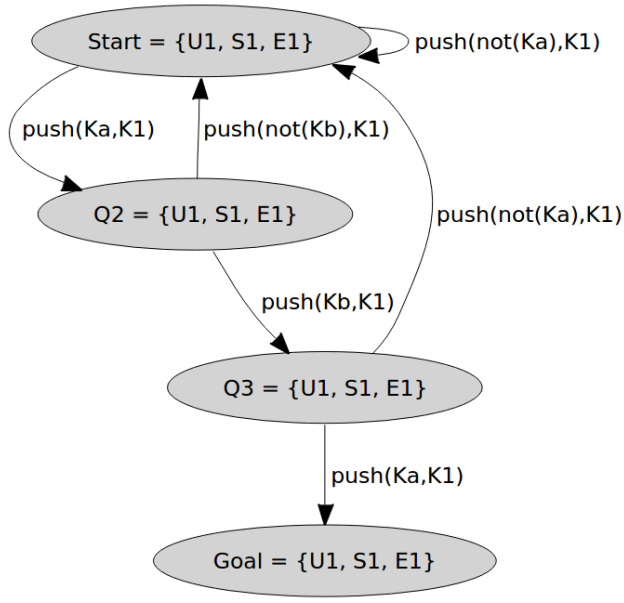

Fig. 7. aai example electronic door with nodes, shortened edge-labels, and subsets of properties

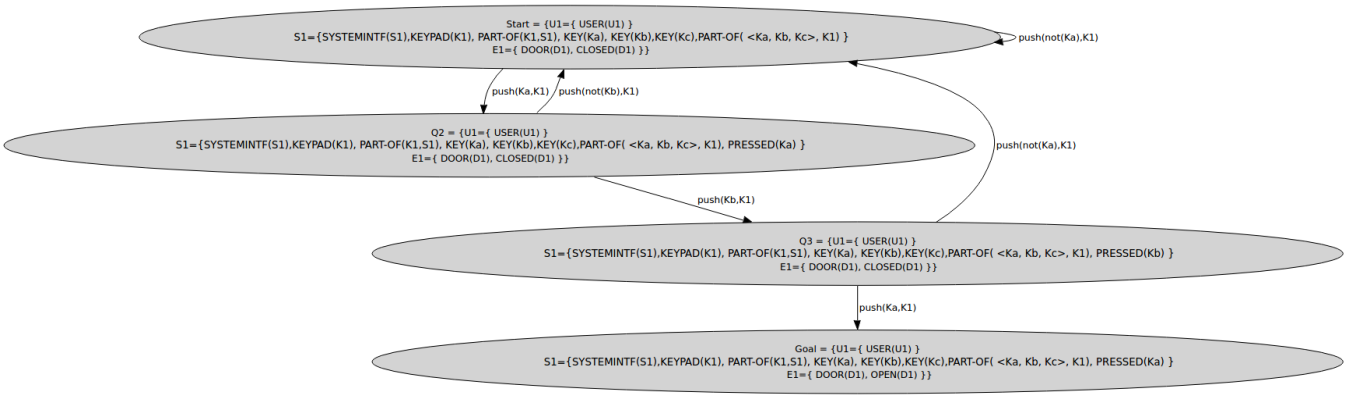

Fig. 8. aai example with a complete graph (only the edge labels are shortened)

4) CHANGE-AS: $\langle Q2, Start, push(not(Kb), K1), d(), c()\rangle$, $\langle$ Q2,Q3,push(Kb,K1), d(), c(PRESSED(Kb))$\rangle$,
5) Q3 = U1 $\cup$ S1 $\cup$ Env1
   U1 = $\{$USER(U1)$\}$
   S1 = $\{$SYSTEMINTF(S1),KEYPAD(K1), PART-OF(K1, S1), KEY(Ka), KEY(Kb),KEY(Kc),PART-OF($\langle Ka, Kb, Kc\rangle$, K1),PRESSED(Kb)$\}$
   Env1 = $\{$DOOR(D1), CLOSED(D1)$\}$

