AAI - Actor-Actor Interaction. A Toy-Example, No.1

eJournal: uffmm.org, ISSN 2567-6458
13.Dec.2017
Email: info@uffmm.org

Gerd Doeben-Henisch
gerd@doeben-henisch.de

13.Dec. 2017

Contents

Abstract

Following the general concepts of the paper 'AAI - Actor-Actor Interaction. A Philosophy of Science View' from 3.Oct.2017 this paper illustrates a simple application where the difference as well as the interaction between an actor story and several actor models is shown. The details of interface-design as well as the usability-testing are not part of this example.(This example replaces the paper with the title 'AAI - Case Study Actor Story with Actor Model. Simple Grid-Environment' from 15.Nov.2017)

1 Problem

We assume a *problem document* D_p with a content as follows: There is a *room* in a building, whose *door* shall be protected by an *electronic key*. Everybody who wants to *enter* this room by the door has to *key-in* a *number*. If the number is *not correct* then the door stays *closed*, otherwise the door will *open*. No further constraints.[1]

2 AAI-Check

Whether the problem P from the *problem document* D_p is a *candidate for an AAI-analysis* depends from the presence of the following properties:

1. Is there at least *one task (T)* given, which has to be realized by at least one *user (U)*?
2. Is the *environment (E)* to the task T specified?
3. Which kinds of *constraints (C)* are specified?
4. Which kinds of *assistive actors (AssA)* are assumed?

There is at least *one task (T)* given with the task to enter a room in a building. The *environment (E)* of this task is given implicitly by the building from which the room is a part. The *user (U)* is given by all the actors which have the duty and the right to enter this room; this can be human persons, but this could also be *intelligent machines* designed for such a task. The *assistive actor (AssA)* is in this case a system managing an automatic door controlled by an electronic key. The door together with the electronic key represent an interface of the system. Further *constraints (C)* are not yet specified.

3 Actor-Story (AS)

To analyze this problem further one has to dig into the problem so far that one is able to *tell a complete story*, how to *realize* the task of the problem document.

It can be some work to *investigate* the details of such a story.[2] 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.

The idea of telling a story assumes that every task-realization can be seen as a *process* whith some *start state* S_{START} and at least one *goal state* S_{GOAL}. Between start and goal state there can be finitely many *intermediate states* $S_i, i \in n$. The complete set of states ST has at least two members: start and goal. While the goal state is an *end state* with no connections back, it holds for all other states that there can be connections to every other state of the *finite set of states* ST.

A *state* $S \in ST$ is understood as a *collection of static properties* π. A *change* of a state can only happen, if *at least one property* of a state has changed. If S_i is such a state then the state S_{i+1} is the product of such a change if either at least one property π_r of state S_i has been *deleted* in the follow-up state S_{i+1} or a new state π_s has been *created*. *Modifications* are handled here as a combination of deletion of the old version and creation of the new.

That what can *cause a change* is here called an *event*. Events can be of different kinds, mostly *actions* of some actors. Generally are events mappings from states into states: $\epsilon : ST \longmapsto ST$.

To *communicate a story* three basic modes are assumed in the actual theory: *textual*, *mathematical*, as well as *pictorial*. Here only the textual and mathematical modes are illustrated. For the pictorial mode thee some follow-up paper.

3.1 AS as a Text

1. **START:** The user stands before the closed door of the room which he has to enter. The

[1] The missing of any kind of constraint is in a real problem completely unrealistic.

[2] In reality this is often the biggest challenge of a project

door has a keypad with buttons for the numerals (0,1,. ..., 9).

2. **CHANGE-EVENT:** *ACTION*: The user keys in two numbers in sequence. *EFFECT*: Either nothing is changing or the door opens.
3. **GOAL:** The user stands before the door of the room which he has to enter and the door is open.

Comments: This is the actor story from the point of view of an external observer. In an everyday-context the story gets its plausibility if one assumes, that the *user* somehow *knows* what he has to do. Thus in this case he has to open the door. Additionally must the *system* posses a kind of a *control-mechanism* which can *observe*, what the user has done and can therefore *decide* whether the user-actions *match* the control condition. This additional knowledge can not directly be expressed within the actor story. For this one has to construct in this concrete case two *actor models*, one for the human user and one for the system.

