The Next Step in the Evolution of Artificial DNA: the Abstract ADNA

Aleksey Koschowoj, Mathias Pacher and Uwe Brinkschulte

Goethe-University Frankfurt am Main, Computer Science Departement, Embedded Systems
Frankfurt am Main, Germany

Abstract
This paper describes the extension of the Artificial DNA (ADNA) towards the abstract ADNA (A^2DNA) and a knowledge base. It also presents and analyzes the determinator that specifies the A^2DNA to a hardware specific ADNA using the knowledge base. The ADNA represents a building plan that enables an embedded system to build and organize itself. In its current state the ADNA demands high amounts of knowledge on the target hardware's available sensors already in the design phase. Thus, it becomes rigid and cannot be adapted easily to change in the hardware. The A^2DNA solves this problem by describing demands about the required sensors using so called abstract sensors instead of being tied to a specific sensor like the ADNA. The determinator then specifies the A^2DNA at the build and initialization phase directly on the hardware into an ADNA. To achieve this it uses a semantic knowledge base which provides both descriptions of the hardware's available sensors and knowledge on their relations in form of equations.

Keywords: Artificial DNA • Semantic Knowledge • Organic Computing • Virtual Sensors

1. Introduction

Real-time embedded systems equipped with sensors and actuators are becoming more and more commonplace. This leads to an expansion of their areas for application, which in turn demand more complex systems. As a means to study these systems, the research field of Organic Computing has been established by [1]. Organic Systems are characterized by self-* properties described in [2], such as self-organization, self-configuration, self-improvement and self-healing. An exemplary system for these self-* properties is the Artificial Hormone System (AHS) with the Artificial DNA (ADNA) developed by [3]. While the ADNA is a fine approach, it demands knowledge on the target hardware, like sensors and actuators, already at design time. Therefore, the ADNA is strongly influenced by the hardware without providing much information on the hardware itself. In this paper, we present our contribution to ongoing research on the AHS and ADNA, the enhancement of the ADNA with a semantic knowledge base on sensors, actuators and their interdependencies towards the abstract ADNA (A^2DNA). Further, we present and analyze an algorithm that specifies A^2DNA into a system specific ADNA when initializing the AHS on the target system.

koschowoj@es.cs.uni-frankfurt.de (A. Koschowoj)

The paper is structured as follows: First, we provide a brief overview of the ADNA's current state. Second, we describe the *A^2DNA*'s core ideas. Next, we present and analyze the determinator's algorithm. Finally, we discuss the *A^2DNA* and our results and provide an outlook into the *A^2DNA*'s future developments.

2. State of the ADNA

The AHS, as described in [4] and [3], is a decentralized, self-organizing, self-configuring, self-improving and self-healing mechanism that assigns task to *processing elements* (PEs) in embedded real-time systems. Inspired by the hormone system of higher mammals where cells communicate by sending hormones via the bloodstream, the PEs send short messages, so-called *hormones*, to communicate and decide the task assignment among themselves. On its own, the AHS has no knowledge of the system it has to realize. Any knowledge including the required tasks, their communicative interconnections and the PEs' initial suitability for each task is provided by the ADNA. When a task is assigned to a PE, the PE derives the parametrization from its local copy of the ADNA. This process is shown in Figure 1. The ADNA is based on the observation that most embedded systems can be assembled from a limited number of basic elements, e.g. sensors, actuators, arithmetic/logic units, etc. Thus, it is possible to compose a given embedded system by combing a sufficient multiset of these elements and providing a fitting parameterization for each element.

When designing an ADNA, the available sensors and actuators in the used hardware strongly influence the ADNA's final shape. As an example consider the control loop in Fig. 2 and its netlist in Fig. 3. In this description, the sensor/actuator building blocks are mapped to specific sensors/actuators on the hardware. Not only the available sensors/actuators have to be known at this design stage, any change on the hardware, even reindexing, may result in an ADNA incompatible to the hardware. Thus, the ADNA cannot be used on another hardware without an external adaptation, making it rigid and bound to hardware.

3. Extension to the *A^2DNA*

As a solution to the ADNA's rigidness, we propose the *abstract ADNA* (*A^2DNA*). On the one hand, the *A^2DNA* aims loosen the bindings between a specific ADNA[1] and the hardware. On the other hand, it should provide more knowledge on the sensors[2] and their attributes' relations such that the ADNA may be specified only when initializing the system.

Figure 4 shows the process of specifying an *A^2DNA* for a given hardware. The *A^2DNA* demands sensors in order to be used on the hardware while hardware knowledge is provided by the description of the available sensors. Both use so called *abstract sensors* to describe their sensors. A determinator specifies the *A^2DNA* for the described hardware, resulting in an ADNA, using its knowledge on the relations between the abstract sensor's attributes. The knowledge about the relations is provided in form of equations. Both the sensor description and equations form the knowledge base.

[1]For simplicity's sake we keep referring to it as ADNA.
[2]In the following, we mainly focus on sensors to provide a basic understanding of the *A^2DNA*.

