ICED13
INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN, ICED13
19-22 AUGUST 2013, SUNGKYUNKWAN UNIVERSITY, SEOUL, KOREA

BEYOND FRAMES: A FORMAL HUMAN-COMPATIBLE REPRESENTATION OF IDEAS IN DESIGN USING NON-GENETIC AD-HOC AND VOLATILE CLASS MEMBERSHIPS AND CORRESPONDING ARCHITECTURE FOR IDEA OPERATORS

Christos SPITAS
TU Delft, The Netherlands

ABSTRACT
This paper proposes a formal human-compatible representation of ideas in design using ad-hoc and volatile class memberships and corresponding architecture for idea operators. The proposed theory departs from genetically inspired object-oriented concepts used in state-of-the-art implementations of Frames, Semantic Networks and the Semantic Web, and Description Logics, which are limited by the endemic complexity of (multiple) inheritance, and whereby reinventing/ reorganising ideas, as is a staple of creativity, is tedious and error prone. Instead, the paper proposes that ideas do not get 'born' out of 'parents'; rather, they exist as either present or not (yet) to consciousness. Thus ideas can be considered as either eternal, or more practically, self-emergent. This is an obvious yet radical and highly promising shift in paradigm, allowing ad-hoc classes, volatile classes, and an unlimited capacity to reinvent/ reorganise classes. Limitations of tree structures and forced directionality present in the genetically-inspired state-of-the-art vanish altogether.

Keywords: human behaviour in design, integrated product development, ontologies

Contact:
Prof. Dr. Ing. Christos Spitas
TU Delft
Industrial Design Engineering
Delft
2628CE
The Netherlands
C.Spitas@tudelft.nl

1 INTRODUCTION

Design is the activity most characteristic of civilisation, in that it is design that produces the visible manifestations thereof. Although in recent decades there has been much progress in supportive methods and technologies for design, it is still the case that automatic means have not been able to progress beyond the level of 'pastiche', i.e. with results that can be actually surprising and considered creative: it is still up to humans to design.

At the same time, the research in computerised representations of knowledge, be it towards the direction of Artificial Intelligence or just 'humble record-keeping' has given rich fruit in itself: Frames (Farenhorst & de Boer, 2009) (incl. Ontology languages), Semantic Networks and the Semantic Web (Hitzler, et al., 2009) all serve to describe ideas, which are either known (knowledge) or emerging. Objects within these organizational frameworks are considered to have genetic-type relationships, i.e. 'child' ideas will 'inherit' at least some properties from 'parent' ideas, giving rise to a sort of genealogical tree. For the purposes of design it is not sufficient to simply classify and enumerate ideas like that, but useful relationships must be established: Often classification is enough, as in frameworks used in medical databases, where a diagnosis can be as straightforward as checking all the symptom classifications until only one possible parent class remains un-eliminated. In more advanced models, Description Logics (Baader, et al., 2007) can be used to connect ideas in logically verifiable statements/ propositions.

However, although such representations have found some use in critical applications such as aerospace and the military, it has not been practical, in general, to adopt these representations in the daily human creative process of design. We can attribute this to two main reasons:

1. *The genetic analogy does not necessarily hold for ideas*, as it does for biological organisms. Although it is quite common to assign the term 'genesis' (birth) to any phenomenon of emergence, there is no evidence that any of the biological mechanisms, such as inheritance, are at play when ideas emerge. In fact, single or multiple inheritance create such dependencies between ideas that, just like in genetics, there is a lot of information carried over from one 'generation' to another that can be absolutely useless, and tedious to a human who tries to think in these terms. Secondly, reclassifying ideas (which should normally be impossible, if we stay true to the genetic analogy) is a staple of human creativity, but can lead –if attempted– to incompatible cyclic references between classes, hence critical errors, and is tedious at best.
2. *The representations are not sufficiently accessible by humans*, in that when visualised they take the form of sprawling tree structures or neural networks, both of which start to challenge the capacity of human cognition even with simple systems being represented. Even when there are algebraic semantics behind the representation, there is no accessible visual calculus applicable and visual thinking can become confused and confusing.

On the other side of the methodological spectrum stand much more human-accessible methods such as mind-mapping, sketching, collage and an entire host of other formal and improvised methods for representing ideas on paper, the computer screen or elsewhere; however, these suffer from a lack of formalism that makes it difficult to describe ideas to significant depth, or even reproducibly (including by others) and unambiguously. Sooner or later, these same 'intuitive' methods are encumbered by a similar complexity as seen in Frames and the other formal methods discussed above. Interestingly, object-oriented inheritance notions underlie the use of some of the informal methods as well, although in a somewhat less obvious way, as evidenced by the predominantly tree-like representations of, i.e. mind-mapping rules and software.

