Tom De Smedt, Frederik De Bleser, Vincent Van Asch,
Lucas Nijs, Walter Daelemans

Gravital: natural language processing for computer graphics

1 Introduction

We have a recurring joke in our research lab. It is simple, and perhaps also silly. Someone yells: "the Graphic Design Manual has been stolen!" This tends to humour the software developers (many researchers in the lab are), since their field (computer science) is clearly defined with procedures, rules and goals. By contrast, the domain of art and design is an exotic jungle. "How could you not have a manual", the developers wonder. The linguists in the lab find the joke mildly amusing. They are used to working with ambiguity and underspecification in language. They can resort to statistical prediction if the manual goes missing. "If there is no manual, we'll need lots of data", they think. Now, artists on the other hand respond with a puzzled look. "Why would anyone steal a useless book?" they ask. In their opinion, everything can be transformed into anything to fit their art. When presented with a manual, we wager some of them would instantly create new artwork that contradicts every rule in the book; equilibrium is not necessarily a goal. So a new chapter of rules would have to be written, and so on, until the manual fills the entirety of The Library of Babel.

No system is without rules however – or pigs would fly. Perhaps we haven't unraveled all of them yet for creativity. Or we may need to break it down into smaller subsets. But as Veale, Feyaerts & Forceville have argued in their introduction to this volume, we know that mental agility is a common hallmark of creativity. Our experiments with "*Gravital*", a software system for sketching and brainstorming, have sought to provoke this mental agility. To put it in the words of Veale, Feyaerts and Forceville, Gravital is a generator of ideas, produced by roaming a search space of concepts. These ideas can be inspiring, interesting, trivial, silly, or rude. Gravital doesn't always recognize the subtle distinction between them.

A central theme in our system's design are concept properties. Properties of things can be used to think about how concepts relate to each other. For example, a beach ball and a balloon both are round, buoyant and playful. This statement is a rule of sorts, but flexible. We work with the notion of "agile concepts". Depending on the context, such a concept can have many meanings: for example, a balloon drifting through an empty subway station is rather sad, and creepy. This

kind of agility or duality allows the system to describe concepts more broadly, in terms of "latent" properties.

2 Computer graphics and user interfaces

Traditionally, software applications for computer graphics have been based on real-world analogies. Each icon in the application's user interface represents a concrete object – a pen, an eraser, scissors, etc. This model raises creative limitations. For one, you can only use the features as the software developers implemented them; creative recombination of tools is impossible when not foreseen. The classical solution, adding more features, is a never-ending cat-and-mouse game that complicates the software with each version. Furthermore, the software's possibilities are also its limitations: users will tend to think along the lines of what is possible and not about what they want (Cleveland [2004]). But perhaps the most critical limitation in the pen-paper-analogy stems from its dependence on the computer mouse. All the different steps and considerations in the creation process are literally lost in translation.

Over the course of two research projects our goal has been to overcome these limitations by building a software prototype that allows people to express themselves creatively regardless of artistic or technical skill, and without being limited to a predefined user interface. The ability to use natural language for interaction with the software is a key component in this software prototype.

In 2002 we released an open-source computer graphics application called NodeBox[1] (Figure 1). It is mainly used to produce "generative art", an art form inspired by emergence and techniques from AI and artificial life (Boden [2009]). NodeBox creates 2D graphics from Python programming code. Users can write Python scripts from a set of programming commands (e.g., rectangle, line, circle, rotate, scale) that yield visual output. This is a fundamental shift away from direct manipulation and towards a focus on language – in this case a programming language. The use of programming code allows the automation of traditionally production-intensive tasks, such as creating a hundred-page document in one consistent visual style but with subtle differences per page. Moreover, the creation process is retained in the script. It can be shared, adapted and reused. The script generates visual output, but it also functions as a vessel of domain-specific expertise.

