Towards a Formal Model of Recursive Self-Reflection

Axel Jantsch
TU WIen
[Vienna, Austria]
axel.jantsch@tuwien.ac.at

Abstract

Self-awareness holds the promise of better decision making based on a comprehensive assessment of a system's own situation. Therefore it has been studied for more than ten years in a range of settings and applications. However, in the literature the term has been used in a variety of meanings and today there is no consensus on what features and properties it should include. In fact, researchers disagree on the relative benefits of a self-aware system compared to one that is very similar but lacks self-awareness.

We sketch a formal model, and thus a formal definition, of self-awareness. The model is based on dynamic dataflow semantics and includes self-assessment, a simulation and an abstraction as facilitating techniques, which are modeled by spawning new dataflow actors in the system. Most importantly, it has a method to focus on any of its parts to make it a subject of analysis by applying abstraction, self-assessment and simulation. In particular, it can apply this process to itself, which we call recursive self-reflection. There is no arbitrary limit to this self-scrutiny except resource constraints.

2012 ACM Subject Classification Computer systems organization → Embedded and cyber-physical systems; Computer systems organization → Self-organizing autonomic computing

Keywords and phrases Cyber-physical systems, self-aware systems, self-reflection, self-assessment

Digital Object Identifier 10.4230/OASIcs.ASD.2019.6

1 Introduction

When the autonomous system itself and its environment are exceedingly complex, dynamic and unpredictable, a comprehensive and correct assessment of the system's situation is a prerequisite for good decisions. This insight has led to a proliferation of research that approach the challenges from various angles and run under names like autonomic computing [29, 32] and organic computing [23]. Self-awareness has become associated with many self-* properties including self-monitoring and self-adaptation and it has been identified as key element for designing complex computer systems [1] and cyber-physical systems [3]. The challenge has been picked up by funding organizations such as DARPA [25] and the European Commission [4] who have allocated significant funds for this research. These efforts have resulted in many conference papers, journal articles and four books [11, 18, 26, 32]. Several surveys have systematically reviewed the research landscape [9, 16, 19, 27].

While the term self-awareness is used in the literature in different ways and various definitions have been provided, researchers at a 2015 Dagstuhl Seminar have proposed a comprehensive working definition, as summarized by Kounev et al. [13], which is worth quoting in full:

> Self-awareness, in this context, is defined by the combination of three properties that IT systems and services should possess:
>
> 1. *Self-reflective*: i) aware of their software architecture, execution environment and the hardware infrastructure on which they are running, ii) aware of their operational

Workshop on Autonomous Systems Design (ASD 2019).
Editors: Selma Saidi, Rolf Ernst, and Dirk Ziegenbein; Article No. 6; pp. 6:1–6:16
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

goals in terms of QoS requirements, service-level agreements (SLAs) and cost- and energy-efficiency targets, iii) aware of dynamic changes in the above during operation,

2. *Self-predictive*: able to predict the effect of dynamic changes (e.g., changing service workloads or QoS requirements) as well as predict the effect of possible adaptation actions (e.g., changing service deployment and/or resource allocations),
3. *Self-adaptive*: proactively adapting as the environment evolves in order to ensure that their QoS requirements and respective SLAs are continuously satisfied while at the same time operating costs and energy-efficiency are optimized.

Two reference architectures have been developed where these principles are at least partially implemented, the EPiCS architecture [17, 18] and the Learn-Reason-Act loop [12].

Although these and similar definitions are useful, they are still vague and imprecise. For instance the definition above repeatedly uses the term "aware" in defining self-awareness and thus does not explain what is meant by *awareness*. What would be the difference between being "aware of operational goals in terms of QoS requirements" and storing a list of QoS requirements and using them during operation? Does "aware of dynamic changes" mean that some variables and models are updated and then the system continues to use the new values, or does it mean that the system realizes that a change has happened and ponders its cause and its implications?

Not least because much of the research on self-awareness is inspired by psychology (e.g. see [16]) the term "self-awareness" seems to suggest more than a set of variables and models that represent some features of the system, that it can access during operation. In particular, the definition above, and all other definitions presented in the computing literature, leave it open if the self-models are self-created based on self-observations or if the self-models are provided by the designer. If the latter is the case, would the self-model keep track if the reality changes? Also, should the system be aware of its self-awareness? And should this awareness be recursive without bounds? Should the self-awareness be self-adaptive as the environment, the system, and the self-model changes, as tasks become more or less urgent, as resources become available or are withdrawn?

Different answers to these questions can lead to technically useful solutions and there seems to be a spectrum between the point where everything is defined at design time and the point where everything is self-constructed at run-time. Self-models, self-adaptation and self-awareness push towards run-time, but how far should we go and how do we determine the trade-offs?

Addressing these questions will require to be precise with terminology and to define and model the involved concepts explicitly and in stringent formal terms. The following is an attempt of a formal model of self-awareness but it should not be taken as the final solution but rather as a first step. At several points we are less precise and less complete than we would like to be, partially due to limited space but mostly because of an incomplete understanding of what would be the best choices that lead to a sound basis for modeling, design, exploration and verification. The hope is that a precise formalism will eventually facilitate a design methodology and effective exploration of the design choices in the space of self-aware systems.

2 Notation

We use dynamic dataflow based on static dataflow process network models such as [7, 8, 14]. But we generalize these models to allow for dynamic changes in the network structure which results in dynamic dataflow not unlike the dynamic model proposed by Grosu and Stølen [5,6].

The processes in the network are called actors and they communicate with each other through signals.

2.1 Signals

Actors communicate with each other by writing to and reading from signals. Signals may be produced by sensors or may be the control inputs for actuators. The environment of an actor can also be modeled as an actor; hence, the actors communicate with each other and with the environment by means of signals.

Given is a set of values V, which represents the data communicated over the signals. *Events*, which are the basic elements of signals, are or contain values. Signals are sequences of events. Sequences are ordered and we use subscripts as in e_i to denote the i^{th} event in a signal. E.g. a signal may be written as $\langle e_0, e_1, e_2 \rangle$. In general signals can be finite or infinite sequences of events and S is the set of all signals.

We assume an untimed model of computation [8, 15] and signals encode only a partially ordered time, meaning that events within one signal represent a relative ordering in time but events in different signals are not directly related in time. I.e. an event e appearing before another event e' in the same signal occurs before e'; but we do not know which of two events in different signals occur earlier or later.

