A formal descriptive theory of software-based creative practice

Vincent Akkermans

Thesis submitted in partial fulfilment
of the requirements of the University of London
for the Degree of Doctor of Philosophy

Queen Mary University of London

2018

I, Vincent Akkermans, confirm that the research included within this thesis is my own work or that where it has been carried out in collaboration with, or supported by others, that this is duly acknowledged below and my contribution indicated. Previously published material is also acknowledged below.

I attest that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge break any UK law, infringe any third party's copyright or other Intellectual Property Right, or contain any confidential material.

I accept that the College has the right to use plagiarism detection software to check the electronic version of the thesis.

I confirm that this thesis has not been previously submitted for the award of a degree by this or any other university.

The copyright of this thesis rests with the author and no quotation from it or information derived from it may be published without the prior written consent of the author.

Signature:

Date:

Abstract

Creative artefacts, from concert posters to architectural plans, are often created in entirely software-based workflows. Software tools can be easily made to record all user interactions, thereby capturing the observable part of creative practice. Although recording software-based creative practice is easy, analysing it is much harder. This is especially true if one wishes to analyse the cognitive process that underlies the recorded creative practice. There are currently no clear methods for the analysis of recorded creative practice, nor are there any suitable theories of the cognition underlying creative practice that can serve as the basis for the development of such methods.

This thesis develops a formal descriptive theory of the cognition underlying software-based creative practice, with the aim of informing the development of analysis of recorded creative practice. The theory, called the Software-based Creative Practice Framework (SbCPF), fits with extended and predictive views of cognition. It characterises creative practice as a process of iteratively working from an abstract idea to a concrete artefact, whereby the required low-level detail to decide on action is imagined in flight, during practice. Furthermore, it argues that this iterative just-in-time imagination is necessary, because of the predictive nature of the mind.

The SbCPF was developed through the use of a novel method for the analysis of creative practice displayed in video tutorials. This method is based on Grounded Theory, Rhetorical Structure Theory, Gesture Theory, Category Theory, and a novel taxonomy describing the relation of action to speech.

The method is applied to produce a grounded theory of the creative practice of 3D modelling and animation with the Blender software. The grounded theory forms the basis of the aforementioned formal theory. Finally, the formal theory is further illustrated, evaluated, and explored by way of implementing a computational model.

Acknowledgements

I would like to thank my supervisor Prof. Geraint Wiggins for his continued support, reassurance, and inspiration. I expect that without it the PhD journey would not have been so rewarding.

I'd also like to thank everyone at the Computational Creativity Lab and the Media and Arts Technology programme. It is truly wonderful to have you as friends.

Special thanks go to Tom Haines and Peter Kemp who allowed me to work with them at 3Dami[1]. Even though the data set I recorded with them was not ultimately used for this thesis, it was instrumental in shaping my work.

I'd like to also thank Ton Roosendaal at the Blender Foundation[2] for working with me as part of the Google Summer of Code.

Thanks are also due to Queen Mary University of London, and the EPSRC Centre for Doctoral Training in Media and Arts Technology EP/G03723X/1, for financially supporting this work.

To Sophie I owe, without doubt, the most gratitude. Thank you for all your love and support.

[1] `http://www.3dami.org.uk/`
[2] `https://www.blender.org/`

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Creativity is often seen as a defining quality of human intelligence and it would be difficult to overstate the important role it plays in our lives. Its importance is not only reflected in the reverence with which we treat musicians and artists, but also in almost every detail of the world we have designed and built around us. Our creativity leads us to defiantly create order in an otherwise chaotic world and makes that our environment suits and pleases us. It is no wonder that creativity is a much studied phenomenon.

The workflows in creative industries, such as advertising and video game design, are often entirely software-based, meaning that the practical creation of artefacts involves only creative software tools like image editors. As a result, the observable part of the creative practice, that is the sequence of artefact-transforming interactions with the software, can be easily recorded, analysed, summarised, and presented.

This fact presents us with some exciting new opportunities. A creative software tool that records its own use would not only produce an artefact at the end of a session of use, but would package that artefact up with an explanation of how it was made. The session history could provide insight into the creator's underlying reasoning. What part of the artefact did they struggle with? Did they have a clear idea of what they wanted or were they experimenting? Such questions about a creator's internal reasoning are perhaps not ultimately knowable from just a recorded session history, but some information can surely be gleaned. A description of the creator's reasoning underlying a session of creative practice can be very valuable, even if it just serves to strike up a conversation between a teacher and a student, or between colleagues.