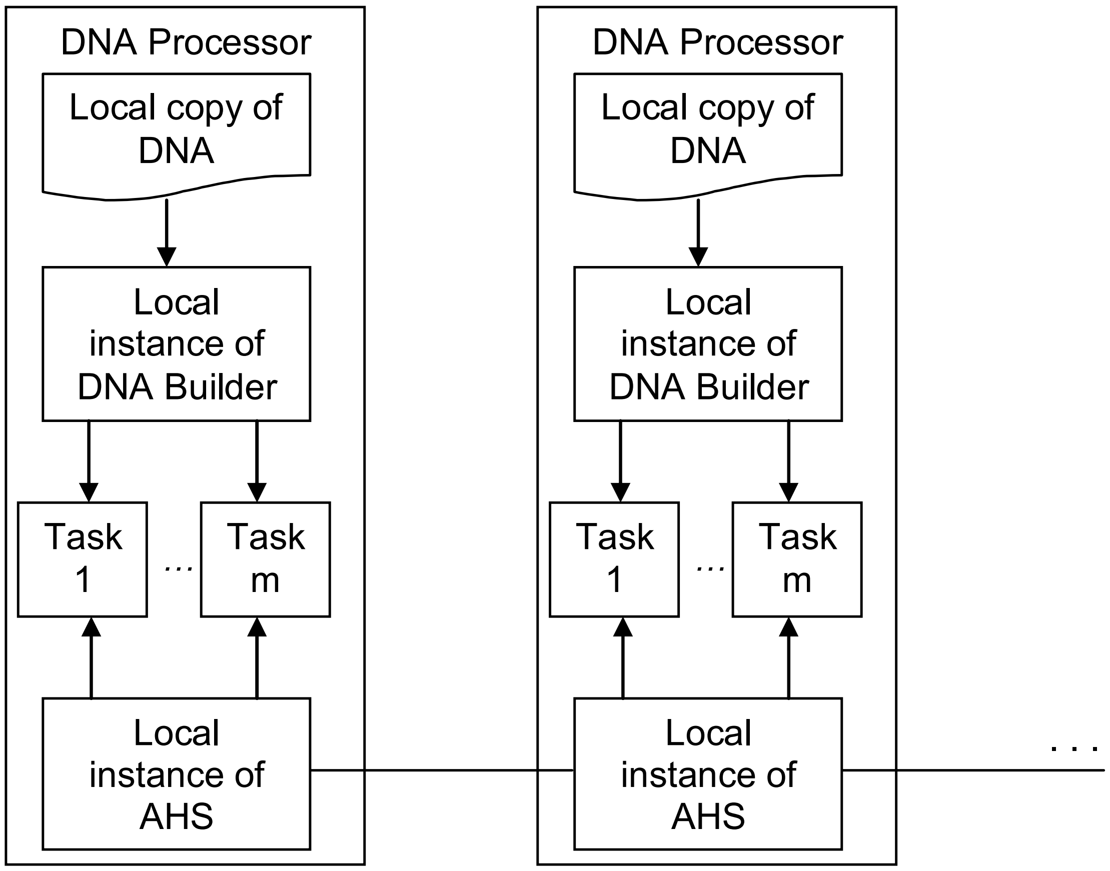


Figure 1: ADNA architecture, from [3]

3.1. Abstracting the Sensors

In contrast to the ADNA where a sensor is described by a sensor building block's *Resource ID* parameter, a sensor in an A^2DNA is described by an abstract sensor building block listing the required attributes.

Inspired by the ADNAs described in [3], we use the following three attributes for the abstract sensors. Firstly, we distinguish sensors form each other by the physical quantities $\mathcal{Q}$ they measure. Secondly, since some quantities are vectors we have to take into account the sensor's relative direction $\mathcal{D}$. As a naming convention, we choose the local frame described by [6], an example is shown in Figure 5. Finally, we have to differentiate between the sensor's target $\mathcal{T}$, i.e. does the sensor provide data about the system, the surroundings or only a part of either. Given that, each attribute provides different information, the sets are disjoint. The abstract sensor block structure and a parameterization are shown in Figure 6.

Definition 1 (Abstract sensor). *An abstract sensor* $s = (q_s, d_s, t_s) \in \mathcal{Q} \times \mathcal{D} \times \mathcal{T}$ *is defined as triple of measured physical quantity* $q_s \in \mathcal{Q}$*, the measurement's direction* $d_s \in \mathcal{D}$ *and measurement's target* $t_s \in \mathcal{T}$*.*

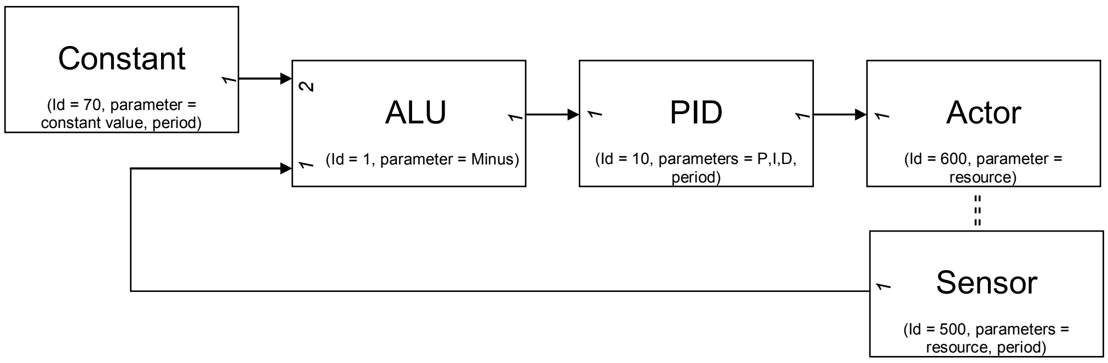

Constant
(Id = 70, parameter = constant value, period)
ALU
(Id = 1, parameter = Minus)
PID
(Id = 10, parameters = P,I,D, period)
Actor
(Id = 600, parameter = resource)
Sensor
(Id = 500, parameters = resource, period)
1
2
1
1
1
1
1
1
1