We use angle brackets, "$\langle$" and "$\rangle$", to denote ordered sets or sequences of events, but also for sequences of signals if we impose an order on a set of signals. $\#s$ gives the length of signal s. Infinite signals have infinite length and $\#\langle\rangle = 0$.

We use the notation $\texttt{Signal}(V)$ to denote a type of signal that consists of elements of the set V. E.g. $\texttt{Signal}(\mathbb{R})$ would denote signals with real numbers, $\texttt{Signal}(\mathbb{N})$ would denote signals with natural numbers and $\texttt{Signal}(\{T, F\})$ would denote signals that contain the two types of elements T and F.

Signals are point-to-point connections between actors, and there can only be one producer and one consumer for each signal. If events of a signal should be used by more than one actor, we need a copy actor that copies the input signal to two or more output signals. If two or more actors should contribute to one signal, we need a merge actor that defines how the events from the producing actors are merged. In the figures of this article we sometimes omit the copy and merge actors for convenience and clarity, but the model always requires them.

2.2 Signal Partitioning

We use the partitioning of signals into sub-sequences to define the portions of a signal that are consumed or emitted by an actor in each activation cycle.

A *partition* $\pi(\nu, s)$ of a signal s defines an ordered set of signals, $\langle r_i \rangle$, which, when concatenated together, form the original signal s. The function $\nu : \mathfrak{S} \to \mathbb{N}$ defines the lengths of all elements in the partition, where $\mathfrak{S}$ is the set of states of the partitioning process. For example, if we have a signal $s = \langle 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 \rangle$, the partitioning process runs through the sequence of states $\langle q_0, q_1, q_2, \cdots \rangle$, and $\nu(q_0) = \nu(q_1) = 3, \nu(q_2) = 4$, then we get the partition $\pi(\nu, s) = \langle \langle 1, 2, 3 \rangle, \langle 4, 5, 6 \rangle, \langle 7, 8, 9, 10 \rangle \rangle$.

Note, that there is nothing static about this partitioning, because the size of the next partition can be determined by the actor during each activation. Signal partitioning only captures the notion of activation cycles of actors which repeatedly consume part of the input and produce more and more of the output.

2.3 Actors

An Actor $A \in \mathcal{A}$ maps a set of input signals to a set of output signals. Actors repeatedly evolve through activation cycles, and in each cycle part of the input signals are consumed and part of the output signals are generated. Also, an actor may have an internal state which is also drawn from the set of all subsets of values V.

$\mathcal{A}$ denotes the set of actors, and $\mathfrak{S} = \mathcal{P}(V)$, the power set of the set V, denotes the set of states, with $\epsilon \in \mathfrak{S}$ being the empty set, i.e. the state that has no values. To capture the notion of activation cycle and to model the behavior of actors, we introduce the state, the next state function g, the output encoding function f, and the partitioning function ν of an actor. Thus an actor with n input and m output signals is an eight tuple $A = \langle \mathfrak{T}, I, O, z_0, f, g, \nu, \vec{\mathfrak{m}} \rangle$ as follows:

$$
\begin{array}{ll}
\mathfrak{T} \subseteq \mathfrak{S} & \text{... set of states} \\
I \subseteq \mathcal{P}(S) & \text{... set of input signals} \\
O \subseteq \mathcal{P}(S) & \text{... set of output signals} \\
z_0 \in \mathfrak{T} & \text{... the initial state} \\
\nu : \mathbb{N} \rightarrow \mathcal{P}(\mathbb{N}) & \text{... input partitioning function} \\
f : \mathcal{P}(S) \times \mathfrak{S} \rightarrow \mathcal{P}(S) & \text{... output encoding function} \\
g : \mathcal{P}(S) \times \mathfrak{S} \rightarrow \mathfrak{S} & \text{... next state function} \\
\vec{\mathfrak{m}} : \mathfrak{S} \rightarrow \texttt{Action} & \text{... a meta operator}
\end{array}
$$

Note, that the sets of input and output signals can dynamically change during the operation, and, consequently, the functions ν, f and g may have to deal with different numbers of signals at different time. Events from an input signal not consumed by an actor during an activation cycle are left in the signal for later consumption, and if no events are generated for a particular output signal, the signal is unchanged. Thus, an actor is free to ignore input and output signals in which case they are never modified.

The meta operator $\vec{\mathfrak{m}}$ can invoke any of the following actions, which modify the global dataflow network:

`Action`	
`addsig`(s)	add signal s
`connectisig`(s, A)	add signal s to the set of input signals in actor A.
`connectosig`(s, A)	add signal s to the set of output signals in actor A.
`delsig`(s)	delete input signal s and remove it from the input and output signals of the connected actors.
`addactor`$(\mathfrak{T}', I', O', z_0', f', g', \nu', \vec{\mathfrak{m}}')$	create a new actor $A' = \langle \mathfrak{T}', I', O', z_0', f', g', \nu', \vec{\mathfrak{m}}' \rangle$
`delactor`(A')	delete actor A'
`nop`	do nothing

Since the `addactor` action assumes all involved signals exist, unconnected signals have to be created first with `addsig` and used in `addactor` calls or attached to existing actors with `connectisig` and `connectosig` actions.

An actor A can be applied to a set of input signals to generate events on output signals. It does so by repeatedly consuming values from the input signals and producing values for the output signals. Each such activity is called *activation* or *activation cycle.* The number of input values consumed in each activation cycle is determined by the partitioning function ν. g computes the sequence of states $\langle z_1, z_2, z_3, \cdots \rangle$ and f gradually produces the values for the output signals. For actor A we write $A(\langle s_1, s_2 \rangle) = \langle s_3, s_4 \rangle$ to denote an actor that consumes two input signals s_1 and s_2 and generates two output signals s_3 and s_4.

Let A and B be two actors with input and output signals I_A, I_B, O_A and O_B, respectively. If a subset O'_A of A's output signals has the same type as a subset of B's input signals I'_B, they can be connected such that the signals $O'_A = I'_B$. This results in a *compound actor* C with input signals $I_C = I_A \cup (I_B \setminus I'_B)$ and output signals $O_C = (O_A \setminus O'_A) \cup O_B$. The semantics of such actor networks (or process networks) are developed in [7], together with an analysis of loops and deadlocks.