1 NodeBox for Mac OS X, http://*nodebox.net*

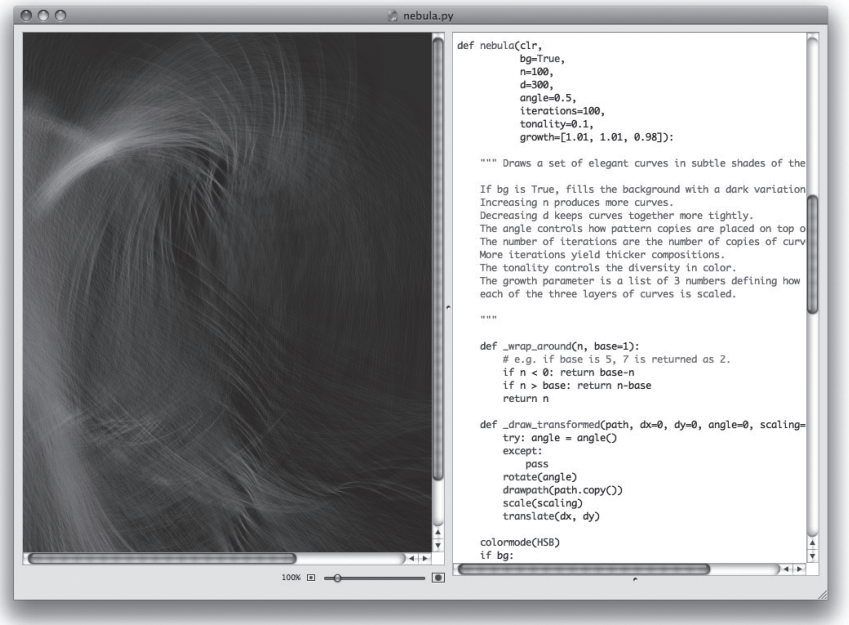

Figure 1: NodeBox screenshot. Python scripts are used to produce visual output.

As the project progressed we turned to the nature of artistic creativity, and how it might be modelled. Providing the technical constructs to create a programming interface for automated graphic design is one step. The next leap is providing functionality that captures some of the heuristics used in graphic design (e.g., function over form, associative thinking, colour harmony, considering the negative, etc.). In this paper we discuss the main components and paradigms in follow-up project to NodeBox: Gravital, an endeavour to simulate and facilitate creative sketching and brainstorming.

Inspiration was drawn from ideas on creativity and fluidity proposed by Margareth Boden and Douglas Hofstadter. Essentially, the system functions on a level of small-C combinatorial and exploratory creativity (Boden [2003]). Small-c creativity produces ideas that are new to their author but not historically novel (big-C creativity, the wheel for example). Consider an art student that, by experiment, "discovers" two colours that look pleasing together (say, deep purple and bright orange). This might strike him or her as a marvellous find. But the find will not necessarily shake the foundations of art history. Combinations of purple and orange were already popular in the seventies. And Roman generals were probably quite fond of the gold embroideries on their purple togas, we wager. But on to Gravital.

In the first part of this paper we give an overview Gravital's Graphics Language Processor (GLP) used to transfer natural language to a meaning representation language. In the second part we look into more detail at the Perception module that uses analogy to deal with underspecification and fuzziness in the representation. In the third part we discuss Gravital's node-based interface that transforms the representation into a network of building blocks that together generate visual output.

3 Graphics Language Processor

3.1 Why using natural language?

A user interface with programming code introduces an obstacle to people not trained in programming, such as artists and designers. Having to learn to write code steepens the learning curve and distracts from the problem at hand, however beneficial the future advantage. For these people it usually is more comfortable to say:

Let's draw some stars!

instead of:

```
for i in range(10):
    star(random(WIDTH), random(HEIGHT), random(20), 50, 20)
```

We can safely assume that few art projects start out with a procedural flowchart or a logical calculus. What we want is a system that can tie in with a real-life brainstorm: convert natural language into interesting ideas, visual sketches, augmenting incomplete linguistic input with thought-provoking design decisions. Supplementing computer graphics software with a language-processing unit leads to new opportunities.

At the start of an art or design project – establishing the look & feel of the printed documents, website and signage of a hospital for example – the primary ideas and concepts can be rather underspecified and schematic. For example: "it should look clean, calm and safe". A skilled designer will know how to handle this incomplete information based on past experience and training. But even a skilled designer usually needs several attempts before obtaining a satisfying result, as the creative process is based on trial and error. The transformation from

language space to visual space is a point where designers may express feelings of "being stuck" or "stumped for ideas". A system that produces variations of acceptable visual output based on linguistic input (what will *safe* look like and why?) can speed up the heuristic process and lead to new creative ideas.

For the sentence "It should look clean, calm and safe", the graphical representation is highly underspecified since the shape, colour, placement, rotation, etc. of the visual composition is left out of the description. The Graphics Language Processor (GLP) must be provided with routines to fill in the unspecified properties of the concepts evoked in the discourse, through the use of well-chosen and context-dependent default values. The default value for a certain property can be stochastically chosen from a range of appropriate values to preserve an amount of unexpectedness. The language-processing step offers the designer the possibility of obtaining a detailed sketch (and variations of the sketch) on the basis of the underspecified description the designer has given. This way, the system supports both the inexperienced designer by supplementing his input with well-chosen, contextually appropriate decisions, and the expert by producing unexpected variations on the basis of his input.

3.2 Memory-based shallow parser

The task of the natural language processing step is to transform information in natural language sentences to a meaning representation language, like First Order Predicate Calculus (FOPC) or a frame-based representation. This task has a long history in AI. In the seventies and eighties, knowledge representation and natural language processing were intimately connected disciplines in this effort (see Schank [1997] for an exponent of this approach).