3.2 Translation of a Textual AS into a Formal AS

To express something *formally* one has infinite many possibility to do this. In this text we restrict this space to a *Quantifier-Free-Predicate-Language (QFPL)*.[3] Furthermore one has to have a meaning-relation which tells how one has to *interpret* these expressions with regard to the real world.[4]

We will proceed in two steps: (i) First we will directly *translate* the text version of the actor story into a formal expression and then (ii) we will give a graphical version of this formal expression.

1. **START:** The user stands before the closed door of the room which he has to enter. Besides the door is a keypad with buttons for the numerals (0,1,. ..., 9).

[3]For a full definition of this language see section 7
[4]See section 8

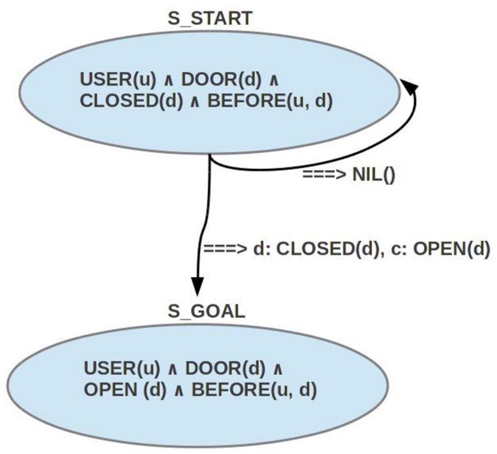
S_START
USER(u) ∧ DOOR(d) ∧ CLOSED(d) ∧ BEFORE(u, d)
===> NIL()
===> d: CLOSED(d), c: OPEN(d)
S_GOAL
USER(u) ∧ DOOR(d) ∧ OPEN (d) ∧ BEFORE(u, d)

Figure 1: Toy Example No.1 - As a formal graph (simplyfied)

2. $S_{START} = \{USER(u) \wedge DOOR(d) \wedge CLOSED(d) \wedge BEFORE(u,d) \wedge KEYPAD(p) \wedge NUMERALS(\{0,1,...,9\})\}$
3. **CHANGE-EVENT:** *ACTION*: The user keys in two numbers in sequence. *EFFECT*: Either nothing is changing or the door opens.
4.
$$\begin{array}{rcl} ENTER(u,\{2,3\},p) & \Rightarrow & \\ S_{GOAL} & = & S_{START} \\ Case1 & : & NIL() \\ Case2 & : & \\ DELETE & S_{GOAL} & CLOSED(d) \\ CREATE & S_{GOAL} & OPEN(d) \end{array}$$
5. **GOAL:** The user stands before the door of the room which he has to enter and the door is open.
6. $S_{GOAL} = \{USER(u) \wedge DOOR(d) \wedge OPEN(d) \wedge BEFORE(u,d) \wedge KEYPAD(p) \wedge NUMERALS(\{0,1,...,9\})\}$

3.3 AS as a Formal Expression

1. $S_{START} = \{USER(u) \wedge DOOR(d) \wedge CLOSED(d) \wedge$

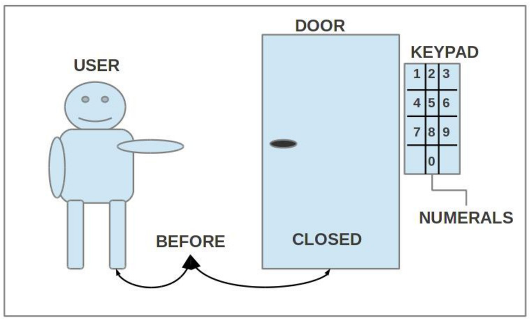

Figure 2: Comic version of the example start state

$BEFORE(u, d) \wedge KEYPAD(p) \wedge NUMERALS(p, \{0, 1, ..., 9\})\}$

2. $$\begin{array}{cccc} ENTER(u, \{2, 3\}, p) & \Rightarrow & & \\ & S_{GOAL} & = & S_{START} \\ & Case1 & : & NIL() \\ & Case2 & : & \\ & DELETE & S_{GOAL} & CLOSED(d) \\ & CREATE & S_{GOAL} & OPEN(d) \end{array}$$

3. $S_{GOAL} = \{USER(u) \wedge DOOR(d) \wedge OPEN(d) \wedge BEFORE(u, d) \wedge KEYPAD(p) \wedge NUMERALS(\{0, 1, ..., 9\})\}$

Comment: There can be more than only one event being attached to a state. Two different events attached to the same state have to have either different effects or different target states.