Regardless of their choice of method and supporting framework/ software, we can thus conclude that designers are encumbered in their creative process, having to rely on idea representations that are either intuitive and accessible but not formal, or formal but not intuitive and accessible. The vision behind the present research is that idea representations should be intuitive to humans and easy to work with in a casual manner, and at the same time formal enough to allow for validations and implementation in computerised design environments.

In this paper we attempt to address the problem outlined above by proposing a representation of ideas, both in formal mathematical terms and graphical-visual terms, that allows ad-hoc and volatile classifications and logical operations. We base that on a fundamental proposition of equivalence between ontology and activity, or in other words: existence and truth. The proposed theory departs from genetically inspired object-oriented concepts used in state-of-the-art implementations of Frames,

Semantic Networks and the Semantic Web, and Description Logics, which are limited by the endemic complexity of (multiple) inheritance, and whereby reinventing/ reorganising ideas, as is a staple of creativity, is tedious and error prone. Instead, the paper proposes that ideas do not get 'born' out of 'parents'; rather, they exist as either present or not (yet) to consciousness. Thus ideas can be considered as either eternal, or more practically, spontaneously-emergent. This is an obvious yet radical shift in paradigm, allowing ad-hoc classes, volatile classes, and an unlimited capacity to reinvent/ reorganise classes. We shall offer some first examples of the use of the proposed representations, and demonstrate how limitations of tree structures and forced directionality present in the genetically-inspired state-of-the-art vanish altogether.
This first exploratory work is grounded on reasoning from basic principles and as such appropriately offers only basic examples for the purpose of demonstration. No attempt is made at prescribing a fully-fledged design process/ algorithm based on the proposed representation. Empirical validation on case studies of higher complexity is to be a next step in this research.

2 MODELLING OF IDEAS

2.1 Fundamental considerations

We set as principal requirement that *any model used in design (including its representations of concepts and processes) must be intelligible to humans*. Simply put, the human and the computer must be able to speak about the same 'things' in the same way. Already we make the choice that such a model must be human-based, and therefore we must seek adequate computer representations and implementations of such a model, as opposed to adapting a computer-based model to humans: the reason is that with the affordances of modern computer languages we can programme computer implementations almost at will, whereas we cannot 're-programme'/ condition human cognition as easily.
Thus the first step will be to define a currency for 'things'. In a previous work (Spitas, 2011) we postulated that the 'idea', stripped from any other connotation than its lexicographical meaning, is central to describing everything in the context of human cognitive activity, including design. To a human, *an idea is anything that is actually or potentially present to consciousness* (Merriam-Webster, 2012). This is also backward-compatible to existing theories: In particular, 'idea' can be used to describe both 'objects' and 'processes' involving objects, as defined in Korn's linguistic approach (Korn, 1996), 'objects' of the Autogenetic Design Theory (Vajna, et al., 2005), 'modules' and their 'connections' as per the Formal General Design Theory (Maimon & Braha, 1999), 'working surface pairs', 'channel and supports structures' and other constructs of the Contact & Channel method (Albers & Matthiesen, 2002), 'concept' and 'knowledge' in the C-K theory (Hatchuel & Weil, 2003). Further, 'idea' is casually used, i.e. by Walker (Dickson, 1990), in the context of the more abstracted levels of product development, and in fact any innovative thought, envisioned embodiment of an artefact or process, foresight, etc. is commonly referred to as an idea: 'I have an idea!' (Spitas, 2011).
There are those who would assign the words 'idea', 'concept' and 'embodiment' to ideas of increasing level of maturity (or detailing) within the design process, but if we consider the natural semantics of the words (Merriam-Webster, 2012) this is an artificial restriction: i.e. there is no reason, given the lexicographical definition of 'idea', why 'concept' should mean anything different.
The all-encompassing definition of 'idea' admittedly poses challenges to the computer implementation (which is tackled in the next section), but has an obvious benefit to offset this challenge: *because it is the common denominator of all the more specialised representations of cognitive objects, it allows the cross-talk of domains that use very different representations, terminology, and relationships*. In fact, ideas suffice to represent 'things', relations between 'things', and so forth, across any conceivable level of abstraction and allows to cross any preconceived barriers between ontology and function: i.e. Define 'red': is it a colour, an object property, or a descriptor of a function, subject to lighting conditions and the state of the human eye receptor? Or: When a 'concept' becomes 'knowledge', does it stop being a concept, as in 'present to consciousness' (Merriam-Webster, 2012)? Such paradoxes found in orthodox Frame-based classifications are no longer relevant when we adopt the idea-class as the single root class in the cognitive domain.
Additionally, product development entails the consideration, representation and manipulation of ideas not only referring to the product system itself, but also at least the user(s), the business/ enterprise(s) developing and otherwise related to the product, and further the context at large, society and the

environment. These ideas belong to very different domains, but are obviously and essentially coupled, as shown in Figure 1. In such a situation it becomes vital to use such a common denominator as is the 'idea', if only to assure that indeed such cross-talk can take place.