6) CHANGE-AS: $\langle Q3, Start, push(not(Ka), K1), d(), c()\rangle$, $\langle$ Q3,Goal,push(Ka,K1), d(CLOSED(D1)), c(PRESSED(Ka), OPEN(D1))$\rangle$
7) Goal = U1 $\cup$ S1 $\cup$ Env1
   U1 = $\{$USER(U1)$\}$
   S1 = $\{$SYSTEMINTF(S1),KEYPAD(K1), PART-OF(K1, S1), KEY(Ka), KEY(Kb),KEY(Kc),PART-OF($\langle Ka, Kb, Kc\rangle$, K1),PRESSED(Ka)$\}$
   Env1 = $\{$DOOR(D1), OPEN(D1)$\}$

As one can see the formal description of the actor story offers no information about the internal structures which determine the behavior of the different users, the executive actor as well as the assistive actor. To enhance this one has to define additional actor models.

6.2.10 A GOMS Model Example

We will start the construction of a GOMS model for the user of the electronically locked door example. For this we simplify the GOMS-Model format as follows: *IF Input THEN Operation ... ELSE Operation ....* The *Input* can either be some value from the set *I* of possible inputs or from the set *IS* of the internal states of the system. In the used example are all properties of the states a possible input or the properties of the internal states. All these IF-THEN rules are subsumed under the *goal* to enter the open door.

1) GOMS MODEL FOR USER U1
2) INPUT U1 = VB; OUTPUT U1 = MOT

3) GOAL : OPEN(D1) & DOOR(D1) & enter(D1)

   a) IF VB='CLOSED(D1)' & IS='MEM(U1,(Ka,0,Kb,0,Ka,0))' THEN IS='MEM(U1,(Ka,1,Kb,0,Ka,0))' & MOT='push(Ka,K1)'
   b) IF VB='CLOSED(D1)' & IS='MEM(U1,(Ka,1,Kb,0,Ka,0))' THEN IS='MEM(U1,(Ka,1,Kb,1,Ka,0))' & MOT='push(Kb,K1)'
   c) IF VB='CLOSED(D1)' & IS='MEM(U1,(Ka,1,Kb,1,Ka,0))' THEN IS='MEM(U1,(Ka,1,Kb,1,Ka,1))' & MOT='push(Ka,K1)'
   d) IF VB='OPEN(D1)' & IS='MEM(U1,(Ka,1,Kb,1,Ka,1))' THEN IS='MEM(U1,(Ka,0,Kb,0,Ka,0))' & MOT='enter(D1)'

These IF-THEN-Rules follow the general behavior function $\phi : I \times 2^{IS} \longmapsto 2^{IS} \times O$

The system interface S1 has its own GOMS-Model.

1) GOMS MODEL FOR SYSTEM-INTERFACE S1
2) INPUT S1 = K1; OUTPUT S1 = states of the door $\{$CLOSED, OPEN$\}$

3) GOAL : OPEN(D1) & DOOR(D1)

   a) IF INPUT='KEY-PRESSED(K1,Ka)' & IS='CODE(C2,(Ka,0,Kb,0,Ka,0))' THEN IS='CODE(C2,(Ka,1,Kb,0,Ka,0))'
   b) IF INPUT='KEY-PRESSED(K1,Kb)' & IS='CODE(C2,(Ka,1,Kb,0,Ka,0))' THEN IS='CODE(C2,(Ka,1,Kb,1,Ka,0))'
   c) IF INPUT='KEY-PRESSED(K1,Ka)' & IS='CODE(C2,(Ka,1,Kb,1,Ka,0))' THEN IS='CODE(C2,(Ka,1,Kb,1,Ka,1))' & OUT='OPEN(D1)'

Thus a complete Process is an interaction between the actor story (AS) and the both actor models written as GOMS-Models.

6.2.11 Further Extensions

Based on the first version of an actor story one can introduce many more aspects into the story.

The *first enhancement* of an actor story happens when one defines the *different sub states* of a state with their specific properties. Usually the sub states represent typical actors and objects of a situation. In the AAI case are these special objects the executive actors, the system interfaces as assistive actors as

well as the environment as an actor.