How would one go about analysing creative practice? Some early work for this thesis involved collecting a sizeable number of session histories (see Chapter C). The idea was that through experimentation a suitable analysis method would be found soon enough. Alas, with no sound theoretical basis the attempted analyses were distinctly uninspired and ad hoc. What was needed first was a model of the cognition underlying software-based creative practice. The work in this thesis started at the moment of that realisation.

1.1 Problem

There are ample models of creativity, but not many that are applicable to the cognition underlying software-based creative practice. Schön (1991) writes about how professionals, mostly in design professions, are engaged in a reflexive conversation with an uncertain situation when they design. That is, to solve a design problem is to try to understand the problem by iteratively attempting solutions and learning from those attempts. However insightful, Schön's

descriptions of practice do not consider the cognition of creativity, nor do they consider the use of software tools.

Boden (2004) and Wiggins (2006a) propose that creative cognition can be described as search in a conceptual space. Wiggins (2012b) and Clark (2015c) go a level deeper and explain creativity as a side effect of the brain's function as a predictive organ. Here the focus is clearly on the cognition of creativity, but creative practice is not considered in enough depth.

Other work focuses on cognition and practice, but not creativity. For example, embodied and extended notions of cognition argue that our bodies and our environment play such a crucial role in our cognition that they should really be seen as constituent parts of our cognition (Varela, E. Thompson, and Rosch 1993; Clark and Chalmers 1998). These notions allow us to approach creative practice from a different angle, but they are too abstract to deal with software-based creative practice directly.

Although all of these areas of work are relevant, none are quite right to serve as the foundations to build a method for analysing session histories on. Some do not consider practice as part of the creative process, while others focus on practice, but not creativity. However, to analyse session histories in terms of the creator's underlying cognitive process we need a model that bridges the gap between creativity in cognition and creativity in practice.

1.2 Objectives

The first and foremost objective of this thesis is to develop a descriptive theory of software-based creative practice involved in creating an artefact. It should shed light on the interplay between creative practice and creative thinking (if indeed these can be separated so easily) and the role of the software tool in creative cognition.

The theory should be formal in the sense that it should be expressed in a formal language. The precision inherent in formal expression aids in comparing against related models and in evaluation of the theory. Also, the theory should be developed from observations of creative practice and evaluated by way of implementation of a computational model.

The scope of the theory is to be limited to sessions of software-based creative practice that involve only one person and so group creativity should not be considered. It should only consider the part of the creative process that takes place when a creator uses a software tool. That is, the thinking, sketching, and discussing of a creative idea before the software-based creative practice starts is not to be considered here. The artefact that is created is assumed to be a digital artefact, such as an image, music track, or architectural plans.

1.3 Thesis outline

Chapter 1 We introduce the problem studied and give the wider context and rationale. The thesis objectives and contributions are listed.

Chapter 2 We review related work from research on creativity, design, artificial intelligence, cognition, and computational creativity. The focus here lies on work that is concerned with at least two of the three topics of creativity, cognition, and practice.

Chapter 3 We introduce a novel method for the analysis of video tutorials in which creative practice is displayed. The method facilitates the development of both a grounded and a formal

theory of creative practice. The subsequent chapters describe the results of applying the method.

Chapter 4 We present a grounded theory of creative practice, based on the study of Blender video tutorials. Blender is a 3D modelling and animation software[1] and the grounded theory is therefore largely specific to Blender and the 3D modelling and animation domain.

Chapter 5 Here we formalise the grounded theory from chapter 4 and abstract it away from Blender and 3D modelling. The resulting theory, named the Software-based Creative Practice Framework (SbCPF), is expressed in terms of Category Theory.

Chapter 6 We present an evaluation of the SbCPF by way of a computational model. The computational model is an implementation of an agent capable of using a very simple software tool. The computational model gives further insight into aspects of the SbCPF not previously considered.

Chapter 7 The final chapter is a discussion of the findings from chapters 4, 5, and 6 and compares the SbCPF against related work from chapter 2. Finally, we discuss future work and conclude.

1.4 Contributions

This thesis has the following major contributions:

1. A novel method to analyse video tutorials and produce grounded and formal theories.
   (a) Shows how Rhetorical Structure Theory can be used as a preprocessing step in within a Grounded Theory study.
   (b) Includes a novel Action-Speech Taxonomy that describes the possible relations between actions and speech in a tutorial video.
2. An in-depth grounded theory of creative practice in Blender.
   (a) Shows the existence of *content* and *construct* levels.
   (b) Highlights how goals are formed in-flight in moments of reflection and framing.
   (c) Finds patterns of interaction that can be detected in recorded creative practice.
3. The Software-based Creative Practice Framework is a formal descriptive theory of software-based creative practice.
   (a) The SbCPF considers both the creator and software tool to be part of the cognitive system.
   (b) Shows how an agent's predictions of tool actions are incorporated into reasoning.
   (c) Characterises creative practice as an iterative working from an abstract idea to a concrete artefact.
   (d) Shows how details of the goal artefact are imagined in flight and in a close coupling with the external artefact.
4. The computational model shows that limited prediction accuracy can be mitigated by the combination of imagination and practice.