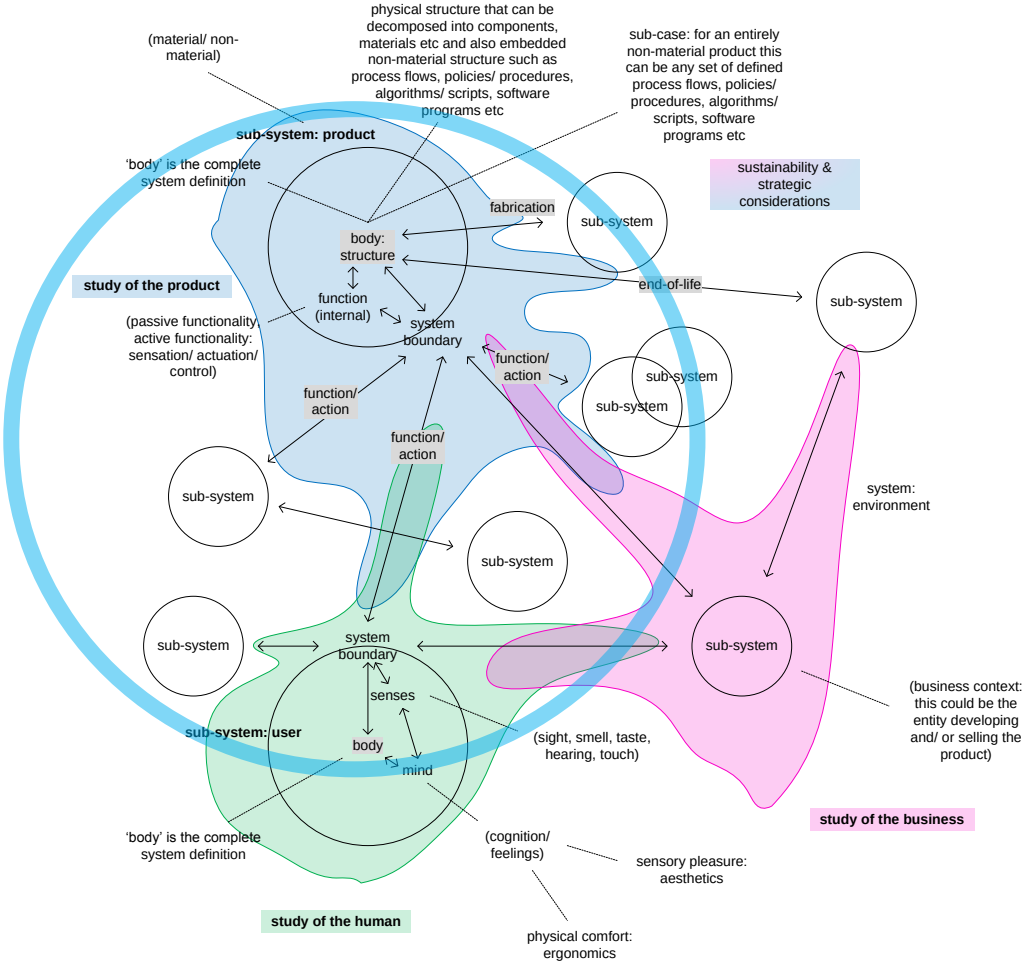


Figure 1. Systems view of Product Development (in blue circle) and couplings to the essential context; ideas pertain to very different domains

2.2 Idea representation

Following from the previous discussion, where we consider ideas as representing all things in the cognitive domain, their actual representation must be both very basic and straightforward. An idea is simply a container of any other ideas, or nothing: the 'null' idea, and when called must evaluate to whatever we perceive as its value. Sets of ideas and operations on or between ideas are also ideas.

The 'content' of the idea needs only be the mental representation we have of it. A set/ list of ideas will contain those very ideas, preceded by the notion that it is a set/ list; an operation on ideas will contain its arguments and relationships, preceded by the notion that it is an operation, etc

Using a Lisp-like format, any idea can be represented based on the above in the form of a cons-pair:

```
(first rest)
```

This simple first-rest form complies with our fundamental requirement that it must be intelligible to humans.