The *next enhancement* is represented by the definition of *special behavior functions* for the different actors. These models can be traditional GOMS models or more elaborated formats of models, especially also learning models.

The *third enhancement* could be the *simulation* of the actor story including the actor models by transforming the graph into an appropriate automaton.

6.2.12 Design Principles; Interface Design

Given the actor model AM of an executive actor A_{exec} one can derive some *actor-based principles* $Ax_{A,exec}$, how the real interface $RIntf_{assis,B}$ of an intended assistive actor B should look like to enable an optimal performance with the executive actor A. To make the actor-based principles $Ax_{A,exec}$ as empirically sound as possible one needs sufficient empirical research of real actors doing jobs like those required in the actor story.

From the dependency of the executive-actor-based principles for the design of an assistive-actor interface it follows that the principles can only be as good as the presupposed model.

6.3 Approaching an Optimum Result

To approach a possible optimum for a finite set of demonstrators one applies a set of usability measurements – called 'usability test' – in an iterative process. A *usability test* UT realizes a mapping of given *demonstrators* D into a set of *usability values* V as follows $\upsilon_{UT} : D \longmapsto D \times V$. A usability test includes a finite set of objective as well as subjective sub-tests. The values V of one usability test are usually given as a finite set of points in an n-dimensional space V^n. Thus after a usability test υ_{UT} has been applied to a demonstrator one has an ordered pair (D, V).

To find the *relative best* demonstrator in a finite set of candidate demonstrators $\{(D_1, V_1), (D_2, V_2), ..., (D_m, V_m)\}$ one has to define a *measure* $\mu : 2^{V^n} \longmapsto V^n$ for the assumed finite many n-dimensional values $\{V_1^n, V_2^n, ..., V_m^n\}$ to compare these values and identify for this set an optimal value. Thus $\mu(V_1^n, V_2^n, ..., V_m^n)$ computes a certain $V_i^n \in \{V_1^n, V_2^n, ..., V_m^n\}$.

Applying this measure to the set $\{(D_1, V_1), (D_2, V_2), ..., (D_m, V_m)\}$ gives the best demonstrator of this set.

7 What Comes Next: The Real System

After the completion of the AAI-analysis after n-many iterations[19] one has an actor story AS in four modes {TAS, PAS, MAS, SAS}. Furthermore one has possibly different actor models $\{AM_{exec}, AM_{assist}, ...\}$, and one has a demonstrator $Demo$ with the best interface (D_i, V^n). Between the assistive and the executive actor model exists a logical dependency as well as between all actor models and the actor story: without the actor story the actor models are underspecified. That means the *whole specified behavior* M_{SR} is only given as the complex structure $\langle AS, AM_{exec}, AM_{assis}, \iota_{as,am-exec}, \iota_{as,am-assis}\rangle$ where the mappings ι connect the actor story with the embedded models.

19. It is actually not clear how 'big' this n should be. Some research is needed.

7.1 Logical Design, Implementation, Validation

To *convert* these results *into a real working system* SYS_{assis} one has to process[20] a *logical design phase* δ which takes into account the whole specified behavior M_{SR} as requirements for the behavior of the intended system. The outcome should be a *blue-print* $M_{SR,design}$ for the implementation of a real system, written as

$$\delta \quad : \quad M_{SR} \longmapsto M_{SR,design} \tag{29}$$

Based on such a blue-print the *implementation phase* σ translates these ideas in a physical entity $M_{SR,real}$, written as

$$\sigma \quad : \quad M_{SR,design} \longmapsto M_{SR,real} \tag{30}$$

Because the transfer from the AAI-analysis phase into the logical design phase as well the transfer from the logical design phase into the implementation phase can principally not completely be defined one has to run a *validation phase* υ_v which compares the *behavior requirements* M_{SR} from the AAI-analysis phase with the *behavior* of the *real system* $M_{SR,real}$. The outcome will be some percentage of agreement with the required behavior, written as

$$\upsilon_v \quad : \quad M_{SR} \times M_{SR,real} \longmapsto [0,1] \tag{31}$$

7.2 Conceptual Gap In Systems Engineering?

The theoretically required validation of the behavior of the real system $SYS_{assis,real}$ with the required behavior specified as whole behavior model M_{SR} can not work out directly, as long as the specified behavior is not available in some *implemented* format.