Ideally, when using a suitable meaning representation language (a "language of thought"), it would be possible to represent the meaning of the sentence without redundancy, in an unambiguous manner, and with all possible inferences made explicit. In practice, the translation of natural language into a deep, completely unambiguous representation (i.e., understanding) turned out to be impossible, except for small domains where all relevant background knowledge needed for interpretation of the text was explicitly modelled. A good example of this situation is Winograd's natural language interface to the blocks world (Winograd [1972]). As robustness and domain-independence were put on the research agenda by funding agencies, the field of natural language processing has switched to robust, efficient and reasonably accurate text analysis methods that analyze text to a more superficial partially syntactic and partially semantic representation (shallow parsing), using machine learning and statisti-

cal methods trained on large annotated corpora. This robustness and efficiency comes however at a cost. Issues such as negation, quantifier scope, representation of time, and inference in general are ignored in this approach.

The shallow parser used by our system is MBSP for Python (De Smedt, Van Asch & Daelemans), a memory-based shallow parser (Daelemans & van den Bosch [2005]) that consists of a number of memory-based learning modules using the Machine Learning package TiMBL (Daelemans, Zavrel, van der Sloot & van den Bosch [2004]). Memory-Based Learning is a form of exemplar-based learning that is based on the idea that language processing involves specific exemplars of language use, stored in memory. Given new input to be processed, analogy-based reasoning is used to find similar exemplars in memory, and these are then used as models to extrapolate from. The approach is similar in spirit to usage-based models in cognitive linguistics (Croft & Cruse [2004]), and radical construction grammar (Croft [2001]).

Rule-based behaviour is considered a side effect of the way the exemplar-based reasoning works. MBSP consists of the following modules (apart from the tokeniser, all modules are exemplar-based systems):

- a tokeniser to split punctuation marks from words,
- a lemmatiser to find the base form of the words,
- a tagger to assign one part-of-speech (POS) tag to each word (Penn Treebank II tags are used),
- a chunker to find non-overlapping sequences of words, where words in a chunk closely belong together (i.e., they are phrases or constituents),
- a subject/object detector to determine which NP chunk is the subject or object of which verbal chunk,
- prepositional phrases (PP) are detected and attached.

In short, the shallow parser tries to detect relations between the words of a text, as can be seen in Figure 2. The shallow parser found two objects (the circles and the square) that have a relation with the verb. Inside each object block the shallow parser specifies the different relations using grammatical terminology. The parser is able to detect these based on the word and the context of a word it infers certain syntactical properties of that word using a large corpus of examples. Post-processing of the output of the shallow parser is responsible for some additional structural arrangements. An example is the indication of multiple elements conjoined by words like “and” and “or”. This is called conjunct identification (Agarwal & Boggess [1992]). In addition to the syntactical relations, the language processor also provides some semantic information about concepts in the domain. Gazetteers (wordlists) of the most common semantic categories in the graphical domain are incorporated in the system. These gazetteers will indi-

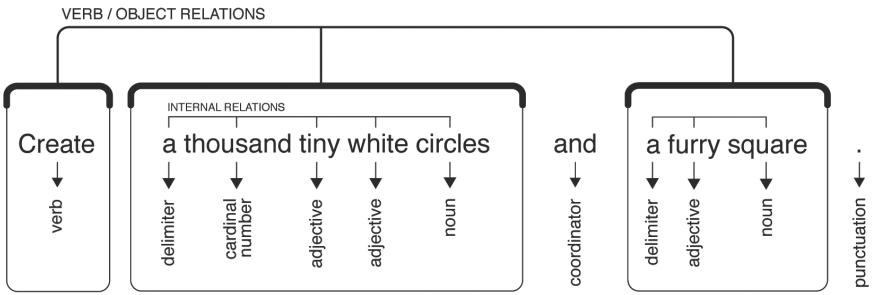

Figure 2: Output of the shallow parser.

cate if a word denotes a colour, size, position, etc. However, at the semantic level, the representation is very shallow, and has to be refined and further interpreted in a later stage.

3.3 Rule-based linker

The output from the parser is transformed to a directed network of linked concepts, so that the user's input (hinting at composition, proportion, shape, quantity, style, colour, geometry) can be explored by the system. The phrase "a blue circle" becomes a circle-concept linked to a blue-concept, which is a property of the circle. The clause "lots of transparent blue circles" results in a cluster with a circle-concept that has a blue-property (colour), a transparent-property (shade) and a lots-property (quantity). Connections between concepts are improved with a rule-based linker. For example, "a page with lots of transparent blue circles" triggers two linking rules. First, `"QUANTITY of NP: A is-property-of B"`, ensures that *lots* is attached as a quantifying property to the *transparent blue circle* prepositional phrase. Secondly, `"CANVAS with|of *: B is-part-of A"`, identifies the circles as being part of a page (i.e., the drawing canvas).