3 Abstraction

Abstraction is a prerequisite for self-modeling because the model that an actor entertains of itself, must be simpler, hence more abstract, than itself. Since we try to capture the notion of unlimited recursive self-modeling, we need to make sure, that the self-model at one level is more abstract than the self-model of the previous level. Here we do not show what a "good" abstraction is or how to derive it, but we only show that certain signal abstractions, that we use in later sections, have reduced information content.

3.1 Signal Abstraction

Given two signals $s_1 : \texttt{Signal}(V_1)$ and $s_2 : \texttt{Signal}(V_2)$, an abstraction of s_1 is a mapping $B_\alpha : \texttt{Signal}(V_1) \to \texttt{Signal}(V_2)$ with an abstraction function $\alpha : \langle V_1 \rangle \to V_2$ that maps sequences of s_1 onto individual values of s_2.

For instance, if a thermometer measures the sequence of temperature values as $s_1 = \langle 36.7, 36.8, 36.7, 36.8, 36.9, 36.9, 37.0, 37.0, 37.1, 37.2, 37.3, 37.2, 37.3, 37.3, 37.4, 37.5, 37.6, 36.6 \rangle$, then the abstraction B_α with

$$\alpha(\langle t_1, t_2, t_3 \rangle) = \begin{cases} \mathfrak{l} & \text{if } (t_1 + t_2 + t_3)/3 < 35.5 \\ \mathfrak{n} & \text{if } 35.5 \leq (t_1 + t_2 + t_3)/3 < 37.5 \\ \mathfrak{e} & \text{if } 37.5 \leq (t_1 + t_2 + t_3)/3 < 38.5 \\ \mathfrak{h} & \text{if } 38.5 \leq (t_1 + t_2 + t_3)/3 \end{cases}$$

the abstraction B_α would map three consecutive temperate measurements onto one symbol, i.e. $B_\alpha(s_1) = \langle \mathfrak{n}, \mathfrak{n}, \mathfrak{n}, \mathfrak{n}, \mathfrak{n}, \mathfrak{e} \rangle$.

Many signal processing functions can be considered abstractions. E.g. an ECG signal can be abstracted into a sequence of pulse periods, or into a sequence of P, Q, R, S, T symbols to indicate the main components of the ECG signal. The important points are that the abstracted signal represents less information and thus can be encoded with fewer bits, and that it reflects regularities and repetitive patterns. If a sequence of values appears many times in the input signal, this sequence can be abstracted into one abstract symbol. (GrammarViz [28] and unsupervised symbolization [20] are examples for general methods for signal abstraction.)

3.2 Information Reduction by Abstraction

The *Shannon Entropy* [2] provides a formalism for measuring the information content of a signal.[1] Let $V = \{v_1, v_2, \dots, v_N\}$ be the set of symbols that appear on the signal s and let

[1] The Shannon Entropy assumes independent, identically distributed random variables, which in fact cannot be assumed in our case. In the following we use the Shannon Entropy as an estimate but recognize the need for a more appropriate model.

the probability of v_i be p_i, then the Shannon Entropy H of signal s is

$$H(s) = -\sum_{i=1}^{N} p_i \log p_i.$$

$H(s)$ gives the average amount of information per symbol. For a signal of length $m, s = \langle e_1, e_2, \ldots, e_m \rangle$ the information content is

$$I(s) = -\sum_{j=1}^{m} \mathbf{E}[\log p(e_j)] = -m \sum_{i=1}^{N} p_i \log p_i$$

where $p(e)$ is the probability of event e and N is the number of distinct symbols.

3.2.1 Value Abstraction

We consider two types of abstraction, time and value abstraction. Let the value abstraction function α_v be

$$\alpha_v(\langle x \rangle) = \begin{cases} \mathfrak{A} & \text{if } x = \mathfrak{a} \text{ or } x = \mathfrak{b} \\ x & \text{otherwise} \end{cases}$$

which maps two symbols $\mathfrak{a}$ and $\mathfrak{b}$ onto the same symbol $\mathfrak{A}$ and leaves all other symbols unmodified. The abstraction B_{α_v} leaves the lengths of signals unchanged but reduces the number of different symbols by one. As a consequence, the information content of the abstracted signal is reduced as well which can be expressed by way of the Shannon Entropy.

Let $V = \{v_1, v_2, \ldots, v_N\}$ be a set of symbols with $v_1 = \mathfrak{a}$ and $v_2 = \mathfrak{b}$, let $V_A = \{\mathfrak{A}, v_3, \ldots, v_N\}$ be another set of symbols with $N-1$ elements, let p_i be the probability of occurrence of v_i with $p_1 = p_\mathfrak{a}$ and $p_2 = p_\mathfrak{b}$, and $p_\mathfrak{A}$. Further, let s be a signal of length m and let $s_A = B_{\alpha_v}(s)$ be the abstracted signal of equal length. The Shannon entropy of these two signals is

$$H(s) = -\sum_{i=1}^{N} p_i \log p_i = -p_\mathfrak{a} \log p_\mathfrak{a} - p_\mathfrak{b} \log p_\mathfrak{b} - \sum_{i=3}^{N} p_i \log p_i$$

$$H(s_A) = -p_\mathfrak{A} \log p_\mathfrak{A} - \sum_{i=3}^{N} p_i \log p_i$$

Since the last sum is identical in both expressions and since $p_\mathfrak{A} = p_\mathfrak{a} + p_\mathfrak{b}$ we have as entropy difference of these two signals

$$H_\delta = H(s) - H(s_A) = p_\mathfrak{a} \log \frac{p_\mathfrak{a} + p_\mathfrak{b}}{p_\mathfrak{a}} + p_\mathfrak{b} \log \frac{p_\mathfrak{a} + p_\mathfrak{b}}{p_\mathfrak{b}}$$

The information content decreases on average by H_δ per symbol and it depends only on the probabilities of the two abstracted symbols $\mathfrak{a}$ and $\mathfrak{b}$. For the special case $p_\mathfrak{a} = p_\mathfrak{b} = p$ and the base 2 logarithm we have $H_\delta = 2p$.