[1] `https://www.blender.org`

1.4.1 Thesis statement

The formal descriptive theory of software-based creative practice that is put forward in this thesis is one that posits that the cognition of creativity cannot be understood properly without attributing to internal imagination and practice equal responsibilities. In creative practice a creator takes an abstract conceptual idea and works iteratively towards a concrete artefact. It is impossible to map a high-level abstract conceptual idea directly to low-level concrete details, for at abstract levels of description there is not enough information to predict those low-level details. Without such detail it is impossible to decide on the right concrete action and advance creation. It is therefore that these details have to be imagined in flight, during practice. In fact, the majority of imagination, and creativity, takes places in practice.

Chapter 2

Related work

2.1 Creativity and creative processes

When we speak of creativity or creative practice it is not immediately clear what is meant. Creativity is a phenomenon that is studied from many angles (Sternberg 1999; Kaufman and Sternberg 2010) and what the term denotes exactly depends on the field of study. We will briefly describe these different angles so that we are able to better specify the kind of creative practice we are interested in.

2.1.1 People, artefacts, cognition, and processes

An often cited definition of creativity is that of Boden, who has written extensively on the subject.

> Creativity is the ability to come up with ideas or artefacts that are new, surprising and valuable. (Boden 2004, p.1)

Boden views creativity as an emergent aspect of human intelligence and not as some special mental faculty. It is clear that in this definition the products of creative efforts can be physical artefacts, but also purely mental constructs. These products have to meet the requirements of being new, surprising, and valuable. On the surface new and surprising are qualities that are easily understood, but on deeper inspection they are not so obvious.

Boden realises that the judgement of whether an idea or artefact is creative lies in the interpretation of the beholder. This leads her to distinguish between two types of novelty: P-creative and H-creative. An idea or artefact is P-creative, where the P stands for psychological, when it is new to the person who comes up with it. It is H-creative, where the H stands for historical, if that same idea is new to everyone else as well.

Similarly, Boden distinguishes between different classes of surprise, whereby the type of surprise engendered by an idea or artefact is determined by the method by which it is produced. The first method is to combine familiar ideas to produce new ideas. For example, one could combine the concepts of "neon lights" and "windmill" and imagine an unusual new landscape. The surprise that results from this method is said to be characterised by the experience of statistical improbability.

The second and third methods rely on Boden's conceptual spaces, which are "structured styles of thought". Here Boden draws from the field of Artificial Intelligence (AI) to define conceptual spaces more precisely as search spaces, which are defined by a) a way of representing

objects in the search space, and b) a set of rules to navigate the space. The second method of producing novelty relies on navigating the conceptual space and finding new objects in it. It is important to note that the boundaries of the space are determined by how the objects are represented and by the rules of how the space can be navigated. Because the space is bounded, no new ideas or artefacts outside of these boundaries can be reached by navigation.

The third method is akin to changing the rules of the game. More precisely, it is the transformation of the way objects are represented, or of the rules by which the conceptual space is traversed. These transformations make it possible to think ideas that were previously impossible to think.

Finally, Boden leaves it up to the beholder to determine the value of an idea or artefact, as it is personal and dependent on context. For example, something can be judged as useful, aesthetically pleasing, inspiring, or revolting. However, if one did not require the quality of valuable then any nonsensical and random idea or artefact could be classified as creative on the grounds that it would be novel.

In Boden's AI inspired take on creativity we already see two uses of the words creativity and creative. Firstly, creativity is seen as a part of human intelligence. That is, it is a cognitive phenomenon. Secondly, the human ability of creativity results in creative artefacts and ideas that are novel, surprising, and valuable.

The work by Ward, Smith, and Finke (1999), like that of Boden, focuses on the cognitive aspects of creativity. Firstly, they point out that any human concept is inherently a creative object, as it is an abstract representation that would have had to be created at some point in time. In addition to being novel it is valuable in the sense that it is helpful in dealing with the world. Geneplore is an early descriptive model of how creative cognition might be described as having two phases: first a generative phase in which mental representations are either retrieved, combined, or newly formed, and second an exploratory phase in which those representations are interpreted and evaluated. Ward, Smith, and Finke highlight that this process can be seen as cyclical, whereby the representations are gradually refined. We will return to the notion of creativity in relation to cognition in Section 2.3.