Rules mainly deal with correctly discerning between different types of prepositions (`PP`-attachments). For example, "above" indicates a visual arrangement between two peer concepts whereas "with" generally indicates a partitive parent-child relation between two concepts. Other rules can deal with transferring the verb-argument structure of the sentence. In "the page is cluttered", two concepts are discovered: page (canvas) and cluttered (composition). The rule `"CANVAS is COMPOSITION: B is-part-of A"` will discard the verb phrase (the copula adds no meaningful information to the model) and connect the cluttered-concept to the page-concept. Modality analysis (Morante & Sporleder [2012]) is used to

transfer more elaborate verb phrases (such as "could very well be") to a connection strength (e.g., "is" = 100%, "could very well be" = 70%). One drawback to this rule-based approach is the prospect of eventually expanding the system to a horrible tangle of coinciding little rules. However, it has the advantage that rules can be exposed to the user-interface in a comprehensible way. This allows users to customize the ruleset to fit a particular purpose on a need-now basis.

4 Perception

The Perception module is a database of what things look and feel like, which Gravital uses as a creative foundation when refining the output of the parser. It uses a deterministic and connectionist approach. Concepts are related to each other in a semantic network (Sowa [1987]). For example, a set of rules for a "rose" concept: rose is-a flower, red is-property-of rose, rose is-related-to romance, etc. Techniques from graph theory (Dijkstra's algorithm [1959], Brandes' betweenness centrality [2001]) are used to retrieve specific clusters of concepts and the paths between them.

The module has two distinct parts: an online visualiser (Figure 3) where new rules can be added to the semantic network and a toolset to retrieve clusters of concepts and analyze them. The online visualiser displays an interactive, force-directed graph inspired by the Visual Thesaurus application (Thinkmap [1998]).[2] It can be freely expanded by the participants in the NodeBox/Gravital community, with the aim of ensuring a diverse ruleset that does not slowly converge into a hand-tailored set bent on solving one specific problem. The purpose of the toolset then is to map unknown concepts in the GLP's output (e.g., rocket) to usable visual elements (shape, colour, composition). This is accomplished through concept analogy based on the perceptual properties of a concept. If a rocket is fast and hot, then what colours manifest speed and heat, and would therefore fit well in a visual artwork dealing with the aspects of rocketry? This kind of (elementary) associative framework can help designers and artists find a viable solution to so-called wicked problems (Buchanan [1992]): assignments in a social context with no clear solution. For example, an artist's sketches for the logo of a new space rocket can be rejected in many ways (e.g., the logo looks nonsensical because he or she is not a rocket scientist), but probably not on the ground that he employed a colour palette of red and yellow shades, because these

2 Visual Thesaurus, http://*www.visualthesaurus.com*

colours can be linked to concepts such as heat, sun, power, courage, etc. This design choice is a credible step towards a solution – finding a good rocket logo.

The Perception module's analogy algorithm uses four components: concept clusters, ranges, paths between properties, and the solver.

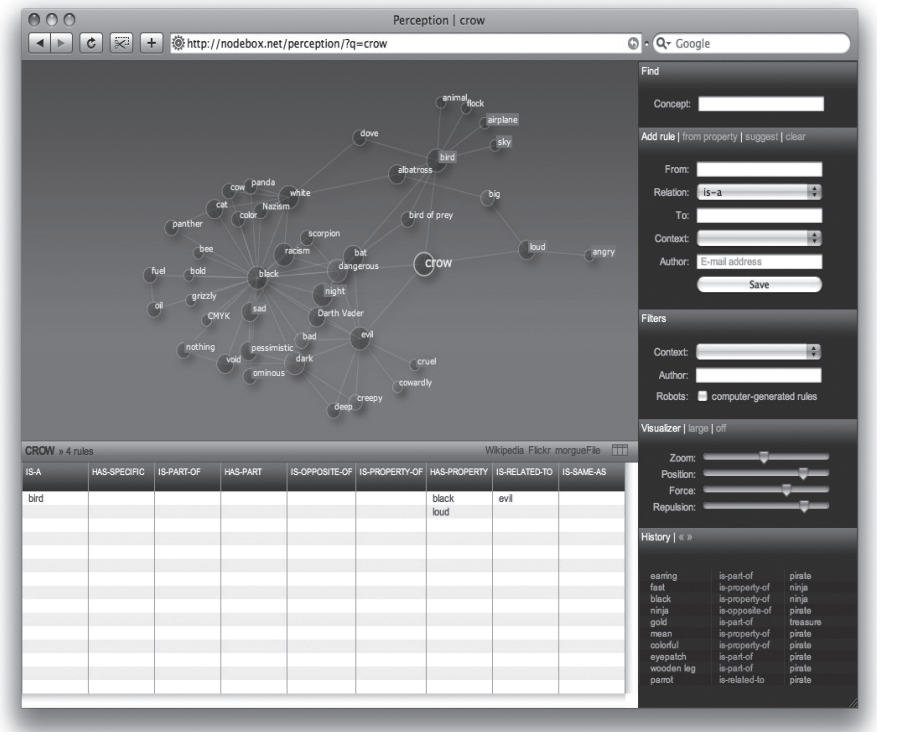

Figure 3: Perception screenshot, showing inferred relations between concepts.