An idea can be given a unique name to be referenced by (here we may at times note these in capital letters, following the Lisp convention) and when called returns a value. We define the value of an idea to be in general also an idea.

One aspect of this value relates to the actual existence or occurrence of the idea: Since ideas can describe both ontological entities and activities, it is common to consider existence-states and occurrence-states. i.e. 'John exists' (existence) and 'John speaks' (occurrence). At the same time, we can postulate that:

John exists = it occurs that (John exists)
John speaks = it exists that (John speaks) = 'John speaking' exists
In this manner we conceptually unify ontology and activity. The state describing each (exists/ occurs) can be uniformly described as true/ false (represented hereafter as T/ F). This deterministic state description (meaning always true/ always false) can be expanded in an obvious way by assigning a probability; furthermore, if the idea being described admits more states than just the Boolean true/ false, then instead of a single probability a Probability Density Function (PDF) can be assigned. It is further conceivable that such a PDF may be multi-parametric, depending on the parameters we currently attribute to the value.
Hereafter we shall refer to this truth-state, or probability, generally as p. As an indication, its values may be true/ false, a real number between 0 and 1 representing a probability, or qualitative descriptors such as 'frequent/ probable', 'rare/ improbable' etc. The information represented by p may be revised ad-hoc to suit the current context of use.
In general, the value of an idea can be reported as a pair (v, p), hence a value accompanied by its truth state or probability. A PDF can replace this pair where applicable.
Here the form `(@ value truth-state)` is pseudo-code representing the pair (v, p). As a convention to help readability of the representations, every idea stated as `(idea)` is understood as equivalent to `(@ idea T)` or `(@ idea 1)`, whereas ideas must be explicitly defined as having a different truth state, when that is the case.
As an example of the utility of using such descriptions during design, consider a simple case of a machine element: a shaft.
Consider that, upon first conception, only the 'shaft' idea itself is present to consciousness, without any other ideas to specify it further. Suppose that we are pretty sure we want a shaft and we are in a position to impose our choice, hence $p = \mathrm{T}$ (or equivalently $p = 1$). With nothing to attribute to it (yet), we represent this shaft as:

`(nil)`, or equivalently `(@ nil T)`

Next the ideas 'solid' and 'cylinder' (i.e. as opposed to hexagon etc) may come to mind, specifying the idea to 'cylindrical shaft'. Again suppose $p = 1$ for this idea.

```
(list
  (solid)
  (cylinder))
```

Again, we may write:

```
(@ (list (solid) (cylinder)) 1)
```
or even `(@ (list (@ (solid) 1) (@ (cylinder) 1)) 1)`

'Cylindrical shaft' next gets elaborated to 'Cylindrical shaft with diameter 3.20mm ±0.02'. Here depending on our fabrication/ procurement expectations suppose it makes sense to assign $p = 0.998$ to this idea, hence (v, p) = (3.18mm-3.22mm,0.998). This means that we cannot be certain that all possible instances of this idea (class) will have the diameter specified. This basic tolerance definition, here embedded in the design process itself, can conceivably be substituted with a PDF, i.e. (v, p) = Gaussian distribution with μ=3.20mm, etc. Further, a length of 8mm may be specified, possibly to be assigned a tolerance later, etc We can represent this idea as follows:

```
(list
  (solid)
  (cylinder (list
    (diameter (@ (interval 3.18mm 3.22mm) 0.998))
    (length 8mm))))
```

2.3 Idea attribution

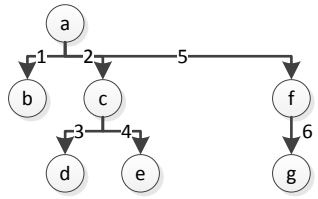
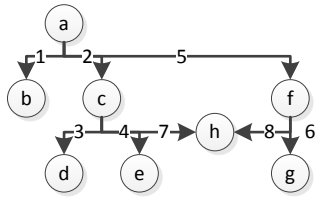
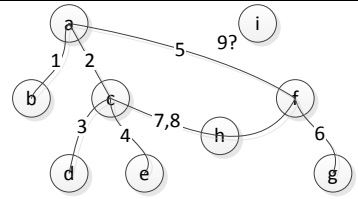
Under the present paradigm ideas can combine in a single way: *attribution*. We define attribution as nesting ideas into cons-structures, i.e. `(a (b))` attributes b to a. Giving a new attribute to an idea is as straight-forward as nesting another idea within it.

This also potentially gives rise to peer relationships, if two or more ideas are attributed to another at the same cons-level, i.e. forming a list: `(list a b)`. Notice how even in a peer relationship the peers are still attributes to the containing idea (i.e. the relationship itself). In neither case must the combination of ideas affect the original ideas (here a or b), hence we define operations to be non-destructive.