Figure 2: A closed control loop consisting of basic elements, from [5]

```
1   = 70  (1:2.2) 100 25    // constant setpoint value, period 25 msec
2   = 1   (1:3.1) -         // ALU, control deviation (minus)
3   = 10  (1:4.1) 4 5 6 25  // PID (4, 5, 6), period 25 msec
4   = 600         1         // actor, resource id = 1
5   = 500 (1:2.1) 2 25      // sensor, resource id = 2, period 25 msec

ADNA line:       linenumber = id destinationlink parameters //comment
destinationlink: (destchannel:destlinenumber.sourcechannel...)
```

Figure 3: The netlist and parameterization of the closed control loop from 2, from [5]

Example 1. *The system (car) shown in Figure 5 may have the abstract sensors (velocity, x-direction, car) and (velocity, y-direction, car) that describe real sensors measuring the car's velocity in direction of the x and y axis respectively. But our* $\mathrm{A}^2\mathrm{DNA}$ *in Figure 6 requires the abstract sensor (velocity, xy-direction, car).*

3.2. Knowledge base

So far, the described attributes lack any meaning or use for the determinator since it has no knowledge on any given connections between the attribute's values. This knowledge is provided to the determinator through equations and serves as the backbone of its knowledge base. These equations include relations, e.g. velocity being the temporal derivation of traveled distance or that the velocities projected onto the x and y axes enable the calculation of the velocity projected onto the xy axis. With such knowledge, the determinator can infer what sensors are implicitly given on the hardware.

Definition 2 (Equation). *Let $\{k_0, k_1, \ldots, k_j\}$ be sub set of either $\mathcal{Q}$ or $\mathcal{D}$ or $\mathcal{T}$ with $j \geq 1$. An equation $eq := k_0 = \mathrm{OP}_{i=1}^{j}\, k_i$ describes the relation between an attribute k_0 and the attributes $k_1, \ldots, k_j$. The j-nary operator OP denotes a specific building block*[3] *that derives the value of the attribute k_0 from the values of the attributes $k_1, \ldots, k_j$.*

[3]The block's exact structure must not be known in the equation, just what block or set of blocks will be needed.

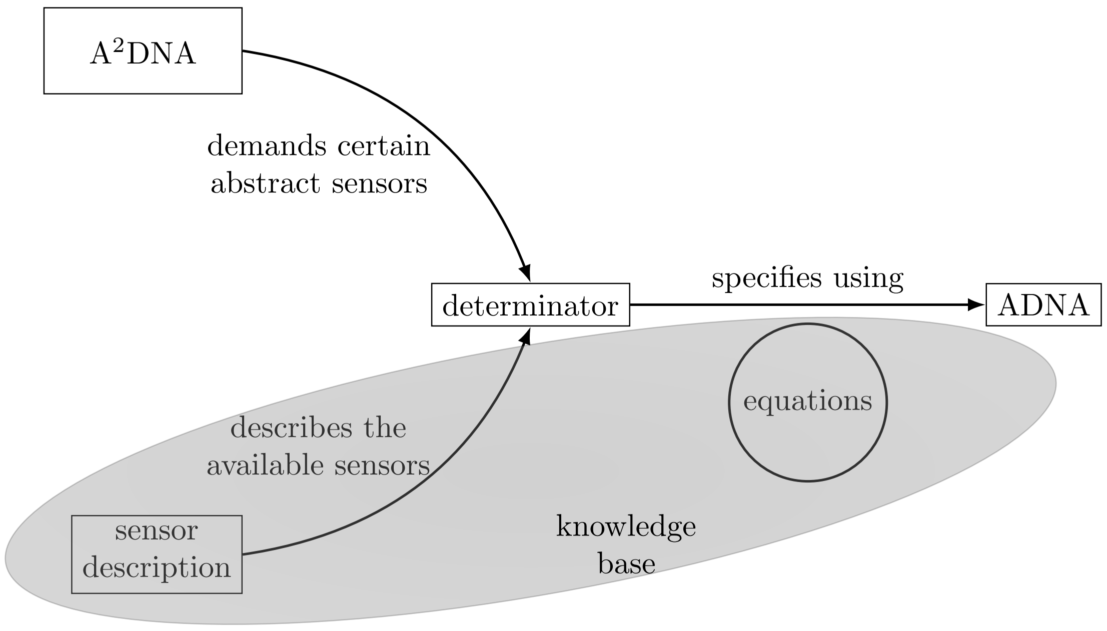


Figure 4: Specification process

Example 2 (Equation). *To our car system in Example 1 we can convey the relation*[4] *between the required xy-direction and available x- and y-directions with the equation:*

$$xy\text{-direction} = \frac{x\text{-direction} + y\text{-direction}}{\sqrt{2}} \tag{1}$$

For example consider that our sensors measure the car's velocity in x-and y-direction as 1 and 3 $^{m}/_{s}$ respectively. Then, a sensor in xy-direction would measure a velocity of $2\sqrt{2}$ $^{m}/_{s}$.
In the next examples, we abbreviate the operation $\frac{a+b}{\sqrt{2}}$ to $\mathrm{OP}_{xy}(a, b)$.