4.1 Concepts as clusters of other concepts

Concepts are surrounded by related concepts that reinforce meaning. A concept cluster is the concept itself, its directly related concepts, concepts related to those concepts, and so on, as deep as the representation requires (depth 2 is fine in most cases). This is called *spreading activation* (Collins and Loftus [1975]; Norvig [1987]). When asked about music, people can readily come up with closely related concepts such as jazz, blues, saxophone, singer, but it takes longer to come up with less obvious concepts (for example, audio compression) because there are

more intermediary steps in the semantic chain. Activation spreads out from the starting concept in a gradient of decreasing relatedness.

Schema:

```
For a given concept:
1. Find all related concepts.
2. For each related concept, repeat 1. until depth = n.
```

4.2 Ranges of concepts by (implicit) type

Some concepts belong to the same family. Red, green, blue and yellow are all colours. The *colour*-range is comprised of dozens of concepts such as turquoise, teal, burnt umber, and so on – all the concepts with an implicit or explicit is-a relation to the *colour*-concept. What we call a range is also known as a semantic field, in the sense of Laurel J. Brinton: a set of words that share a common semantic property, "related to the concept of hyponymy, but more loosely defined" (Brinton [2000]). It is essential in the Perception module that we can easily work with fuzzy ranges instead of defining over and over which concepts exactly make up the range.

Schema:

```
For a given concept:
1. Recursively find all "is-a" relations.
2. The bottom concepts in this taxonomy make up the range.
```

4.3 Paths between properties

Some concepts are properties of other concepts. The Perception module focuses on such properties (i.e., adjectives). They describe what something looks or feels like. For example: romantic is-property-of France, fast is-property-of rocket, dark is-property-of evil, evil is-property-of Darth Vader, etc. A map of connections between each two properties is used to compare different concepts, based on the properties attached to a concept.

Schema:

```
1. Find all concepts A in "A is-property-of B" relations.
2. Find the shortest path between each two A, store it.
   (Dijkstra's algorithm)
```

Admittedly, a focus solely on properties makes it harder for the module to discover similarities of a cultural nature. According to the Perception module, an animal analogous to a sword would be a hedgehog, using the hedgehog's prickly spines as an associative bridge. But it cannot propose a dragon based on the observation that dragons and swords are closely related in fairy-tales. Some of our algorithms follow a predetermined approach, reminiscent of what Schank calls a script (Schank & Abelson [1977]), which describes the general sequence of events in a given situation, for example what happens when you enter a restaurant (enter–sit–order–eat–pay–leave). Or in the case of the Perception module: how to find links between concepts (by using properties: sword–sharp–prickly–hedgehog). The module will not deviate from this script, even though the semantic network contains everything that is needed to work with various other heuristics, such as using cultural associations to summon a dragon.

Historically, there have been two fundamentally different paradigms in AI: the symbolic and the connectionist (although today statistical approaches seem more popular). In the symbolic approach, logical sets of rules are employed for decision-making. In the connectionist approach, nonlinear dynamic systems are used (e.g., neural networks). For an overview of strengths and weaknesses in both approaches, see Chen and Ng (1999) and Toiviainen (2000). The Perception module uses a connectionist approach in some parts (concept clusters with spreading activation) and a symbolic approach in others (paths between properties). Our aim was to enable end-users to create their own decision-making kernels by modifying the symbolic rules, without a deep understanding of dynamic systems.

4.4 Perception solver: fluffy bunny, creepy crow

The solver uses clusters, ranges and paths to sort concepts by similarity based on the semantic chain of properties. For a given property (e.g., creepy) and a range of concepts (e.g., animals) it yields the concepts from the range that best reflects this property (the creepiest animals). In this particular example the solver will suggest such animals as octopus, bat, crow, locust, mayfly, termite, tick, amphibian, arachnid... No fluffy bunnies or frolicking ponies there!

In the case of the octopus the logic is obvious (and perhaps as disappointing as when you discover that the white rabbit had been hidden in the hat all that time): someone has added a *creepy is-property-of octopus* rule to the ruleset. The solver then simply takes the rule for granted and adds the octopus to the top of the list of suggested creepy animals. As more users relate creepiness to the octopus (i.e., more concurrence), it will move higher up the list. However, things

get a little bit more interesting when we examine the suggested bat or crow. No one has declared an *is-creepy* property for these animals. Instead, users tagged the bat concept with a *black* property and relations to *cave*, *evil*, *night* and *radar*. The crow concept was tagged with the properties *black* and *loud* and a relation to *evil*. So here the solver has implicitly inferred that there is a similarity between creepiness and the two birds instead of simply parroting what users said. In fact, the solver didn't even look at any of the tagged words mentioned above. Instead it judged that both the bat and the crow are *dark* things; and that dark is pretty creepy. Now where does *dark* come from? As mentioned previously, the Perception module works with clusters of concepts. For the bat, its direct relations to black, cave, evil, night and radar will lead further to other concepts such as Darth Vader, dark, dangerous, pessimistic, cat, airplane, sky, nothing, … even though no one has explicitly defined any kind of relation between bats and Darth Vader (someone did however add relations from black and evil to Darth Vader). All of these concepts together make up the bat-cluster. It is the "conceptual halo" that the solver takes into account when analyzing bats.