2.4 Ad-hoc definition of ideas versus classification and instantiation

Object oriented paradigms make clear distinctions between classes and instances and furthermore require that classes be defined before their instances. New classes can be produced through single or multiple inheritance and their relationships must be fixed upon definition.

Table 1. Comparison of idea definitions and relationships

		
(a) Single class inheritance, fixed relationships upon definition	(b) Multiple class inheritance, fixed relationships upon definition	(c) Ideas can ad-hoc be attributed to (more than one) relationships

In our present paradigm we have not needed, however, to define or distinguish between classes and instances, or define inheritance mechanisms. All of these concepts fall under our singular definition of 'idea'. This affords the freedom to define ideas ad-hoc and with no constraints. Further, we can relate ideas to each other by attribution, thereby forgoing the necessity of classification and obtaining the ability to produce complicated non-hierarchical networks of relationships, which human cognition already supports.

This can be seen in Table 1-c: Idea/ class (i) can be filed on-the-fly without any relationship to other classes; ideas (c), (h) and (f) are connected by one set of relationships, and (h) is no longer a 'child', but a peer to (c) and (f)). Overall, ideas are self-standing and free to relate and re-relate ad-hoc: the relationships are not fixed in the idea (class) definitions. Compare to 1-a and 1-b in the same Table, where relationships 1-6 would be part of the class definitions of (a)-(g) and 7-8 part of (h), and thus any change in the relationships would have to change the classes themselves.

Revising, specifying or abstracting ideas is as straightforward as modifying the attributions accordingly:

1. *Specification*: This could be observed in the example in section 2.2 where 'shaft' was defined and specified progressively, as more ideas were attributed to it. Important to note is also that there is no explicit hierarchy requirement, i.e. with regard to the level of abstraction or the sequence of the ideas attributed to the shaft.
2. *Abstraction*: Consider the idea `SURFACE-BY-REVOLUTION` defined as `(list (cylinder) (cone) …)`, thus both `CYLINDER` and `CONE` are attributed to it. Substituting either by the former abstracts them, and in turn abstracts any other attribute, such as `SHAFT`. Removing an attribute altogether is also abstraction.
3. *Revision*: Consider the 'cylindrical shaft' being replaced by a 'conical shaft', corresponding (at a high level) to the formal representation `(list (solid) (cone))`. Addition or removal of attributions is also possible. The level of abstraction need not stay the same during this operation, therefore we understand specification and abstraction, as defined previously, as just two subcases of revision.

2.5 Elimination of classification conflicts

Typically in inheritance-based object-oriented representations the question arises: How to classify an idea?

i.e. In CAx and PDM systems this is frequently the case with classifying parts: Should they belong to sub-assembly 1 or sub-assembly 2, or perhaps the overarching assembly or a layer beneath? In mind maps the equivalent question is: to which branch does an idea belong?

Conceptually, this problem stems from inheritance itself, and is eliminated automatically in the present paradigm: Ideas can be free (seen i.e. as the 'floating' idea in Figure 2-c), or attributed ad-hoc, with all such relationships free to be revised later.

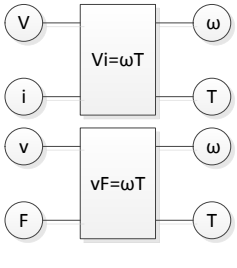
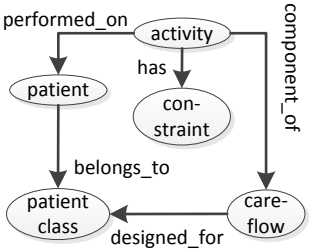
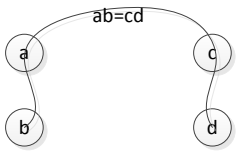
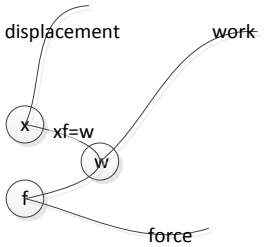
2.6 Operations on ideas

Presently rules and operations offered in state-of-the-art software are either too abstract (Frames, semantic networks), too cumbersome (Description Logics), or not generalizable across domains (i.e. Simulink, Wolfram System Modeller). We examine some typical cases as follows:

1. Typically relationships and physical phenomena are represented as context-specific pre-defined models, each *restricted to a specific class*, i.e. as the transformation blocks seen in Table 2-a, where an electromechanical transformer (i.e. an electromotor) must be represented by a different element than a mechanical-mechanical transformer (i.e. a gear-and-rack) etc, even though clearly the underlying abstraction is the same. This makes such models especially cumbersome i.e. in concept design, where the required specification of all the model parameters is felt as premature.
2. Basic Frames on the other hand will typically allow for operations related to class-membership only (i.e. 'is a'/ 'has a'/ 'does'), which can be represented as directed graphs, as shown in Table 2-b. However, these operations are somewhat trivial and their directionality can be arbitrary (i.e. dictated by the choice of active or passive voice: 'belongs to' or 'owned by'?) and, by the same token, confusing.