Therefore, we can define a knowledge base as follows:

Definition 3 (Knowledge base). *A knowledge base $\mathcal{K} = (\mathcal{S}, \mathcal{E})$ consists of a set of available sensors on the hardware $\mathcal{S}$ and a set of known relations (equations) $\mathcal{E}$.*

Example 3 (Knowledge base). *Combining the abstract sensors and equation from the Examples 1 and 2, the car system shown in Figure 5 may have a simple knowledge base*[5]*:*

$$\mathcal{K} \qquad = \qquad (\underbrace{\{(v, x, c), (v, y, c)\}}_{\mathcal{S}}, \underbrace{\{xy = OP_{xy}(x, y)\}}_{\mathcal{E}})$$

[4] While this relation is true for the any two projections onto orthogonal directions and their resulting diagonal direction, we limit it to x and y in the scope of our example.

[5] For the sake of readability, we applied the abbreviations velocity to v and car to c and dropped the -direction suffix.

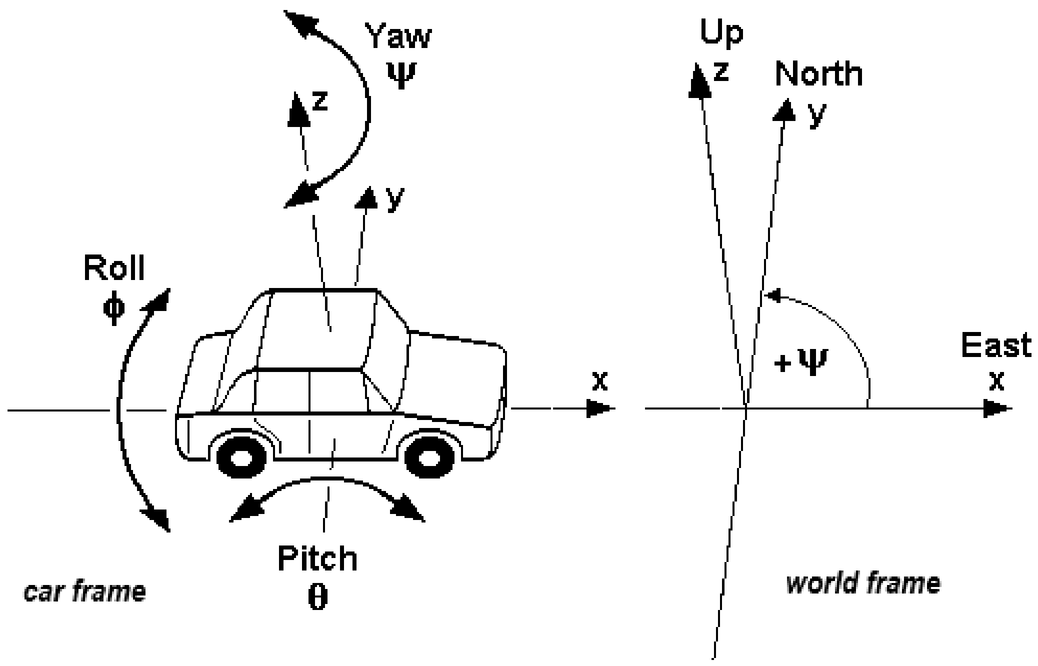


Figure 5: A system (car) with its own coordinate axes (car frame) and a world frame, adapted from [7]

Abstract Sensor
(Id = 599, parameters =
quantity, direction, target)

```
1 = 599 v xy car // abstract sensor,
 quantity = velocity, direction = xy, target = car
```

Figure 6: An abstract sensor block and as A^2DNA describing a car's velocity in xy-direction

3.3. Determinator

Utilizing the knowledge base $\mathcal{K}$, the determinator has now the tools to infer which abstract sensors in the A^2DNA can be determined.

3.4. Applying equations to sensors

First we have to define how the determinator infers new knowledge on the sensors.

Definition 4 (Applying equations). *Let S be a set of sensors and $eq := k_0 = \mathrm{OP}_{i=1}^{j} k_i$ an equation. We say that the set of sensors $S' = S \cup \{s\}$ results from applying the equation eq to the* *set S if we have one of the following cases*[6]*:*

- *If there exists a set $Z = \{(k_i, d, t) | 1 \leq i \leq j\} \subseteq S$, fixed by $k_1, \ldots, k_j$ in eq, then add $s = (k_0, d, t)$.*
- *If there exists a set $Z = \{(q, k_i, t) | 1 \leq i \leq j\} \subseteq S$, fixed by $k_1, \ldots, k_j$ in eq, then add $s = (q, k_0, t)$.*
- *If there exists a set $Z = \{(q, d, k_i) | 1 \leq i \leq j\} \subseteq S$, fixed by $k_1, \ldots, k_j$ in eq, then add $s = (q, d, k_0)$.*

An application is denoted as $s = eq(S)$.
We also say that applying eq to S yields s.