The idea of a concept cluster (or halo) is taken from Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought. Here, Douglas R. Hofstadter quotes the late nineteenth century William James:

> "There is no property ABSOLUTELY essential to one thing. The same property which figures as the essence of a thing on one occasion becomes a very inessential feature upon another. Now that I am writing, it is essential that I conceive my paper as a surface for inscription. […] But if I wished to light a fire, and no other materials were by, the essential way of conceiving the paper would be as a combustible material. […] The essence of a thing is that one of its properties which is so important for my interests that in comparison with it I may neglect the rest. […] The properties which are important vary from man to man and from hour to hour. […] Many objects of daily use – as paper, ink, butter, overcoat – have properties of such constant unwavering importance, and have such stereotyped names, that we end by believing that to conceive them in those ways is to conceive them in the only true way. Those are no truer ways of conceiving them than any others; there are only more frequently serviceable ways to us." (James [1890: 222–224])

Hofstadter goes on to argue that AI-representations of human high-level perception require a degree of flexibility, where objects and situations can be comprehended in many different ways, depending on the context. In this sense, a bat-cluster in the Perception module is no longer objectively seen as just a bat, but more agile as something dangerous that is flying in the sky, something that is reminiscent of a character named Darth Vader, that is pessimistic, feline, … This agility allows for a wider range of possible solutions when analyzing bats in different situations. The solver will inspect all the properties of the entire bat-cluster (dark, black, evil, negative, brown, sad, deep, bad, etc.) and measure each of their

distances to *creepy* using Dijkstra's shortest path algorithm. The shorter paths are preferred, imitating the same cognitive bias in human reasoning (Anderson [1985]). The total "shortness" score is an indicator of the bat's creepiness. It is important to note that different properties in a cluster have a higher or lower influence on the score. For the bat concept, the distance between *dark* and *creepy* is more salient than the distance between *deep* and *creepy*. This is because *dark* is more central in the cluster when we calculate the network's betweenness centrality (Brandes [2001]). More relations between concepts in the cluster pass through *dark*. We take *dark* as a sort of conceptual glue when reasoning about bats.

Schema:

```
Given a property p and a range of concepts R:
1. For a concept in R, find the cluster of related con-
   cepts.
2. Find properties with high betweenness in the cluster.
3. Add up the shortest path length from each property to p.
4. Low sum indicates a concept with high resemblance to p.
```

4.5 Perception analogies: Brussels, the toad

Suppose we want to create an advertisement image to promote Brussels, the capital of the European Union. How can the system effectively pick good colours, fitting shapes, striking images? Or to put it in terms of Veale, Feyaerts and Forceville's duality-seeking perspective: what image can be used that creatively "compresses multiple meanings into a single form", instead of simply lobbing `Brussels_centre.jpg` onto a poster?

The solver in the Perception module finds candidate concepts for a given property. The analogy algorithm is concerned with mapping a concept to another concept by shifting context. So, for example, what animal resembles Brussels? To accomplish a contextual shift the algorithm builds on the solver. It takes all the properties in the Brussels-cluster, mined from Google with a `"I think Brussels is *"` query: famous, easy, sumptuous, boring, bourgeois, beautiful, proud, green, ... For each of these the solver can then rank candidates from an analogy range (e.g., a range of animals). The candidate with the lowest total sum makes the best analogy. By default it will solve 20 properties in the cluster with a dampening effect of 90% for each consecutive property. To make it more "humane", we can adjust the dampener, say, by only looking at 8 relevant properties with a 70% cut-off. In this example, the winner is a toad (green, slow, safe,

slimy, calm, rigid, old). You'd better wear your lucky hat when presenting this idea to the EU marketing department!

Figure 4: A toad in a pond. (picture by Per H. Hagdahl)

As to whether it is a good analogy or a bad analogy: when Brussels Toad was presented at two conferences (TEDxBrussels [2009], CLIF [2009]), reactions varied from amusement to mild irritation. It may be vexing to people who are fond of Brussels' beautiful Art Nouveau architecture and the multicultural hustle and bustle. It may be amusing to people who perceive the city as a big traffic jam surrounded by office buildings. Likely, it is amusing because it is unexpected, and even a bit rude. Perhaps there are as many opinions as there are human individuals. But the point is this: the find evokes reaction, which, after all, is a principal goal of art and design.

On a final note, a problem arises when no rules pertaining to Brussels can be retrieved from the semantic network (as is the case). Even though one or two rules is enough to extrapolate a concept cluster, NO rules means no starting point. In this case we can train the ruleset on the fly using a web query. This is inspired by the simile technique (`"as * as Brussels"`) described by Tony Veale and Yanfen Hao (2007).