3.2.2 Time Abstraction

Let the time abstraction function α_t be

$$\alpha_t(\langle x_1, x_2\rangle) = \begin{cases} \mathfrak{A} & \text{if } x_1 = \mathfrak{a} \text{ and } x_2 = \mathfrak{a} \\ \langle x_1, x_2\rangle & \text{otherwise} \end{cases}$$

B_{α_t} maps two consecutive occurrences of $\mathfrak{a}$ onto $\mathfrak{A}$ and leaves all other symbols unchanged. We assume that all symbols $\mathfrak{a}$ appear in pairs, thus all $\mathfrak{a}$'s are replaced by $\mathfrak{A}$'s. This is not a simplification since we can pick an arbitrary pair of symbols, say $\langle \mathfrak{b}, \mathfrak{c}\rangle$ and first apply a value abstraction to transform it into $\langle \mathfrak{a}, \mathfrak{a}\rangle$ pairs, and then apply the time abstraction function α_t. The key point is that α_t shortens the signal by replacing all pairs of symbols $\langle \mathfrak{a}, \mathfrak{a}\rangle$ by a new symbol $\mathfrak{A}$.

α_t reduces the signal length but not the number of symbols and not necessarily the information per symbol. Thus, the reduction of information content comes from the decreasing signal length. While the general case is quite involved, we can illustrate the trend with a special case. Assume an abstraction function

$$\alpha_t(\langle x_1, x_2\rangle) = \begin{cases} \mathfrak{A} & \text{if } x_1 = \mathfrak{a} \text{ and } x_2 = \mathfrak{a} \\ \mathfrak{B} & \text{if } x_1 = \mathfrak{b} \text{ and } x_2 = \mathfrak{b} \\ \mathfrak{C} & \text{if } x_1 = \mathfrak{c} \text{ and } x_2 = \mathfrak{c} \\ \dots \end{cases}$$

Assume further that s consists only of symbol pairs like $s = \langle \mathfrak{a}, \mathfrak{a}, \mathfrak{c}, \mathfrak{c}, \mathfrak{a}, \mathfrak{a}, \mathfrak{b}, \mathfrak{b}, \mathfrak{a}, \mathfrak{a}, \mathfrak{d}, \mathfrak{d}, \mathfrak{b}, \mathfrak{b}, \dots\rangle$. The abstraction $B_{\alpha_t}(s)$ will then half the length of s but probabilities will be maintained like $p_\mathfrak{a} = p_\mathfrak{A}$, $p_\mathfrak{b} = p_\mathfrak{B}$, $p_\mathfrak{c} = p_\mathfrak{C}$, etc. Thus, the Shannon Entropy for s and $s_A = B_{\alpha_t}(s)$ is

$$H(s) = -\sum_{i\in\{\mathfrak{a},\mathfrak{b},\mathfrak{c},\dots\}} p_i \log p_i$$

$$H(s_A) = -\sum_{i\in\{\mathfrak{A},\mathfrak{A},\mathfrak{C},\dots\}} p_i \log p_i = H(s)$$

Hence, the Shannon Entropy denotes the average information content per symbol, which is unchanged. However, the information content of the entire signal is as follows.

$$I(s) = mH(s)$$

$$I(s_A) = \frac{m}{2} H(s_A) = \frac{I(s)}{2}$$

if the length of s is m and the lengths of s_A is $m/2$ as a result of the abstraction.

Time abstraction has its name because signals encode timing information. This means that merging two consecutive symbols into one decreases the number of symbols per time. Hence, timing abstraction reduces the information content per time unit.

Reducing the amount of information is a necessary condition for an abstraction but it is not sufficient for a useful abstraction. A useful abstraction will reduce the information that is less relevant and keep the important information, thus increasing its prominence. Much could be said about finding good abstractions, see for instance [30] for effective abstraction techniques. Also note, that what constitutes a useful abstraction depends on the actor's goals and condition.

3.3 Abstractions as Actors

Signal abstraction is modeled as an actor that maps one or more input signals onto output signals. Let $B = \langle \mathfrak{T}, I, O, z_0, \nu, f, g, \vec{\mathfrak{m}}_0 \rangle$ with $\mathfrak{T} = \{\epsilon\}$ (the actor is stateless), $I = \{\texttt{Signal}(V_1), \texttt{Signal}(V_2), \dots\}, O = \{\texttt{Signal}(V')\}$ (one or more input and one output signal), $z_0 = \epsilon$, (No initial state), $\nu(.) = \langle c_1, c_2, \dots \rangle$ (the actor consumes a constant number of values from each input signal), $g(\cdot,\cdot) = \epsilon$ (no states), $\vec{\mathfrak{m}}_0(\cdot) = \texttt{nop}$ (no meta actions). B maps one or more input signals consisting of symbols from $V_1, V_2, \dots$ onto one output signal with symbols from V', and if it applies a combination of value and time abstractions, we call it a signal abstraction. We can of course conceive more complex abstractions with internal states, but this kind of abstraction will suffice to illustrate the approach.

Given three actors A, B_I, B_O an **actor abstraction** $\texttt{ActAbstraction}(A, B_I, B_O) = A_A$ denotes an abstraction of actor A, if

$$B_O(A(S_I)) = A_A(B_I(S_I))$$

for all set of input signals S_I that can be consumed by A.

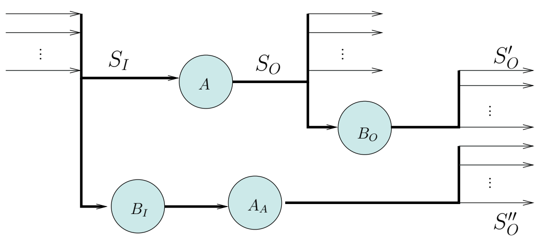


Figure 1 A_A is an abstraction of A if $S'_O = S''_O$.

This situation is depicted in figure 1. A_A operates on an abstraction of the input signals S_I, abstracted by the actor B_I. If the output signals S''_O generated by A_A are identical to the signals S''_O, which are abstractions of S_O, then actor A_A is an abstraction of actor A. This definition is not constructive and does not tell us how to derive A_A, or B_I or B_O; nor does it tell us what a useful abstraction is. Intuitively A_A should be significantly simpler than A but should faithfully reflect relevant properties of A.