3.4 Translation of a Formal AS into a Pictorial AS

Here a simple example of a pictorial version of the actor story. The mapping between the formal expressions and the pictorial elements is giving implicitly in the pictures themselves.

1. **START:** See figure 2
2. **CHANGE-EVENT:** See figure 3
3. **GOAL:** See figure 4

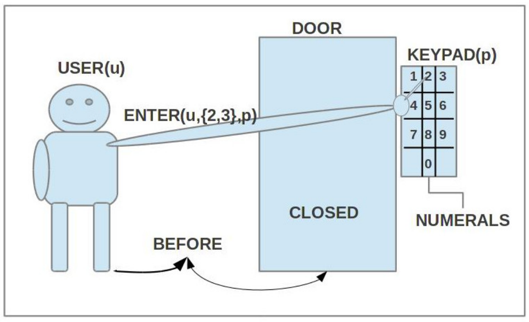

Figure 3: Example action: enter numbers in keypad

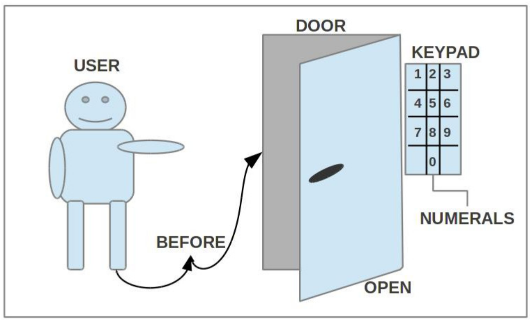

Figure 4: Example: change after the enter-action

4 Actor-Model (AM)

Reading the actor story one can distinguish states and events, but one can in case of an event not necessarily clearly decide the exact conditions why a state change will happen.

Thus if – as in the example above – a user keys-in two numbers one knows in general that there can only be two outcomes: either the numbers are matching the required code or not. If not then the start state will not change, otherwise, it will.

To remedy this uncertainty one has to construct explicit actor models which provide all the information necessary to decided the concrete outcome when such state changes really will happen and when not.

To enable such 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 *inputs (I)* allowing some kind of *perceptions* of his environment as well as *outputs (O)* allowing changes, modifications in the environment. Furthermore it is assumed that every actor has some *behavior function* ϕ which determines how the actor will respond with an output given some inputs. More formally this can be written as follows:

Def: Input-Output System (IOSYS)

$$\begin{aligned} IOSYS(x) \quad & iff \quad x = \langle I, O, IS, \phi \rangle && (1) \\ I \quad & := \quad Input \\ O \quad & := \quad Output \\ IS \quad & := \quad Internal\ states \\ \phi \quad & : \quad I \times IS \longmapsto IS \times O \end{aligned}$$

Thus the behavior function ϕ generates an output O depending from the actual input I and some internal states IS, and – this is reflexive – the behavior can again change the internal states IS such, that these are in another state for a next response. This means that the same input can be followed by different responses depending from the internal states. This includes a property which often is called *learning*.

Applying these concepts for a first sketch of actor models in the above example we can write the following texts.

4.1 AM for the User as a Text

1. **INPUT:** The user can see the door as an object and the keypad for entering numbers. He can also distinguish the buttons with the different numerals $\{0, 1, ..., 9\}$ marked on the buttons.
2. **OUTPUT:** The user can use at least one finger to press the different buttons.
3. **PHI:** When the door is closed then will the user press two times the buttons with certain numerals to realize a code of two consecutive numbers. If the door will not open the user will repeat this action.

4.2 AM for the System as a Text

1. **INPUT:** The system with the keypad- and door-interface can sens the pressing of every button associated with the numbers $\{0, 1, ..., 9\}$ and the properties OPEN and CLOSED of the door.
2. **OUTPUT:** The system with the keypad- and door-interface can change the property of the door to OPEN or to CLOSE.
3. **PHI:** The default property of the door is CLOSED. When two buttons have been pressed one after the other and the associated numbers in a short-term memory as (n1,n2) match a stored code (c1,c2) then the system will open the door. It can sens when the door is open.