To summarize then, the Perception module uses properties as a bridge to translate concepts into other concepts that can be the basis for creative visual output. This kind of associative fluency has been linked with human creativity (Guilford [1967]), as is the capacity to turn remote associations into a useful solution (Mednick [1962]). In this sense the module offers a creative leverage to people with little expertise in the artistic domain.

Schema:

```
Given a concept and a range:
1. Find the cluster of concepts related to the given concept.
2. Find properties with a high betweenness in the cluster.
3. Solve the range for each property, tally the scores.
5. Low total sum indicate good candidate concept.
```

5 Node-based interface

The front-end of the system is a node-based interface grounded in a procedural paradigm (Dataflow process networks, Lee & Parks [1995]) where each operation is represented by a block (or node) that "does something". A node functions as a generator that creates elements (e.g., an ellipse node) or as a filter that changes incoming elements (e.g., a rotate node). However, a node has no fixed purpose, or rather it has many purposes that depend on the other nodes attached to it (Figure 5). In the user interface, nodes can be connected together to form a directed acyclic graph. Creativity is playfully encouraged by allowing users to combine nodes in various ways.

The strength of a node-based approach is that every step in the design process is remembered. The system is explicitly designed to allow the construction history to be modified. For example, if we create a piece of text, convert it to vector outlines and then start changing the Bézier-curves, all of these operations will be represented as separate steps. Changing the original text is as easy as selecting the text node and typing something different while all the other actions are retained.

Users can pick and connect nodes from scratch or use natural language. The language analysis results in a selection of nodes that one can then adapt and expand. Internally, each node is a set of Python programming instructions. Nodes can be "opened" to dig in and adapt the source code. When new users (e.g., art students) start programming, we found that they have a hard time grasping what can be done with programming code. One of the reasons this is so difficult is that programming software usually does not provide mechanisms for discovering features; it provides a clean blank screen. The node-based layer alleviates this problem in three ways. First, since the nodes are the actions one can ask the system perform, listing them with descriptive icons provides a good overview of what kind of functionality is available. Second, when a user opens a node he or she can learn how it works. Altered nodes are stored as a copy of the original so there is no fear of messing up the source code.

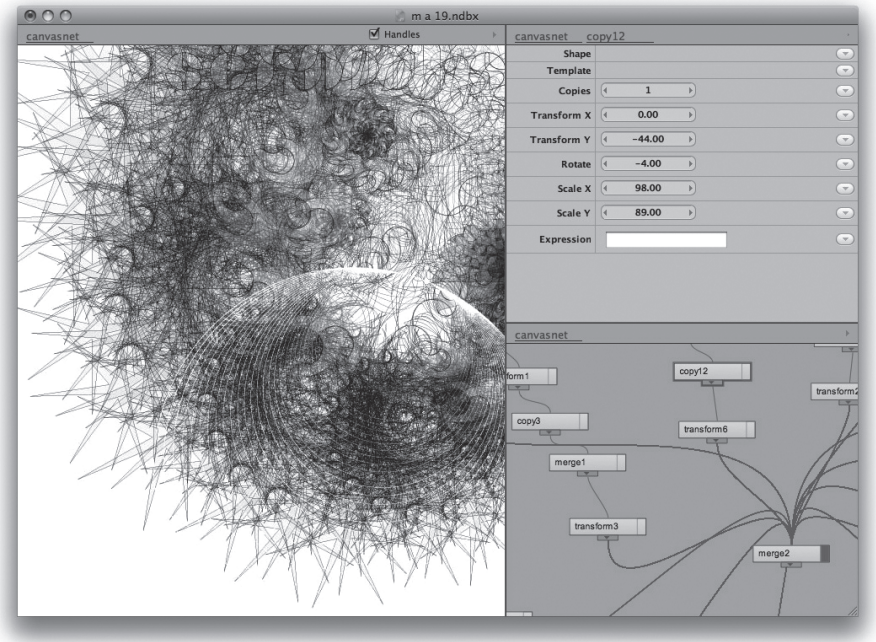

Figure 5: Prototype screenshot, showing generative artwork by Rokas Cicenas.

The third approach is the use of expressions. An expression is a concise programming instruction that creates an implicit connection between two nodes. Expressions can be compared to formulas in a spreadsheet program (such as Microsoft Excel). They provide a gradual learning curve between playing around with existing nodes and programming new nodes. The node-based interface allows users to experience the benefits of a structured approach towards design, but the real power still comes from programming code. Once users are comfortable with the use of expressions the step towards programming is a lot less steep.