4 Self-Model

An actor with a self-model has an abstract model of its own behavior, an abstract model of the environment it interacts with, and the capability to simulate these abstract models together.

Let A be an atomic or compound actor (as defined in section 2.3) arbitrarily complex actor, let B_I and B_O be abstractors of the input and output of A, respectively, and let $\texttt{ActAbstraction}(A, B_I, B_O) = A_A$, just as discussed in section 3.3. Further, let E be the environment the actor interacts with through the signal sets S_I and S_O, and let E_A be an abstraction of E, such that we have $\texttt{ActAbstraction}(E, B_O, B_I) = E_A$.

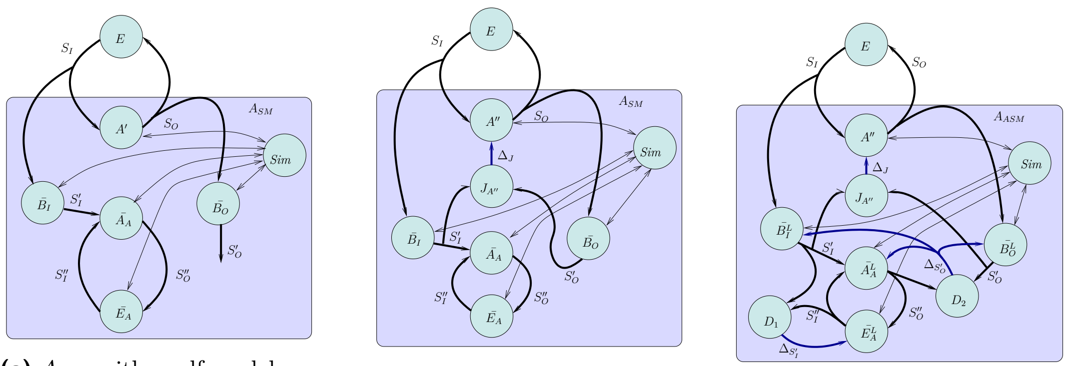
Moreover, let $\bar{A}$ be a *simulatable actor* derived from A which behaves like A with the following additions:

- It has an additional input signal denoted as *control signal*.
- It can be stopped and resumed at will through the control signal.

- For each input signal of A it has two input signals of the same type; hence it has two sets of input signals with identical types. The control signal selects one of the two sets for input in each activation cycle.
- It has an additional output signal, denoted as *status signal* that reports its internal status under control of the control signal.

The whole situation is illustrated in figure 2a. In addition we see in the figure a *Sim* actor, which controls the models $\bar{A}_A$, $\bar{E}_A$, $\bar{B}_I$ and $\bar{B}_O$ to simulate them. Also, instead of actor A we have a modified actor A' which acts just like A but can at appropriate times invoke the simulator, learn about simulation results, which then can support its decision process. Thus, given appropriate actors $A, \bar{A}_A, \bar{E}_A, \bar{B}_I, \bar{B}_O$ and *Sim*, $\texttt{ActorSelfModel}(A, \bar{A}_A, \bar{E}_A, \bar{B}_I, \bar{B}_O) = A_{SM}$ is a resulting actor corresponding to the one shown in figure 2a.

4.1 Self-Assessment



(a) A_{SM} with a self-model can simulate its own behavior together with an abstract model of the environment.

(b) An actor $J_{A''}$ continually monitors and assesses the behaviour and performance of A''.

(c) Adaptive actors require learning capabilities and error signals.

Figure 2 A self modeling actor A_{ASM}.

To allow for self-assessment the actor requires a model of the specification and requirements of itself. Such a model can be an elaborate functional model, or it can be a list of properties that at all times have to be fulfilled. A large body of literature has studied this problem under terms such as run-time monitoring, fault tolerance, and reliability. Thus, we assume solutions readily exist and a dedicated actor, named $J_{A''}$, continually monitors the input and output signals of the actor under observation and detects functional and performance aberrations. We could connect $J_{A''}$ to the actor inputs S_I and outputs S_O, however, it is more likely that $J_{A''}$ operates on abstractions of those signals like the one provided by $\bar{B}_I$ and $\bar{B}_O$. The output of actor $J_{A''}$ in figure 2b is denoted as Δ_J and signifies the difference between expected and observed behavior. It is fed back to actor A'' to allow for the use of this information and improve its performance. Hence, we have a variation of the actor without that facility, which was named A'.

If $J_{A''}$ also maintains a history of the assessment, it facilitates a holistic lifetime self-assessment as a basis for hindsight analysis, self-explanation and self-improvement.

4.2 Adaptive Self-Model

To model adaptive actors we need to capture the notion of learning.[2] A *Learning Actor* A^L is an actor that takes an error signal as input, in addition to the other signals it needs for its operation, and modifies its behavior with the goal to minimize the error in the error signal. Hence, let A be an actor with input signals I_A and output signals O_A, the learning actor A^L has input signals $I_{A^I} = I_A \cup \{\sigma_\epsilon\}$ and output signals $O_{A^L} = O_A$, where σ_ϵ is an error signal that reflects the quality of A's performance in some way. It could be a simple numeric signal or it could be structured to detail which parts and aspects of A's behavior exhibits which quality.

Considering figure 2b, there are several parts that we would like to see continually improved, in particular the abstractions $\bar{B}_I$ and $\bar{B}_O$, and the abstract models $\bar{A}_A$ and $\bar{E}_A$. If we want to make them learning actors, we have to identify the source of the error information. While actors could use application specific information, obvious generic sources are the differences between signals S'_O and S''_O, and between signals S'_I and S''_I.

Consequently, we introduce actors that analyze the differences in two sets of signals to generate Δ signals that inform other actors about observed differences. Figure 2c shows two actors, D_1 and D_2 that analyze and compare signals S'_I, S''_I and signals S'_O, S''_O, respectively, to generate the signals $\Delta_{S'_I}$ and $\Delta_{S'_O}$. These Δ signals are then used by the learning actors $\bar{B}^L_I$, $\bar{B}^L_O$, $\bar{A}^L_A$ and $\bar{E}^L_A$ to improve their models and their behavior. Figure 2c shows one possible scenario but many other strategies are conceivable and other information sources can be utilized to improve learning actors. We imagine that the learning actors in figure 2c start with an initial, relatively crude model or behavior which then is continuously improved with the expectation that this continuous improvement eventually leads to far better models and behaviors for $\bar{B}^L_I$, $\bar{B}^L_O$, $\bar{A}^L_A$ and $\bar{E}^L_A$ than could possibly be accomplished with careful engineering at design time.