The possibility to implement knowledge as abstract as laws (i.e. physical laws), and then have models automatically emerge from these is not present in the previous examples. This restricts much of the state-of-the-art to largely predefined solutions, that fail to support designers if they venture 'out of the box'.

Table 2. Comparison of operation implementations

<table>
<tr>
<td> 
</td>
<td> 
</td>
<td> 
</td>
<td> 
</td>
</tr>
<tr>
<td>(a) Simulink/ System Modeller operators</td>
<td>(b) A typical medical schema, Frame-based directed graph</td>
<td>(c) Generic operation, class independent</td>
<td>(d) Rule implementation using 'floating' attributions</td>
</tr>
<tr>
<td colspan="2">State-of-the art in operation implementations</td>
<td colspan="2">Generalised operations</td>
</tr>
</table>

In the present paradigm we consider that rules and in general operations are ideas also and share therefore all the definitions and properties given in the previous sections. We treat them specially here in that unlike other ideas, which return a fixed value, operations evaluate dynamically, based on ideas passed as arguments.

The absence of class-restrictions means that operators can be made generic and later specified ad-hoc as needed. Obviously, operations will not evaluate until the ideas operated on are made compatible to such evaluation (i.e. are specified enough), but it is important that in the meantime the operation can act as a meaningful placeholder, prompting what is still needed before it can evaluate. Such an operation is shown in Table 2-c, which can effectively replace all fixed-class operations of the first column.

Additionally, the generic nature of many laws can best be captured in a type-neutral way, as i.e. in Table 2-d, which shows the relationship between displacement, force and work: Laws can be represented as rules bundled with 'floating' attributions external to the idea definitions (as opposed to operations being embedded into classes themselves, as per the object oriented paradigm). Using our Lisp-like pseudo-code we can represent this as:

```
(operation (x f w) ((= (* x f) w) (attr-q x displacement)
                                  (attr-q f force)
                                  (attr-q w work)))
```

where `attr-q` returns 'true' if the first argument is an attribute of the second argument, in our case if x is attributed to the set of displacements etc
This conveys the law that work is equal to the product of displacement and force. It can readily be generalised to all kinds of Lagrangian and Hamiltonian DOFs by abstracting displacement and force accordingly.
As a second example, consider the following representation of the equilibrium of a body:

```
(operation (body)
  (= (+ (all x (and (attr-q x body)
                    (attr-q x force))))
    0))
```

where `all` returns a list of all ideas x for which the second argument is true.

So far we considered the deterministically 'true' state of the operations and their arguments. Like all ideas, however, operations admit more truth-states: Note that, just like its value, the truth-state of an operation is the combined result of the truth-state of the operation itself and the truth state(s) of its argument(s), i.e. consider investing a capital that will be between 0-1000 Euro 90% of the time in such a way that it will yield 1-1.1 times the capital 50% of the time, hence:

```
CAPITAL:(@ (interval 0euro 1000euro) 0.9)
INVESTMENT:(operation (x) (@ (* (interval 1 1.1) x) 0.5)
```

What we propose to do is `(investment (capital))`, which will yield
`(@ (interval 0euro 1100euro) 0.45)`

3 IDEA VALIDATION AND DESIGN ALGEBRA

Frames do not include calculus, or any means of validation. Description logics do, as they structure information in the form of logical statements that can be combined and evaluated: a logical combination of such statements evaluating to 'true' can be assumed to be valid. Such validations tend to involve many statements and easily become massive, cumbersome, and cognitively complex. This is because the validation algorithm is external to the ideas involved and must recombine and reevaluate them from the ground up every time something changes.
Idea validation in the present paradigm is no longer external to the ideas. As has been presented in section 2, ideas evaluate to a truth-state which can be T by default, or some other value, including probabilities and PDFs. Any idea evaluating to a given truth-state will be realizable with a corresponding probability. The more accurate the information in our cognitive model is, the more accurate we can expect the corresponding prediction to be. In this way, the 'future' is mapped to the 'cognitive present', and scenarios, new ideas and 'what ifs' can be explored at will.
Ideas represent also operations (including design operations); thus section 2 provides sufficient basis for an Algebra of ideas. It is outside the scope of the present work to define a complete and orthogonal set of operations that would give an algebra sufficient for the purposes of design: this will be the topic of future research.