Remark 1. *When applying an equation two of the yielded sensor's attributes are defined by the sensors in Z and the remaining attribute is defined by the equation.*

Remark 2. *If multiple subsets Z fulfill the condition, then applying an equation eq on the same set S may yield different sensors s. Thus, we may have to apply eq once for every Z.*

Remark 3. *We may also apply an equation to S if for all Z that fulfill the condition, their yielded s is already in S.*

Example 4. *Recall the car system from Example 1 and its knowledge base from Example 3*

$$\mathcal{K} = (\{(v, x, c), (v, y, c)\}, \quad \{xy = OP_{xy}(x, y)\}).$$

Applying the equation $xy = \mathrm{OP}_{xy}(x, y)$ on the set $S = \{(v, x, c), (v, y, c)\}$ results in the set $S' = \{(v, x, c), (v, y, c), (v, xy, c)\}$, thus yielding the sensor $s = (v, xy, c)$. We have inferred the existence of a car velocity sensor in xy-direction, the sensor our $\mathrm{A}^2\mathrm{DNA}$ *required.*

This definition can be extended for multiple applications.

Definition 5 (Produces). *Let S_0 and S_m be sets of sensors. We say that S_0 produces S_m if for $m \geq 1$ exist equations $eq_0, \ldots, eq_{m-1}$ and sets of sensors $S_1, \ldots, S_{m-1}$, s.t. applying the equation eq_i to the set S_i results in the set S_{i+1}.*

With this definition, we can now define when an abstract sensor is determinable, i.e. the determinator can infer the existence of a specific composition for an abstract sensor utilizing the knowledge base $\mathcal{K}$.

Definition 6 (Determinable). *We say a sensor s is determinable in a given knowledge base $\mathcal{K} = (\mathcal{S}, \mathcal{E})$ iff $s \in \mathcal{S}$ or $\mathcal{S}$ produces a set S', s.t. $s \in S'$.*

[6]Since all attributes in an equation are from the same set, we only have this three cases.

Example 5. *Consider again the knowledge base*

$$\mathcal{K} = (\{(v, x, c), (v, y, c)\}, \quad \{xy = OP_{xy}(x, y)\}).$$

Both (v, x, c) and (v, y, c) are in $\mathcal{S}$, thus they are determinable in $\mathcal{K}$. In Example 4, we see how application of one equation yields the sensor (v, xy, c). Since $\mathcal{S}$ produces a set S' with $(v, xy, c) \in S'$, this sensor is also determinable in $\mathcal{K}$.

Since the determinator knows how to construct a determinable sensor, we can consider any determinable sensor part of the sensors available on the hardware. This provides us with another definition for a sensor's determinability.

Corollary 1 (Determinable). *Let $\mathcal{K} = (\mathcal{S}, \mathcal{E})$ be a knowledge base and $s \notin \mathcal{S}$ a sensor. If there are a set of sensors Z and an equation $eq \in \mathcal{E}$, s.t. $s = eq(Z)$ and $\forall z \in Z$: z is determinable in $\mathcal{K}$, then s is determinable in $\mathcal{K}$.*

Since all sensors in Z are determinable in $\mathcal{K}$, we know for each sensor $z \in Z$ which equations we have to apply to produce a set S', s.t. $z \in S'$. Applying all these equations sequentially on $\mathcal{S}$ results in $\mathcal{S}'$, s.t. $Z \subseteq \mathcal{S}'$. Since applying eq to Z yields s, applying eq to S' results in S'' with $s \in S''$, thus s is determinable in $\mathcal{K}$. □

Example 6. *Consider a knowledge base $\mathcal{K}' = (\mathcal{S}', \mathcal{E}')$ slightly different to the previous, where we still have $(v, x, c) \in \mathcal{S}'$ and $xy = OP_{xy}(x, y) \in \mathcal{E}$, but first have to determine (v, y, c). If we determine (v, y, c) in $\mathcal{K}'$, then, by Corollary 1 and Example 5, we know that we can also determine (v, xy, c).*

Finally, we can define the set of all sensors determinable in $\mathcal{K}$.

Definition 7 (All determinable sensors). *Let $\mathcal{K} = (\mathcal{S}, \mathcal{E})$ be a knowledge base. Let $\mathcal{P}_{\mathcal{K}}$ be set produced by $\mathcal{S}$, s.t. applying any equation to $\mathcal{P}_{\mathcal{K}}$ yields a sensor s already in $\mathcal{P}_{\mathcal{K}}$. Hence, every possible application of an equation in $\mathcal{E}$ to $\mathcal{P}_{\mathcal{K}}$ results in $\mathcal{P}_{\mathcal{K}}$.*
We refer to $\mathcal{P}_{\mathcal{K}}$ as the set of all determinable sensors in $\mathcal{K}$.