5 Combined AS and AM as a Text

Now, with these additional models one can specify the complete behavior of the example more precise.

1. AS: **START:** The user stands before the closed door of the room which he has to enter. Besides the door is a keypad with buttons for the numerals (0,1,. ..., 9).
2. AM-User: The user sees the closed door and he knows that he has to press with a finger the buttons of the keypad with the numbers of the known 2-number-code (c1,c2).
3. AM-System: The systems is prepared to receive button-pressings.
4. **CHANGE-EVENT:** *ACTION*: The user keys in two numbers in sequence. *EFFECT*: Either nothing is changing or the door opens.
5. AM-User: The user presses with a finger the buttons of the keypad with the numbers of the known 2-number-code (c1,c2).
6. AM-System: The system senses the two button presses with two associated numbers (n1,n2). If the sensed numbers (n1,n2) match the stored 2-number-code (c1,c2) in the system then the system will change the door to OPEN, otherwise not.
7. **GOAL:** The user stands before the door of the room which he has to enter and the door is open.
8. AM-User: The user sees the opened door and he knows that he can now enter the room
9. AM-System: The system senses that the door is open.

5.1 AM as an Algorithm

Above we have a description of actor models as a text. But the text is already structured using concepts like *input*, *output* as well as *behavior function*. This points to the general structure given in the definition of an input-output-system (cf. page 4).

This is a very general mathematical structure and gives no hint how one should *implement* such a structure as an algorithm.

One very old but still widely used paradigm is the so-called *GOMS*-Paradigm, an abbreviation for *Goals, Operators, Methods, and Selection rules*.[5]. In Card et.al. it is described as follows:

- **METHOD:** A *method* is a composition of goals and operators following the goal to realize it.
- **GOAL:** A *goal* is something to be achieved and will be represented by some *language expression*.
- **OPERATOR:** An *operator* is some *concrete action* which has inputs, outputs, as well as a duration.
- **SELECTION RULE:** A selection rule has an IF-THEN-ELSE structure: *IF* a certain condition is fulfilled, *THEN* some method or goal or operation will be selected, otherwise the *ELSE*-part will be executed.

These concepts are still a bit abstract. To realize a GOMS-model on a real computer one has to translate it in some modern *computer programming language*.

In the next sections it will be shown how a simple simulation of the example No.1 could look like.

6 Simulation

To be done ...

6.1 Simulating the AS

To be done ...

6.2 Simulating the AM

To be done ...

6.3 Simulating AS with AM

To be done ...

[5]A first extensive usage of a GOMS model can be found in Card et al. (1983) [CMN83]:chapt.5, especially pp.144ff

7 Appendix: Formalisms

7.1 Set of Strings

To describe some intended meaning formally one needs at least a formal language. Basically every kind of a *formal language* L is a *finite alphabet* Σ, whose elements – the symbols – can be *concatenated* one after the other to sequences of *symbols* called *words*. A word w is either *empty* ϵ or it has a certain length $|w| > 0$. The concatenation of symbols and words is *associative* and there exists an *identity element* e, which is given by the empty word ϵ.

Def: Formal Language L

$$\begin{aligned}
L(x) &\quad iff \quad x = \langle \Sigma, \Sigma^*, conc, ||, \epsilon \rangle && (2)\\
\Sigma &\quad := \quad Finite\ set\ of\ distinguished\ symbols && (3)\\
\Sigma &\quad \subseteq \quad \Sigma^* && (4)\\
WORD(w) &\quad iff \quad w \in \Sigma^* && (5)\\
w, w' \in \Sigma^* &\quad \Rightarrow \quad conc(w, w') = ww' \wedge ww' \in \Sigma^* && (6)\\
|w| &\quad := \quad Length\ of\ words\ w \in \Sigma^* && (7)\\
\epsilon &\quad := \quad Empty\ word && (8)\\
|\epsilon| &\quad = \quad 0 && (9)\\
\epsilon w &\quad = \quad w\epsilon \ = \ w && (10)\\
\{\} &\quad := \quad emptyset\ \emptyset && (11)\\
\{\} &\quad \neq \quad \{\epsilon\} && (12)\\
a, b, c \in \Sigma^* &\quad \Rightarrow \quad (ab)c \ = \ a(bc)\ (ASSOC) && (13)\\
e &\quad := \quad Identity\ element && (14)\\
a \in \Sigma^* &\quad \Rightarrow \quad ae \ = \ ea \ = \ a\ (IDENTITY) && (15)
\end{aligned}$$