4 GRAPHICAL IMPLEMENTATION

Already in section 2 (Tables 1-c, 2-c and 2-d) we have illustrated specific cases of representations to compare them with the state of the art.

Here, consistently with the definitions given in section 2, we provide a set of primitives and examples for graphically representing ideas (Figure 2, left) using two alternative representations: a formal (explicitly showing attributions) and a simplified one, where ideas can be represented also as lines (crossing their attributes).

On the right side of Figure 2 we offer a comprehensive representation of a law of mechanics in the form of a network. Values and truth states can be indicated next to ideas in these representations as needed (not shown here). This example also illustrates mixed use of formal and simplified representations to improve readability.

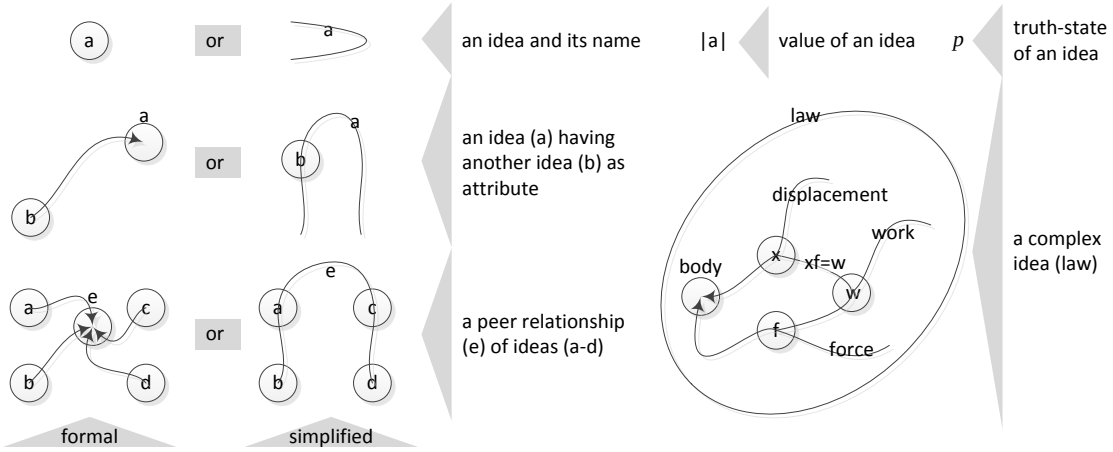


Figure 2. Primitives and examples for the graphical representation of ideas, featuring combinations of formal and simplified notation

It has been established by repeated testing on different cases that the primitives shown in Figure 2 are sufficient for representing all ideas definable under section 2.

Compared to commonly used schemata and tree structures (i.e. as seen in Tables 1-a, 1-b, 2-a, 2-b), these non-hierarchical graphs hold more meaningful information, are extensible and allow visual operations and validation, especially when the truth states are also indicated next to the ideas.

Tentatively, tests with a number of students conducted over the past 1 ½ years indicate that this representation is intelligible to humans, as well as reasonably intuitive. Future research will test the extent of the validity of these assertions.

5 DISCUSSION

In this work we have defined ideas without considering classifications. Rather, every idea is potentially a class of itself. Ideas can be as abstract as to only carry the information of their name (containing nil), or can be specified to any depth by adding cons-levels to them: i.e. the shaft was specified by a solid and a cylinder, which in turn was specified by a diameter and a length, which in turn were specified by numeric values. All ideas were assigned –implicitly or explicitly– truth states.

Essentially, we have shown that the concept of inheritance does not need to play a role in defining ideas. Instead, ideas can be defined to have other ideas as attributes and as such be characterised by those, as i.e. the shaft-idea was characterised by the cylinder-idea and in turn by all its attributes. Idea definitions can be ad-hoc, in the sense that there is no given or exhaustive definition for i.e. ‘shaft’ or ‘cylinder’. Instead, ideas can be extended (by adding peers to their attributes), specified (by adding new cons-levels of attributes), or abstracted (by removing levels of attributes) at will, according to the current needs and inspiration. This allows ideas to be represented on the fly as they emerge (perhaps just by a placeholder with a name), then to be specified only inasmuch as needed to be meaningful in a given context, and then possibly to be elaborated or revised later.

With inheritance losing its significance, classes become more fluid and lose their formal rigidity:

1. Firstly, as any idea can be specified ad-hoc, it can be seen as a class in itself, able to spawn any number of instances, which in turn are classes in themselves and so on. Thus every idea can be seen as a class and also an instance of another class (or of itself).