5 Recursive Self-Reflection

Our actor has been extended quite significantly as illustrated in figure 2c. Before moving on, let's step back and consider what we have done. We have added functionality to our original actor A twice, as indicated by the two ticks. We have added assessment and simulation facilities together with abstract models of the actor itself and the environment. This allows for improved behavior of the actor by using self-assessment information from $J_{A''}$ and by using predictions from Sim. In addition we have introduced learning capabilities for the abstractions. Thus, A'' is continually improving by three different means: self-assessment, simulation based prediction, improving abstract models.

As a result we have obtained the actor A_{ASM}, an *adaptive, self-modeling actor.* Is it self-aware? The abstract self-model, the simulation engine, the self-assessment and the learning capabilities are all ingredients of self-awareness but they are not self-awareness, just like flour, sugar, raisins, yeast are ingredients for a cake, but they are not yet the cake. In fact, A_{ASM} can be considered the cake, but we are not looking for the cake, we are looking for the *process* of baking. So far we have used the mechanisms abstraction, simulation,

[2] The meaning of terms "learning", "adaptation" and "optimization" overlap. Here we use the term "learning" as a basic capability of an actor to modify its own behavior based on an error signal. Depending on how this capability is used, the actor may be *self-optimizing*, when the behavior improves within the same environment, or *adaptive*, when its behavior appropriately changes as response to a changing environment, or both.

assessment, and learning deliberately to construct something which resembles self-awareness, but the result is not self-awareness because self-awareness is the process, not the result. We need a general method that uses those mechanisms and can be applied to any actor, not just A. In particular, it must also be applicable to itself.

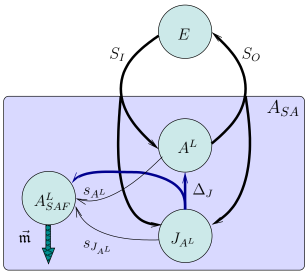
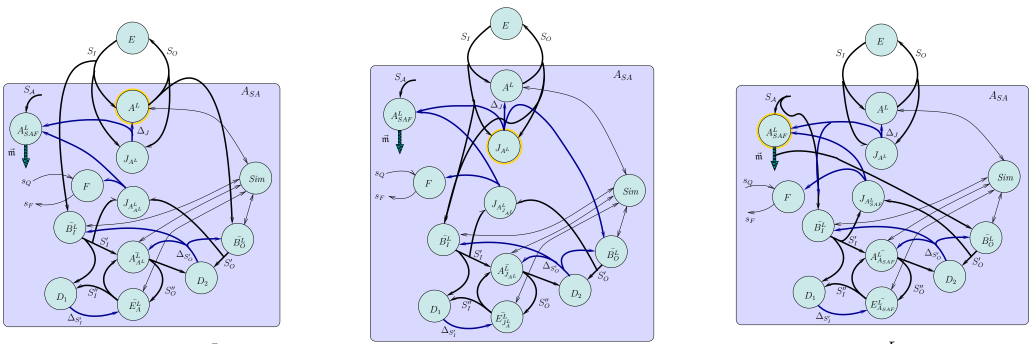


Figure 3 A self-awareness facilitating actor.

Consider figure 3, where a learning actor A^L interacts with the environment and is continuously monitored by J_{A^L}. Imagine the monitor J_{A^L} is more complex than checking properties. It keeps track of a set of goals that may be hierarchically organized and in part mutually contradictory. The goals could be to perform some useful function, to keep the battery loaded, avoid harming people, avoid damage to itself and to its environment, etc. The Δ_J signal informs to which extent these goals are satisfied at any time during operation.

In addition we have an actor A^L_{SAF} that facilitates self-awareness. It is informed by J_{A^L} about the actor's performance and, through the signals s_{A^L} and $s_{J_{A^L}}$ it keeps track of which actors are in the system. If it deems necessary, for instance when it is unhappy about the actor's performance, it can trigger an investigation. At its disposal it has simulation, abstraction, learning and other facilities. Picking an actor, for instance A_L, it can spawn a monitoring and assessment setup as illustrated in figure 4a. It does all this through meta actions through the $\vec{\mathfrak{m}}$ output in the figure. The self-awareness facilitator still keeps track of many, but not necessarily all, actors in the system, which is indicated by the $S_{\mathcal{A}}$ input signal. It can spawn a new investigation into any of the newly created actors if deemed useful and if the available resources suffice.



(a) Target is actor A^L. **(b)** Target is actor J_{A^L}. **(c)** Target is A^L_{SAF}.

Figure 4 A self-awareness facilitating actor A^L_{SAF} targeting other actors for study.

In addition, we propose to also provide an explanation actor F that, through a question and explanation interface (signals s_Q and s_F), provides a mechanism to explain what has happened, which decisions have been taken, what observations have been made. We expect this actor to be useful in the interaction with other systems. In particular in the interaction with humans it will convey to which extent the A_{SA} actor is self-aware and at what level it understands what it is doing.

For now, let's assume A^L_{SAF} deletes the newly created actors and returns to the state shown in figure 3, and then picks actor J_{A^L} as a next target for investigation, the result of which is shown in figure 4b. Moreover, it may target itself, if unhappy with its own performance or if just curious, and thus create a situation as shown in figure 4c.

In its simplest form the proposed self-reflection mechanism picks an actor, atomic or compound, abstracts this actor and assesses the behavior of the abstracted actor by comparing it to the actor's stated goals. Hence, a prerequisite for this operation is the accessibility of stated goals, which may be part of the actor under study or may come from somewhere else. From that it follows, that A^L_{SAF} can study any other actor for which it has access to its inputs, outputs and goals. This may in principle be the case for any of the actors visible in figures 4a-4c. But note, that not every interior detail of an actor is necessarily subject to this mechanism, since it is limited to behavior visible from the outside.

To warrant the name "recursive" the mechanism must be applicable to itself and to thus recursively derived actors, without principle limits. Consequently, any actor created by A^L_{ASF} in figure 4 could have its inputs, outputs and goals again be accessible to A^L_{ASF}. Without working out the details here this is plausible for all created actors because A^L_{ASF} generates the inputs and outputs itself and "knows" what it is supposed to accomplish.