Diverging from the usual processing of systems engineering it will be assumed in this text that the whole specified behavior M_{SR} will be translated into a blue-print within logical design (cf. Formula 29) and similarly will the blue-print version of the whole behavior $M_{SR,design}$ completely be converted in a real version $M_{SR,real}$ including not only the intended assistive actor but also the complementary executive actor as well as the necessary actor story (cf. Formula 30).

One way to realize this concept is to implement real simulators to mimic the required behavior . Especially it should be possible that real users can take over the role of the simulated executive actors within such simulations or the real world is another actor which takes over the role of the simulated world of the simulated actor story.

8 The AASE-Paradigm

The text so far gives only a very limited account of the whole *Actor-Actor Systems Engineering (AASE)* paradigm. We hope to be able to develop it further with many illustrating applications (case studies).

Everybody is invited to share the discussion of this new paradigm with questions, critical remarks, hints, examples, whatever helps to clarify this paradigm.

There exists a minimal project plan to finalize these ideas in a first booklet (theory and case studies) until April 2018 with a publication in May 2018. Then everything can happen.

20. For all assumed phases in a systems engineering process see formula 1 in section 2 and more elaborated in the paper Erasmus & Doeben-Henisch 2011 [4]

REFERENCES

[1] G. Doeben-Henisch and M. Wagner, "Validation within safety critical systems engineering from a computational semiotics point of view," *Proceedings of the IEEE Africon2007 Conference*, pp. Pages: 1 – 7, 2007.

[2] G. Doeben-Henisch, *Formal Specification and Verification: Short Introduction*. http://www.uffmm.org/science-technology/single/themes/computer-science/personal-sites/doeben-henisch/FSV/THEORY/fsv/fsv.html: Gerd Doeben-Henisch, 2010.

[3] L. Erasmus and G. Doeben-Henisch, "A theory of the system engineering process," in *ISEM 2011 International Conference*. IEEE, 2011.

[4] ——, "A theory of the system engineering management processes," in *9th IEEE AFRICON Conference*. IEEE, 2011.

[5] G. Doeben-Henisch, "From hci to aai. some bits of history?" *eJournal uffmm.org*, pp. 1–16, 2018. [Online]. Available: https://www.uffmm.org/2018/04/19/from-hci-to-aai-some-bits-of-history/

[6] ——, "Philosophy of the actor," *eJournal uffmm.org*, pp. 1–8, 2018. [Online]. Available: https://www.uffmm.org/2018/03/20/actor-actor-interaction-philosophy-of-the-actor/

[7] F. Khalique, W. H. Butt, and S. A. Khan, "Creating domain non-functional requirements software product line engineering using model transformations," in *2017 International Conference on Frontiers of Information Technology (FIT)*, Dec 2017, pp. 41–45.

[8] F. Fellir, K. Nafil, and R. Touahni, "Analyzing the non-functional requirements to improve accuracy of software effort estimation through case based reasoning," in *2015 10th International Conference on Intelligent Systems: Theories and Applications (SITA)*, Oct 2015, pp. 1–6.

[9] D. Mairiza, D. Zowghi, and V. Gervasi, "Conflict characterization and analysis of non functional requirements: An experimental approach," in *2013 IEEE 12th International Conference on Intelligent Software Methodologies, Tools and Techniques (SoMeT)*, Sept 2013, pp. 83–91.

[10] A. Suhr, C. Rosinger, and H. Honecker, "System design and architecture ?? essential functional requirements vs. ict security in the energy domain," in *International ETG-Congress 2013; Symposium 1: Security in Critical Infrastructures Today*, Nov 2013, pp. 1–9.

[11] B. Yin, Z. Jin, W. Zhang, H. Zhao, and B. Wei, "Finding optimal solution for satisficing non-functional requirements via 0-1 programming," in *2013 IEEE 37th Annual Computer Software and Applications Conference*, July 2013, pp. 415–424.