Example 7. *Consider again the knowledge base*

$$\mathcal{K} = (\{(v, x, c), (v, y, c)\}, \quad \{xy = OP_{xy}(x, y)\}).$$

From Example 4, we know that $\mathcal{S}$ produces the set $S' = \{(v, x, c), (v, y, c), (v, xy, c)\}$. $S' = \mathcal{P}_{\mathcal{K}}$ because applying our only equation $xy = OP_{xy}(x, y)$ to S' yields $(v, xy, c) \in S'$. Thus S' is the set of all determinable sensors in our small knowledge base $\mathcal{K}$.

4. Determinability algorithm

With the A^2DNA's general concepts defined, we can now focus on the determinators exact functionality. As a proof of concept, we describe a naive algorithm for checking the determinability of a set of sensors Z. This algorithm constructs the set of all determinable sensors $\mathcal{P}_{\mathcal{K}}$ (see Definition 7) inductively, then checks for each $s \in Z$ if it is also in $\mathcal{P}_{\mathcal{K}}$. By definition all sensors in $\mathcal{S}$ are determinable, thus we initialize $\mathcal{P}$ as $\mathcal{S}$. Now we try to apply each equation as often as it yields a new sensor not in $\mathcal{P}$ to $\mathcal{P}$. If at least one equation yielded a new sensor, then we have expanded $\mathcal{P}$ and need to retry our equations. If no equation yielded new sensors, then $\mathcal{P}$ has become $\mathcal{P}_{\mathcal{K}}$ and we can proceed to the next step. For each $s \in Z$, we check if $s \in \mathcal{P}_{\mathcal{K}}$. If at least one is not, we immediately return false. If all sensors were in $\mathcal{P}_{\mathcal{K}}$, we return true.

Before we can analyze Algorithm 1, we need to clarify how we check if eq is applicable to $\mathcal{P}$. By Remark 1, we only have to check for each possible pair of the two attributes not affected by the equation if there is a set of possible sensors that fulfills the condition. This leads to Algorithm 2 which returns for all pairs of the unaffected attributes' values if all sensors for the application of $k_0 = \mathrm{OP}_{i=1}^{j} k_i$ are in $\mathcal{P}$. For the description we focus without loss of generality[7] on the case $k_0 \in \mathcal{Q}$. First, we generate a hash map A where all pairs of $(d, t) \in \mathcal{D} \times \mathcal{T}$ serve as

```
Input: Knowledge base $\mathcal{K} = (\mathcal{S}, \mathcal{E})$ and a set of sensors $Z$
Output: Boolean
$\mathcal{P} = \mathcal{S}$;
found_new_sensors = true;
while found_new_sensors:
    found_new_sensors = false;
    foreach $eq \in \mathcal{E}$:
        while Applying $eq$ to $\mathcal{P}$ yields a new sensor:
            found_new_sensors = true;
            $\mathcal{P} = eq(\mathcal{P})$;
foreach $s \in Z$: if $s \notin \mathcal{P}$: return false;
return true;
```

Algorithm 1: Determinability check

keys. The values represent the number of sensors found. Second, we iterate through $\mathcal{P}$. On the one hand, if a sensor (k_i, d_s, t_s) is found, we increase the value of the pair (d_s, t_s) by one. If, on the other hand, the sensor (k_0, d_s, t_s) is found[8], the value of the pair (d_s, t_s) is reduced by one. Thus, only if for a fixed pair (d, t) all sensors $(k_1, d, t), \dots (k_j, d, t)$ are in $\mathcal{P}$ and (k_0, d, t) is not in $\mathcal{P}$, the entry $A[(d, t)]$ will have the value j and the equation will be applicable[9].

Theorem 1 (Complexity of Algorithm 2). *Algorithm 2 has a complexity of*

$$\mathcal{O}\Big((|\mathcal{P}| + |\mathcal{N}|)|\mathcal{N}|\Big)$$

with $|\mathcal{N}| = \max\{|\mathcal{Q}|, |\mathcal{D}|, |\mathcal{T}|\}$.

In each case, we generate a hash map with a size of at most $|\mathcal{N}|^2$ entries. This has a run time of $\mathcal{O}(|\mathcal{N}|^2)$. Next, we iterate through $\mathcal{P}$ and check for every entry if it has any k_i. Since, all

[7]All cases behave the same just focus on different pairs.
[8]Therefore we can no longer apply the equation.
[9]Note that all pairs in A where the value is exactly j have a set of sensors Z and the sensor s is not part of the set.

```
Input: An equation $k_0 = \mathrm{OP}_{i=1}^{j} k_i$ and a set of sensors $\mathcal{P}$
Output: All pairs where the equation is applicable
switch type($k_0$):
    case $k_0 \in \mathcal{Q}$:
        Generate a hash map $A$ with every pair $(d,t) \in \mathcal{D} \times \mathcal{T}$ as a key and initialize the
          values as 0;
        foreach $(q_s, d_s, t_s) \in \mathcal{P}$:
            if $q_s \in \{k_1, \dots, k_j\}$: $A[(d_s, t_s)] += 1$;
            elif $q_s == k_0$: $A[(d_s, t_s)] -= 1$;
    case $k_0 \in \mathcal{D}$:
        Generate a hash map $A$ with every pair $(q,t) \in \mathcal{Q} \times \mathcal{T}$ as a key and initialize the
          values as 0;
        foreach $(q_s, d_s, t_s) \in \mathcal{P}$:
            if $d_s \in \{k_1, \dots, k_j\}$: $A[(q_s, t_s)] += 1$;
            elif $d_s == k_0$: $A[(q_s, t_s)] -= 1$;
    case $k_0 \in \mathcal{T}$:
        Generate a hash map $A$ with every pair $(q,d) \in \mathcal{Q} \times \mathcal{D}$ as a key and initialize the
          values as 0;
        foreach $(q_s, d_s, t_s) \in \mathcal{P}$:
            if $t_s \in \{k_1, \dots, k_j\}$: $A[(q_s, d_s)] += 1$;
            elif $t_s == k_0$: $A[(q_s, d_s)] -= 1$;
return $A$;
```