2. Secondly, as in principle any idea can be attributed to any other, formal class hierarchies become impractical. I.e. consider a ‘solid shaft’ and a ‘shaft-like solid’. They are two different ideas, both stemming from the attribution of one idea to another and vice-versa. Clearly, the resulting hierarchies by attribution are context-specific and no global class hierarchy can be defined between ‘shaft’ and ‘solid’. Instead, all ideas are potentially peers or attributes to one another.
3. Thirdly, just as classification is replaced here by ad-hoc attribution of ideas, reclassification is replaced by changing the attributes to ideas.

Operations being ideas themselves, and admitting the same representations and rules, allow a simultaneous overview of both design objects and the design process and thus potentially support an increased level of awareness and corresponding initiative on the part of the designer. Furthermore we have shown that by virtue of assigning truth states to ideas, validation is straightforward and intuitively accessible.

Lastly, we have proposed a Lisp-like computerised syntax and a corresponding small number of graphical primitives that allow representation of ideas both formally, allowing computerised support via the developed algebra, and intelligibly to humans.

Thus we propose that this paradigm is more consistent with the self-emergent nature of ideas and the way human cognition naturally operates, compared to state-of-the-art representations such as Frames, Semantic Networks or Description Logics. While clearly this paradigm is heavily inspired by the same state-of-the-art and the Lisp programming language and builds upon many of the concepts of the theories it proposes to replace, it departs sufficiently to allow a more designer-centred and human-compatible way of working. We can expect that this will support creative and intuitive thinking by means of idea representations that are intuitive to humans and easy to work with in a casual manner, and at the same time formal enough to allow for validations and implementation in computerised design environments.

Given that this is a tentative exposition of this representation paradigm, early proof of concept has been given by means of simple examples only. Several more have been explored thus far with students, giving consistent results. A more thorough treatment of the topic, including a) the definition of a complete orthogonal set of operations sufficient to design, b) the prescription of a design methodology based on these operators and c) empirical validation and benchmarking to other theories, will be the subject of further research.

ACKNOWLEDGMENTS

The author would like to extend his gratitude to the good colleagues Profs. Vasilios Spitas, Sven Matthiesen, Tetsuo Tomiyama, Sandor Vajna, and Mounib Mekhilef, and by extension to the whole Autogen-CT project team, for the many inspirational discussions over the past year, in particular with regard to the state of the art in object representations in design and rule-based modelling.

REFERENCES

Albers, A. & Matthiesen, S. (2002) Konstruktionsmethodisches Grundmodell zum Zusammenhang von Gestalt und Funktion technischer Systeme - Das Elementmodell ‚Wirkflächenpaare & Leitstützstrukturen’ zur Analyse und Synthese technischer Systeme. *Konstruktion, Zeitschrift für Produktentwicklung,* Volume 54, pp. 55-60.

Baader, F., Horrocks, I. and Sattler, U. (2007) Description Logics. In: *Handbook of Knowledge Representation.* s.l.:Elsevier.

Dickson, K. (1990) The role of design in the innovation process. *Journal of Engineering Design,* Vol. 1, No. 3, pp. 269–278.

Farenhorst, R. & de Boer, R. (2009) Knowledge Management in Software Architecture: State of the Art. In: M. Babar, T. Dingsoy, P. Lago & H. van Vliet, eds. *Software Architecture Knowledge Management- Theory and Practice.* s.l.:Spinger.

Hatchuel, A. & Weil, B. (2003) *A new approach of innovative design: an introduction to C-K theory.* Stockholm, s.n., pp. 109–124.

Hitzler, P., Krötzsch, M. and Rudolph, S. (2009) *Foundations of Semantic Web Technologies.* s.l.:CRCPress.

Korn, J. (1996) Domain-independent design theory. *Journal of Engineering Design,* Vol. 7, No. 3, p. 293–311.

Maimon, O. & Braha, D. (1999) A mathematical theory of design: representation of design artifacts (part I). *International Journal of General Systems, Vol.* 27, No. 4, p. 275–318.
Merriam-Webster (2012) *Merriam Webster On-Line Dictionary & Thesaurus.* [Online] Available at: http://www.merriam-webster.com/
Spitas, C. (2011) Analysis of systematic engineering design paradigms in industrial practice: Scaled experiments. *Journal of Engineering Design*, Vol. 22, No. 7, pp. 447-465.
Vajna, S., Clement, S., Jordan, A. and Bercsey, T. (2005) The Autogenetic Design Theory: An evolutionary view of the design process. *Journal of Engineering Design*, Vol. 16, No. 4, pp. 423-440.