In summary, we define self-awareness as the capability to pick any actor in the system, it may be a simple or compound actor or the entire system itself, and apply abstraction, assessment, prediction, and learning techniques, as outlined in this article, in order to analyze, assess and possibly improve its performance.

6 Related Work

As alluded to in the introduction a substantial amount of papers have been published on the topic of self-awareness. Here we only compare our proposal to definitions of self-awareness that have similar scope and ambition.

In 2009 Agarwal et al. [1] argue that self-aware subjects should be "*introspective*" (they can observe and optimise their own behaviour), "*adaptive*", "*self-healing*", (they monitor themselves for faults and take corrective actions), "*goal oriented*", and "*approximate*", (they use the least amount of precision to accomplish a given task).

In 2011 Lewis et al. [19] base their concepts on work in psychology, in particular on Morin's definition of self-awareness as "*the capacity to become the object of one's own attention*" [22] and Neisser's five-level model [24] which includes the "*ecological self*", the "*interpersonal self*", the "*extended self*", the "*private self*" and the "*conceptual self*", the last being "*the most advanced form of self-awareness, representing that the organism is capable of constructing and reasoning about an abstract symbolic representation of itself*" [19].

In 2014 Jantsch et al. [10] give seven properties that constitute awareness and define a subject to be aware at level 0 to 5, depending on which of these properties are exhibited by the subject. For instance level 4 requires that the subject assesses its own performance over the history of its lifetime, and can simulate future actions for prediction and planning purposes. The highest level 5 defines group awareness which requires subjects to be aware of its peers in a group.

We have cited the 2017 definition by Kounev et al. [11] in the introduction and repeat here only that it requires a subject to be self-reflective, self-predictive and self-adaptive.

All these definitions have some concepts in common, like goal orientation, adaptation, and introspection, but also differ in whether they include self-healing, approximation, learning, or prediction. But note that a definition that does not include an aspect such as learning probably does not mean to exclude it either. What is mentioned explicitly may only reflect the prominence given to some of the aspects, while others are less emphasized. These ambiguities and imprecision are a consequence of the informal style used to describe rather than define the key concepts of self-awareness.

Hence the first main difference to the work cited above is our attempt to provide a formal semantic for the involved concepts thus avoiding ambiguities and imprecision. We admit, that this attempt in giving a formal semantic is not complete but we argue it is a first step that shows the contours of such a semantic and that suggests it can be given.

The work by Vassev and Hinchey [31] is a formal approach to model self-awareness based on knowledge representation. It captures knowledge the system has about itself that includes information, rules, constraints and methods. The formal model has the benefit of clarity and unambiguity which makes clear that awareness is reduced to knowledge representation. In the described case study this self-knowledge is used by robots in a swarm to make situation dependent decisions that sensibly contribute to an overall swarm behavior. However, a mechanism to observe, assess and reason about its own usage of self-knowledge is missing.

Hence, the second main difference is the concept of recursive reflection. No other previous definition or model allows for applying self-awareness recursively onto its own activity. However, we contend that this unbounded recursion is the essence of self-awareness and it requires a formal model to demonstrate its feasibility and its utility.

7 Conclusions

The proposed formal model of self-awareness is based on a dynamic dataflow semantics. It captures the notion of signal abstraction, actor abstraction, adaptive actors, self-assessment, and recursive self-reflection. Even though many details of the formalism are still missing and the approach has not yet been demonstrated we are hopeful that it can be implemented and simulated in an appropriate framework.

A particular appealing aspect of recursive self-reflection is its promise, that any particular situation can be abstracted up to a level, where it is amenable to the assessment and planning capabilities of the system. Thus, there is no situation too complex that the self-aware actor is able to handle, provided it finds the appropriate sequence of abstractions. Since an abstraction step reduces the amount of information and since abstractions can be recursively applied, a given situation can be abstracted up to the level, where its information amount is within the limit of the system. The human mind seems to be doing something similar, because it manages to analyze, elaborate, and handle arbitrarily complex subjects even though the amount of conscious information processing is severely limited as has been established in Miller's seminal paper in 1956 on the magical number seven [21], and confirmed many times since then. If this analogy is correct, and if sufficiently effective and efficient abstraction techniques can be developed and employed, recursive self-reflection would turn out to be a wonderfully general tool for dealing with arbitrary situations where assessment and planning is crucial but an overwhelming diversity and complexity seems to render any general technique futile. These are big Ifs and a number of questions arise.

Abstraction techniques We need efficient techniques for automated abstraction. The definition of `ActAbstraction` is not constructive and there seems to be no good, general method to abstract an arbitrary actor. However, many abstraction methods exist but all of them have their strength and drawbacks. Thus, we need to identify good abstraction methods for our purpose and we need methods to select the most appropriate for a specific actor and for specific objectives.

Abstraction level Related to the abstraction method is the question of the right abstraction level. A given set of data and a given abstractor can be abstracted more or less. It is not well understood what constitutes a good abstraction level in general, and how to identify a good abstraction level in a particular case.

Assessment techniques We need good assessment techniques. Again, we do not have good general methods for assessment of an arbitrary actor.

Goal Management Complex systems often have a complex goal structure, which may be hierarchical and dynamic with partially overlapping and partially mutually exclusive goals. Handling these goals and assessing an actor's performance with respect to given goals is an interesting challenge.

Learning Machine learning is an active research domain and many methods have been proposed and studied. The challenge for us is to identify appropriate and efficient learning methods streamlined for our purpose.

Simulation Finally, general and efficient simulation methods will be instrumental to make self-awareness as proposed efficient. The key here is probably not the simulation method itself, but to find the right abstraction level in combination with efficient simulation methods.

With a precise, formal and operational model of self-awareness we can identify its challenges, address the open problems and study its benefits and drawbacks in the context of specific applications. As a result, self-awareness could be made into a powerful generic method that can be the foundation of truly autonomous systems.