Algorithm 2: Check if $k_0 = \mathrm{OP}_{i=1}^{j} k_i$ yields a new sensor

$k_0, k_1, \dots, k_j$ form a subset of either $\mathcal{Q}$, $\mathcal{D}$ or $\mathcal{T}$, we have to check at most $|\mathcal{N}|$ values. Thus, this step is at most $\mathcal{O}(|\mathcal{P}||\mathcal{N}|)$. Therefore, Algorithm 2 has a complexity of $\mathcal{O}((|\mathcal{P}| + |\mathcal{N}|)|\mathcal{N}|)$. $\square$ With this part analyzed, we can finally analyze Algorithm 1's complexity.

Theorem 2 (Complexity of Algorithm 1). *Algorithm 1 has a complexity of*

$$\mathcal{O}\Big(|\mathcal{P}_{\mathcal{K}}||Z| + (|\mathcal{P}_{\mathcal{K}}| + |\mathcal{N}|)|\mathcal{N}||\mathcal{E}||\mathcal{P}_{\mathcal{K}}|\Big)$$

with $|\mathcal{N}| = \max\{|\mathcal{Q}|, |\mathcal{D}|, |\mathcal{T}|\}$.

First we copy $\mathcal{S}$, this has a complexity of $\mathcal{O}(|\mathcal{S}|)$. Next, we have a nested loop. The outermost loop only terminates if we have not determined any new sensor in the previous iteration, hence $\mathcal{P}$ is maximal and has become $\mathcal{P}_{\mathcal{K}}$. In the worst case, each iteration may only add one new sensor, thus the loop has at most $|\mathcal{P}_{\mathcal{K}}| - |\mathcal{S}|$ iterations. Because of $|\mathcal{S}| \leq |\mathcal{P}|$, this has a complexity of $\mathcal{O}(|\mathcal{P}_{\mathcal{K}}|)$. The next layer iterates through all equations, therefore it has a complexity of $\mathcal{O}(|\mathcal{E}|)$. In the innermost loop, we apply an equation to $\mathcal{P}$ as often as it yields a sensor not in $\mathcal{P}$. All such applications are calculated by Algorithm 2. By Theorem 1 and since for each call of Algorithm 2 $\mathcal{P}$ is at most $\mathcal{P}_{\mathcal{K}}$, this loop has a complexity of $\mathcal{O}((|\mathcal{P}_{\mathcal{K}}| + |\mathcal{N}|)|\mathcal{N}|)$. All in all, the nested loop's complexity is $\mathcal{O}(|\mathcal{N}||\mathcal{E}||\mathcal{P}_{\mathcal{K}}|(|\mathcal{P}_{\mathcal{K}}| + |\mathcal{N}|))$.

The final loop iterates once through Z and checks if a certain element is in $\mathcal{P}$, this costs at most $\mathcal{O}(|Z||\mathcal{P}_{\mathcal{K}}|)$. Therefore, Algorithm 1 has a complexity of $\mathcal{O}(|\mathcal{P}_{\mathcal{K}}||Z| + (|\mathcal{P}_{\mathcal{K}}| + |\mathcal{N}|)|\mathcal{N}||\mathcal{E}||\mathcal{P}_{\mathcal{K}}|)$
□

Corollary 2 (Complexity of Algorithm 1). *Algorithm 1 has a complexity polynomial in its input.*

Algorithm 1's input n consists of $\mathcal{K} = (\mathcal{S}, \mathcal{E})$ and Z. We can derive the sets $\mathcal{Q}, \mathcal{D}$ and $\mathcal{T}$ from this input by iterating once through each of $\mathcal{S}$ and $\mathcal{E}$ adding each new entry to its respective set. Thus, all three sets are linear in size to n and the largest of them $\mathcal{N}$ has size n. Since we can determine at most all possible abstract sensors, $\mathcal{P}_{\mathcal{K}} \subseteq \mathcal{Q} \times \mathcal{D} \times \mathcal{T}$. Thus, $\mathcal{P}_{\mathcal{K}}$ is at most cubic in size to n. All in all, we can broadly limit our complexity to $\mathcal{O}(n^3 \cdot n + (n^3 + n) \cdot n \cdot n \cdot n^3) = \mathcal{O}(n^8)$. Therefore, the complexity is polynomial in the input size n. □

4.1. Specification algorithm

We can adapt Algorithm 1 by adding a hash map that stores for each sensor s not in $\mathcal{S}$, which equation application added s to $\mathcal{P}$. Note that multiple equations may exist that applied result in adding a specific sensor. If we now want to specify an abstract sensor in the *A²DNA*, then we just have to substitute each sensor in the *A²DNA* with either a sensor in $\mathcal{S}$ or with the operator and sensors implied by the equation. If any substituted sensor is still not in $\mathcal{S}$, we repeat the substitution until all sensors are in $\mathcal{S}$.