7.2 Predicate Language

For the task of modeling it is helpful to introduce a more specialized language called *quantifier-free predicate language* $QFPL$ (or short L_P), which will serve as part of the modeling language L_m used in the tetx. In this L_P one has *names* for objects, *predicates* for properties and relations, and some *logical operators*, but no *quantifiers*. Predicates and names together represent *atomic quantifier-free predicate expressions* $AQFPE$, which represent a subset of the L_P, which will be called *atomic* L_P or AL_P. Related to some given *domain of reference* D^R one can define an *interpretation* ι which maps every atomic predicate expression into the domain of reference.

If there are expressions ϕ, ϕ' from L_P one can apply some *logical operators* onto these expressions. In this context the following logical operators are assumed: *negation* $\neg$ as in $\neg\phi$ and the operator *and* $\wedge$ as in $\phi \wedge \phi'$.

Def:Atomic Quantifier-free Predicate Expressions $AQFPE$ (or: L_P)

$$\begin{aligned} AL_P &\subseteq \Sigma^* & (16)\\ P(...) &:= set\ of\ predicates\ (uppercase\ letters) & (17)\\ x... &:= set\ of\ names\ (lowercase\ letters) & (18)\\ P(x...) &:= atomic\ predicate\ expression & (19)\\ P(x...) &\in AL_P & (20) \end{aligned}$$

Thus, an atomic expression $a \in AL_P$ is an n-ary predicate $P(...)$ with n-many arguments $x_1, ..., x_n$, where each x_i is a name of an object such, that it is decidable with regard to a domain of reference D^R whether the object in the argument of the predicate is part of D^R and that the predicate is part of D^R and that the relation between the objects mentioned and the predicate are given in D^R as assumed in the expression.

In general it is a difficult – and until today a not completely solved – problem, how one can establish a fully correct mapping between expressions of a language and some parts of the intended real world sections, as long as the referees of these decisions are human persons.

Def:Logical Connectors LC

$$\begin{aligned} \{\neg, \wedge\} &\subseteq LC & (21)\\ \neg &:= negation & (22)\\ \wedge &:= and & (23) \end{aligned}$$

Def: Quantifier Free Predicate Language L_P

$$\begin{aligned} AL_P &\subseteq L_P & (24)\\ \phi \in L_P &\Rightarrow \neg\phi \in L_P & (25)\\ \phi, \phi' \in L_P &\Rightarrow \phi \wedge \phi' \in L_P & (26) \end{aligned}$$

Because the binding of the operator $\neg$ to an expression is *stronger* then the binding of the operator $\wedge$ one does not need additional brackets in an expression.

8 Appendix: The Meaning of Expressions

Using an expression e of some language L $e \in L$ is only associated with some additional *meaning* μ if there exists some mapping from the set of expressions given by L into some *domain of references (DR)*.

Thus in the English Language L_{EN} the expression 'dog' is usually rooted in some *learned pattern* given as *internal states (IS)* of the user, which corresponds some properties in the *real world (W)*. The internal states represent only a *subjective* kind of meaning compared to the *objective* meaning in the real world. Often there exists also a *derived objective* meaning if one uses *pictorial elements* which are accepted by speakers as *sufficient representations* of the intended meaning. Thus if one has a photo or a comic painting of a 'real dog' then this pictorial representation is often used as if it is an objective meaning.

8.1 States

In the case of *states* it is assumed that states are *sets of properties* and that properties can be written with predicate expressions from the language of atomic quantifier-free predicate expressions L_{AQFPE}.

Thus a state S could be written as $S = \{DOOR(d) \wedge OPEN(d)\}$. An individual expressions like *DOOR(d)* representing a property has as such *no meaning* as long there exists no mapping into some domain of reference.