5.1 An emergent node repository

Gravital is not limited to its built-in functionality but instead allows anyone to create his or her own nodes and publish them. Custom nodes are either created by combining existing nodes into a group or by writing the Python code from scratch. Nodes are written in a prototype-base style (Ungar & Smith [1987]), meaning that each existing node "recipe" can be the starting point (or prototype) of a new node. For example: a rectangle node can be easily cloned to a rectan-

gle-with-rounded-corners node or a square node. A node that draws a fountain could be an adaptation of a node that draws an explosion (which is a combination of vector and force nodes), and serve as the base for a more abstract node that expresses elegance, or sadness. We can crawl our way up from the bottom. To quote Hofstadter again:

> "Thus in each specific event, there is the germ of a whole class of similar events. This idea that there is generality in the specific is of far-reaching importance." (Gödel, Escher, Bach: an Eternal Golden Braid, pp. 358)

More nodes broaden the space of combinatory possibilities for both users and the language parser. It will be interesting to observe the social and creative dynamics once users start building and exchanging nodes.

6 Acknowledgements

Gravital was funded by the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT), and a consortium of 13 Belgian companies active in the creative industry. The project was a collaborative effort by the Experimental Media Research Group (Sint Lucas School of Arts, Antwerp, BE) and the CLiPS Computational Linguistics Group (University of Antwerp, BE).

7 References

Agarwal, Rajeev & Lois Boggess. 1992. A simple but useful approach to conjunct identification. In *Proceedings of the 30th annual meeting of the Association for Computational Linguistics*, Association for Computational Linguistics, 15–21.

Anderson, John. 1985. *Cognitive Psychology and its Implications*. New York: Freeman.

Boden, Margaret. 2003. *The Creative Mind: Myths and Mechanisms*. London: Abacus.

Boden, Margaret. 2006. What is generative art? *Digital Creativity*, 20: 21–46.

Brandes, Ulrik. 2001. A Faster Algorithm for Betweenness Centrality. *Journal of Mathematical Sociology,* 25: 163–177.

Brinton, Laurel. 2000. *The structure of modern English: a linguistic introduction*. Amsterdam: John Benjamins.

Buchanan, Richard. 1992. Wicked Problems in Design Thinking. *Design Issues*, 8(2): 5–21.

Chalmers, David, French, Robert & Douglas Hofstadter. 1991. High-Level Perception, Representation, and Analogy: A Critique of Artificial Intelligence Methodology. *Journal of Experimental & Theoretical Artificial Intelligence*, 4: 185–211.

Chen, Hsinchun & Tobun Ng. 1999. An algorithmic approach to concept exploration in a large knowledge network (automatic thesaurus consultation): Symbolic branch-and-bound search vs. connectionist Hopfield net activation. *Journal of the American Society for Information Science*, 46(5), 348–369.

Cleveland, Paul. 2004. Bound to technology – the telltale signs in print. *Design Studies*, 25(2): 113–153.

Collins, Allan & Elizabeth Loftus. 1975. A Spreading-Activation Theory of Semantic Processing. *Psychological Review*, 82(6): 407–428.

Croft, William & Alan Cruse. 2004. *Cognitive Linguistics*. Cambridge: Cambridge University Press.

Daelemans, Walter, Zavrel, Jakub, van der Sloot, Ko & Antal van den Bosch. 2004. Timbl: Tilburg memory-based learner. In *Technical Report ILK 04–02*, Tilburg University.

Daelemans, Walter & Antal van den Bosch. 2005. Memory-based language processing. In *Studies in Natural Language Processing*. Cambridge: Cambridge University Press.

De Smedt, Tom, Van Asch, Vincent and Walter Daelemans. 2010. Memory-based Shallow Parser for Python. *CLiPS Technical Report Series (CTRS)*, 2, University of Antwerp.

Dijkstra, Edgser. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1: 269–271.

Guilford, Joy. 1967. *The Nature of Human Intelligence*. New York: McGraw-Hill.

Lee, Edward and Thomas Parks. 1995. Dataflow process networks. In *Proceedings of the IEEE*, 773–799.

Mednick, Sarnoff. 1962. The Associative Basis of the Creative Process. *Psychological Review*, 69(3): 220–232.

Morante, Roser & Caroline Sporleder. 2012. Modality and Negation: An Introduction to the Special Issue. *Computational Linguistics*, 38(2): 223–260.

Norvig, Peter. 1987. Inference in Text Understanding. In *National Conference on Artificial Intelligence AAAI-87*, 561–565.

Schank, Roger & Robert Abelson. 1977. *Scripts, plans, goals, and understanding: an inquiry into human knowledge structures*. Hilldale, NJ: Erlbaum.

Sowa, John. 1987. Semantic Networks. In *Encyclopedia of Artificial Intelligence*. New York: Wiley.

Toiviainen, Petri. 2000. Symbolic AI Versus Connectionism in Music Research. In *Readings in Music and Artificial Intelligence*. Amsterdam: Harwood Academic Publishers.

Ungar, David & Randall Smith. 1987. Self: The Power of Simplicity. *Lisp and Symbolic Computation*, 4(3): 187–205.

Veale, Tony & Yanfen Hao. 2007. Learning to Understand Figurative Language: From Similes to Metaphors to Irony. In *Proceedings of CogSci 2007, the 29th Annual Meeting of the Cognitive Science Society*.

Winograd, Terry. 1972. *Understanding Natural Language*. New York: Academic Press.