[12] X. L. Zhang, C. H. Chi, C. Ding, and R. K. Wong, "Non-functional requirement analysis and recommendation for software services," in *2013 IEEE 20th International Conference on Web Services*, June 2013, pp. 555–562.

[13] I. Menzel, M. Mueller, A. Gross, and J. Doerr, "An experimental comparison regarding the completeness of functional requirements specifications," in *2010 18th IEEE International Requirements Engineering Conference*, Sept 2010, pp. 15–24.

[14] Y. Liu, Z. Ma, R. Qiu, H. Chen, and W. Shao, "An approach to integrating non-functional requirements into uml design models based on nfr-specific patterns," in *2012 12th International Conference on Quality Software*, Aug 2012, pp. 132–135.

[15] M. Kassab, O. Ormandjieva, and M. Daneva, "An ontology based approach to non-functional requirements conceptualization," in *2009 Fourth International Conference on Software Engineering Advances*, Sept 2009, pp. 299–308.

[16] X. Lian, J. Cleland-Huang, and L. Zhang, "Mining associations between quality concerns and functional requirements," in *2017 IEEE 25th International Requirements Engineering Conference (RE)*, Sept 2017, pp. 292–301.

[17] S. Abrahão, C. Gravino, E. Insfran, G. Scanniello, and G. Tortora, "Assessing the effectiveness of sequence diagrams in the comprehension of functional requirements: Results from a family of five experiments," *IEEE Transactions on Software Engineering*, vol. 39, no. 3, pp. 327–342, March 2013.

[18] H. J. Eysenk, *Die IQ-Bibel. Intelligenz verstehen und messen*, 1st ed. Stuttgart: J.G.Cotta'sche Buchhandlung Nachfolger GmbH, 2004, englische Originalausgabe 1998: Intelligence. A New Look.

[19] D. H. Rost, *Intelligenz. Fakten und Mythen*, 1st ed. Weinheim - Basel: Beltz Verlag, 2009.

[20] ——, *Handbuch Intelligenz*, 1st ed. Weinheim - Basel: Beltz Verlag, 2013.

[21] E. R. Hilgard, R. L. Atkinson, and R. C. Atkinson, *Introduction to Psychology*, 7th ed. New York - San Diego - Chicago - et.al.: Harcourt Brace Jovanovich, Inc., 1979.

[22] E. C. Tolman, "Cognitive maps in rats and men," *The Psychological Review*, vol. 55, no. 4, pp. 189–208, 1948, 34th Annual Faculty Research Lecture, delivered at the University of California, Berkeley, March 17, 1947. Presented also on March 26, 1947 as one in a series of lectures in Dynamic Psychology sponsored by the division of psychology of Western Reserve University, Cleveland, Ohio.

[23] H. Ebbinghaus, *Über das Gedächtnis: Untersuchungen zur experimentellen Psychologie*, 1st ed. Leipzig: Duncker & Humblot, 1885, uRL: http://psychclassics.yorku.ca/Tolman/Maps/maps.htm.

[24] ——, *Memory: A Contribution to Experimental Psychology*, 1st ed. New York: Teachers College, Columbia University, 1913, translated from the German Edition 1885 by Henry A. Ruger & Clara E. Bussenius 1913, URL: http://psychclassics.yorku.ca/Ebbinghaus/index.htm.

[25] S. K. Card, T. P. Moran, and A. Newell, *The Psychology of Human-Computer Interaction*, 1st ed. Mahwah (NJ): Lawrence Erlbaum Associates, Inc., 1983.

[26] E. R. Gansner, E. Koutsofios, S. C. North, and G.-P. Vo, "A technique for drawing directed graphs," *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, vol. 19, no. 3, pp. 214–230, 1993.

[27] E. R. Gansner, Y. Koren, and S. North, "Graph drawing by stress majorization," in *Graph Drawing*, ser. Lecture Notes in Computer Science, J. Pach, Ed., no. 3383. Berlin - Heidelberg: Springer-Verlag, pp. 239 – 250.

[28] E. R. Gansner, E. Koutsofios, and S. C. North, "Drawing graphs with dot," pp. 1–40, 2015, online: http://www.graphviz.org/pdf/dotguide.pdf.