To map such formal expressions directly into the real world W without any kind of ambiguity is until today technically impossible. As the history of philosophy of science could shown with numerous examples it is even not possible in the case of so-called empirical theories.[6] Therefore one has to compromise to deal with this requirement.

[6]See e.g. the overview in F.Suppe (1977) [Sup77]

The simplest procedure which most authors are using is to use as formal expressions those which resemble expression of everyday English like in the before mentioned expression 'DOOR(d)' or 'OPEN(d)'. Thus doing this the author of the formal expressions with similarities to an everyday language is implicitly assuming that the reader has some knowledge of the everyday language with their meaning and that the reader *automatically* is using the meaning mechanisms of the known everyday language without giving explicitly a technical meaning relation μ. In this case the reader will automatically interpret the formal expression 'DOOR(d)' in the manner that there is some object with the name 'd' and this object has the property 'to be a door' in the intention of the everyday meaning of the English language. Similarly with the expression 'OPEN(d)': The assumed object 'd' has the additional property 'to be open'.

In most cases this can work sufficiently well. But has to keep in mind that these implicitly introduced meanings are not clearly defined and are basically *vague* and *ambiguous*. this means that they are possible sources of errors.

Knowing this one can interpret even complex formal expressions like the following one: $S_{START} = \{USER(u) \wedge DOOR(d) \wedge CLOSED(d) \wedge BEFORE(u, d) \wedge KEYPAD(p) \wedge NUMERALS(p, \{0, 1, ..., 9\})\}$. Applying an 'everyday meaning-relation' μ_{EN} could then give the interpretation that we have a start-state G_{START} with some properties like an object 'd' which is a door and is closed, and an object 'u' which is a user and 'stands before' the door. There is also an object 'p' which is a keypad with the numerals '0, ..., 9'.

A compromise between the implicit meaning relation μ_{EN} and a more explicit meaning relation μ_{Pict} realized by pictorial elements is a *pictorial language* L_{pict} which can generate 2-dimensional pictures whose elements are mapped into formal expressions of L (cf. figure 2).

8.2 Changes by Events

States are assumed to be *static*. All properties are static. In that moment something is *changing* then the state is changing from the *actual* state S_i to a *follow-up* state S_{i+1}, written here as $S_i \Rightarrow S_{i+1}$.

While the *meaning of the properties of a state* $\mu(S)$ is either a *borrowed meaning* from some everyday language into real world or a defined meaning into some pictorial language it is difficult to define a *change*, because a change is no direct object. A change is a mapping between two different sets of properties like $\mu : ST \longmapsto ST$.

If one looks to the states as finite sets of properties then a follow-up state S_{i+1} differs from the source state S_i either by some property which has been *deleted* or has been *created*.

Thus it could be an idea to describe changes as *properties of properties* from a *meta-level*: given are two states S_i, *Si+1* and S_{i+1} can be constructed out of S_i by explicitly deleting properties as well as creating. The general schema can look like this:

Def: CHANGE-EVENT

$$\begin{array}{rcl} P(...) & \Rightarrow & \\ S_{i+1} & = & S_i \\ DELETE & S_{i+1} & P(...), ..., P(...) \\ CREATE & S_{i+1} & P(...), ..., P(...) \end{array} \quad (27)$$

Example:

Given is the state S_i as $\{USER(u) \wedge DOOR(d) \wedge CLOSED(d) \wedge BEFORE(u,d) \wedge KEYPAD(p) \wedge NUMERALS(\{0,1,...,9\})\}$

Then the action occurs that the user enters the numerals $\{2,3\}$ into the keypad. The effect of this action is that the door opens; a new state arises:

$$\begin{array}{rcl} ENTER(u, \{2,3\}, p) & \Rightarrow & \\ S_{i+1} & = & S_i \\ DELETE & S_{i+1} & CLOSED(d) \\ CREATE & S_{i+1} & OPEN(d) \end{array} \quad (28)$$

References

[CMN83] Stuart K. Card, Thomas P. Moran, and Allen Newell. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates, Inc., Mahwah (NJ), 1 edition, 1983.

[Sup77] Frederick Suppe, editor. *The Structure of Scientific Theories*. University of Illinois Press, Urbana - Chicago - London, 2 edition, 1977.