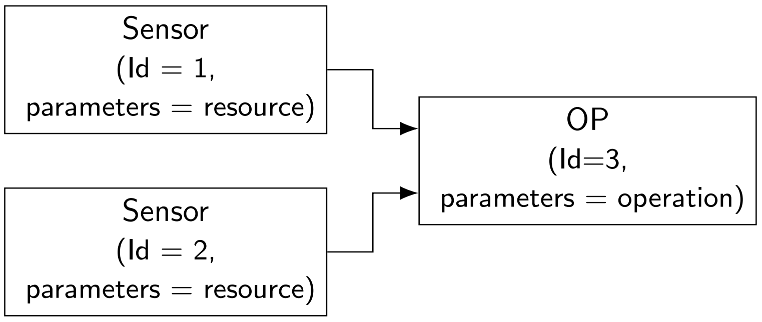
Example 8. *Consider again the knowledge base*

$$\mathcal{K} = (\{(v, x, c), (v, y, c)\}, \quad \{xy = OP_{xy}(x, y)\})$$

and the A^{2}DNA *fragment in Figure 6. The determinator has derived the specification for the abstract sensor with ID 1. We have to use a building block for* OP_{xy} *and connect the sensor blocks for the car's velocity in* x*- and* y*-direction. An exemplary ADNA and net list are sketched in Figure 7.*

5. Conclusion

In this paper, we have proposed a solution to the ADNA's rigidness and lack of semantic knowledge about the hardware by extending the ADNA to the *A²DNA* with its own knowledge base. We have described how the *A²DNA* connects back to the ADNA by specifying the *A²DNA* and have provided a first algorithm in polynomial time for this task. The *A²DNA* and the knowledge base strength the system's self-explaining property by providing a more nuanced understanding of the system's hardware. Additionally, if the determinator has specified an abstract sensor with a combination of sensors, it can also explain the functionality of these



```
1 = 500 (1:3.1) 1 // sensor, resource id for the x-direction sensor
2 = 500 (1:3.2) 2 // sensor, resource id for the y-direction sensor
3 = 2 xy          // operation, performs the axes addition
```

Figure 7: A simplified representation of Figure 6's specified A^2DNA.

structures. Similarly, a hardware extension or new equations are easily communicable to the knowledge base. The A^2DNA also allows the system to better adapt to the loss of sensors. If the access to a sensor is lost, the determinator can specify the A^2DNA on its 'new' reduced hardware at run time and adapt to the changes. Thus, providing the base for a better self-healing.

For the future we plan to provide a first full implementation of the A^2DNA, the knowledge base and the determinator. OWL by [8] and its C# API by [9] for SPARQL seem a promising base for such an implementation[10]. In the first stages, the knowledge base will be provided by experts but we plan to later supplement this expert knowledge with learning algorithms. Finally, we plan to determine more precise boundaries for the time complexity and perform a space complexity analysis.

References

[1] R. Allrutz, C. Cap, S. Eilers, D. Fey, H. Haase, C. Hochberger, W. Karl, B. Kolpatzik, J. Krebs, F. Langhammer, et al., POSIPAP organic comp - vde, 2003. URL: https://www.vde.com/resource/blob/932548/bfcfaa9bae199aa27f888319c396d6ed/fa-6-1-organic-computing-download-akkordeon-data.pdf.

[2] S. Tomforde, B. Sick, C. Müller-Schloer, Organic Computing in the Spotlight, CoRR abs/1701.08125 (2017). URL: http://arxiv.org/abs/1701.08125. arXiv:1701.08125.

[3] U. Brinkschulte, Technical report: Artificial dna - a concept for self-building embedded systems, ArXiv abs/1707.07617 (2017).

[4] U. Brinkschulte, An artificial DNA for self-descripting and self-building embedded real-time systems, Concurrency and Computation: Practice and Experience (2015).

[5] U. Brinkschulte, M. Pacher, Semantic Description of Artificial DNA for an Organic Computing Middleware Architecture, in: Proceedings of the 1st International Workshop on Middleware for Lightweight, Spontaneous Environments, MISE '19, 2019, p. 1–6.

[10]An experimental implementation exists but would go beyond this paper's scope.

[6] H. Schaub, J. L. Junkins, Analytical Mechanics of Space Systems, American Institute of Aeronautics and Astronautics, Inc., 2018.

[7] Qniemiec, RPY angles of cars, 2010. URL: https://commons.wikimedia.org/wiki/File:RPY_angles_of_cars.png, https://commons.wikimedia.org/wiki/
File:RPY_angles_of_cars.png
Changes made: Removed the boxes "RPY angles" and "ENU coordinates", renamed "body frame" to "car frame".

[8] B. Parsia, P. Patel-Schneider, B. Motik, OWL 2 web ontology language structural specification and functional-style syntax, W3C, W3C Recommendation (2012).

[9] Project dotNetRDF, DotNetRDF, 2009. URL: https://dotnetrdf.org/, https://dotnetrdf.org/.