References

1. A. Agarwal, J. Miller, J. Eastep, D. Wentziaff, and H. Kasture. Self-aware computing. *Final Technical Report AFRL-RI-RS-TR-2009-161*, page 81, 2009.
2. T. M. Cover and J.A. Thomas. *Elements of Information Theory*. Wiley, Hoboken, NJ, USA, 2006.
3. Lukas Esterle and Radu Grosu. Cyber-physical systems: challenge of the 21st century. *e & i Elektrotechnik und Informationstechnik*, 133(7):299–303, Nov 2016.
4. European Commission. Self-Awareness in Autonomic Systems, Jan 2013.
5. Radu Grosu and Ketil Stølen. A denotational model for mobile point-to-point dataflow networks. Technical Report Technical Report SFB 342/14/95 A, Technische Universität München, 1995.
6. Radu Grosu and Ketil Stølen. A model for mobile point-to-point data-flow networks without channel sharing. In *In Proc. AMAST'96, LNCS*, pages 504–519, 1996.
7. Axel Jantsch. *Modeling Embedded Systems and SoCs - Concurrency and Time in Models of Computation*. Systems on Silicon. Morgan Kaufmann Publishers, June 2003.
8. Axel Jantsch. Models of embedded computation. In Richard Zurawski, editor, *Embedded Systems Handbook*. CRC Press, 2005. Invited contribution.
9. Axel Jantsch, Nikil Dutt, and Amir M. Rahmani. Self-awareness in systems on chip – a survey. *IEEE Design Test*, 34(6):1–19, December 2017.
10. Axel Jantsch and Kalle Tammemäe. A framework of awareness for artificial subjects. In *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis*, CODES '14, pages 20:1–20:3, New York, NY, USA, 2014. ACM.

11 S. Kounev, J.O. Kephart, A. Milenkoski, and X. Zhu, editors. *Self-Aware Computing Systems.* Springer, 2017.

12 Samuel Kounev, Peter Lewis, Kirstie Bellman, Nelly Bencomo, Javier Camara, Ada Diaconescu, Lukas Esterle, Kurt Geihs, Holger Giese, Sebastian Götz, Paola Inverardi, Jeffrey Kephart, and Andrea Zisman. The notion of self-aware computing. In Samuel Kounev, Jeffrey O. Kephart, Aleksandar Milenkoski, and Xiaoyun Zhu, editors, *Self-Aware Computing Systems*, pages 3–16. Springer, 2017.

13 Samuel Kounev, Xiaoyun Zhu, Jeffrey O. Kephart, and Marta Kwiatkowska. Model-driven Algorithms and Architectures for Self-Aware Computing Systems (Dagstuhl Seminar 15041). *Dagstuhl Reports*, 5(1):164–196, 2015.

14 Edward A. Lee. A denotational semantics for dataflow with firing. Technical Report UCB/ERL M97/3, Department of Electrical Engineering and Computer Science, University of California, Berkeley, January 1997.

15 Edward A. Lee and Alberto Sangiovanni-Vincentelli. A framework for comparing models of computation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(12):1217–1229, December 1998.

16 Peter R. Lewis. Self-aware computing systems: From psychology to engineering. In *Proceedings of the 2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1044–1049, 2017.

17 Peter R. Lewis, Arjun Chandra, Funmilade Faniyi, Kyrre Glette, Tao Chen, Rami Bahsoon, Jim Torresen, and Xin Yao. Architectural aspects of self-aware and self-expressive computing systems. *IEEE Computer*, August 2015.

18 Peter R. Lewis, Marco Platzner, Bernhard Rinner, Jim Torresen, and Xin Yao, editors. *Self-Aware Computing Systems: An Engineering Approach.* Springer, 2016.

19 P.R. Lewis, A. Chandra, S. Parsons, E. Robinson, K. Glette, R. Bahsoon, J. Torresen, and Xin Yao. A survey of self-awareness and its application in computing systems. In *Self-Adaptive and Self-Organizing Systems Workshops (SASO), 2011 Fifth IEEE Conference on*, pages 102–107, October 2011.

20 Yue Li and Asok Ray. Unsupervised symbolization of signal time series for extraction of the embedded information. *Entropy*, 19(4), 1 2017.

21 George A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63:81–97, 1956.

22 Alain Morin. Levels of consciousness and self-awareness: A comparison and integration of various neurocognitive views. *Consciousness and Cognition*, 15(2):358 – 371, 2006.

23 Christian Müller-Schloer, Hartmut Schmeck, and Theo Ungerer, editors. *Organic Computing — A Paradigm Shift for Complex Systems.* Birkhauser, 2011.

24 Ulric Neisser. The roots of self-knowledge: Perceiving self, it, and thou. *Annals of the New York Academy of Sciences*, 818:19–33, 1997.

25 L.D. Paulson. DARPA creating self-aware computing. *IEEE Computer*, 36(3):24, March 2003.

26 Jeremy Pitt, editor. *The Computer after Me : Awareness and Self-Awareness in Autonomic Systems.* Imperial College Press, 2014.

27 J. Schaumeier, J. Jeremy Pitt, and G. Cabri. A tripartite analytic framework for characterising awareness and self-awareness in autonomic systems research. In *Proceedings of the Sixth IEEE Conference on Self-Adaptive and Self-Organizing Systems Workshops (SASOW)*, pages 157–162. IEEE Computer Society, 2012.

28 P. Senin, J. Lin, X. Wang, T. Oates, S. Gandhi, A.P. Boedihardjo, C. Chen, S. Frankenstein, and M. Lerner. GrammarViz 2.0: a tool for grammar-based pattern discovery in time series. In *ECML/PKDD*, 2014.

29 R. Sterritt and M. Hinchey. SPAACE IV: Self-properties for an autonomous and autonomic computing environment - part IV a newish hope. In *IEEE International Conference and Workshops on Engineering of Autonomic and Autonomous Systems (EASe)*, pages 119–125, 2010.

30 Joshua B. Tenenbaum, Charles Kemp, Thomas L. Griffiths, and Noah D. Goodman. How to grow a mind: Statistics, structure, and abstraction. *Science*, 331(6022):1279–1285, 2011.

31 E. Vassev and M. Hinchey. Knowledge representation and awareness in autonomic service-component ensembles - state of the art. In *2011 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, pages 110–119, March 2011.

32 Martin Wirsing, Matthias Hölzl, Nora Koch, and Philip Mayer, editors. *Software Engineering for Collective Autonomic Systems: The ASCENS Approach*. Springer, 2015.