Csikzentmihalyi rejects the notion that creativity happens solely in the mind, and posits that it is something that occurs in the "interaction between a person's thought and a sociocultural context" (Csikzentmihalyi 1997, p.23). This definition of creativity includes three parts: the domain, the field, and the individual. The domain is a particular domain of knowledge, such as mathematics, and can be seen as a set of symbolic rules and procedures. The knowledge of this domain is shared between people in a community and the field can be seen as a subset of this community. More precisely, the field is the set of individuals that decide whether an idea or artefact is worthy of inclusion in the domain; they are the gatekeepers. Finally, it is the individual who contributes new knowledge or artefacts to the domain. In this systemic model of creativity an idea or artefact is creative if it is produced by an individual and judged to be worthy of inclusion into a domain of knowledge. An idea, artefact, or person is creative if it has a certain impact such that it changes a domain or even results in the creation of a new domain. We see that Csikzentmihalyi applies the adjective creative to artefacts and to people and what qualifies them as such is the impact they have. That is, they have to be H-creative in Boden's terms. Above all, Csikzentmihalyi sees creativity as a social system.

In Csikzentmihalyi's work it is also natural to apply the adjective creative to people; there are those that are eminently creative and those that are not. This is closely related to the study of methods to measure a person's creativity in the field of psychology (Plucker and Renzulli 1999). Measuring creative ability allows finding other factors related to personality

and intelligence that are likely to co-occur. This so-called psychometric approach can also be seen in Kaufman and Beghetto (2009)'s four C model of creativity. The four C's (Big-C, little-C, mini-C, and Pro-C) are categories that highlight different aspects of creativity. The Big-C creativity category, originally proposed by Gardner (1993), is reserved for eminent creative contributions to a domain. The little-C creativity category, also originally proposed by Gardner, applies to everyday creativity such as arranging photos in a collage. Mini-C creativity is concerned with the novel interpretation of events, similar to how Ward, Smith, and Finke (1999) describe concept formation as creativity, and is therefore related to learning. The final Pro-C category is for people who have attained expertise at a professional level in a creative area.

Although the four categories are perhaps helpful as a descriptive framework they are not well-defined enough to use as an explanatory model. For one, the different categories contain different things: creative people, creative artefacts, and creative cognitive processes. For example, Big-C creativity is defined as a category of contributions where the contributions have had a particular impact. However, mini-C seems more concerned with the cognitive phenomena that produce creative insights and are important for learning. It is similarly unhelpful that Kaufman and Beghetto continually use the categories to classify people, rather than contributions, which raises the question of what exactly the categories are meant to represent. Also, Kaufman and Beghetto propose that Heisenberg would be an example of a physicist that has attained the Big-C status, but that Einstein is an example of a physicist who has attained a legendary status. In such a fashion one could make an infinite number of categories.

A more pragmatic approach is taken by Hewett et al. (2005) who give a simple definition of creativity that is similar to Boden's in that it underlines novelty and value. They also note that current research does not converge as to allow a clearer definition of the term. Instead, they propose to classify *research into creativity* along several dimensions. The first dimension is whether creativity is assumed to be a property of people, products, or cognitive processes. The second dimension is whether creativity is viewed as a personal or a societal phenomenon, much like the distinction between P-creativity and H-creativity or mini-c and Big-C creativity. The third dimension is whether creativity is thought of as frequent or rare and the fourth dimension is whether elements of creativity are domain-specific or general. The fifth dimension is whether creativity is seen as quantitative, for example how creative a person is, or qualitative, for example what type of creativity a person displays. The sixth and final dimension is whether creativity is seen as an individual or collaborative effort. This is a helpful model for understanding the differences between research into creativity, but it is important to note that the dimensions themselves are not dimensions of creativity as a phenomenon.

Shneiderman (2000) synthesises from much work on creativity to propose a distinction between three approaches to creativity. The first approach is that of what Shneiderman calls inspirationalists, which see creativity as a flash of insight after which much elaboration is required. Inspirationalists, of which Bono (1990) is an exemplar, assume that creativity can be improved by activities such as brainstorming and free association. The second approach is that of the structuralists, who emphasise the importance of domain expertise and methodical exploration of a problem space. The third approach is that of the situationalists, who emphasise the social aspect of creativity and is exemplified by Csikszentmihalyi. Although these distinctions do not necessarily shed light on creativity as a phenomenon, they do help to understand how people have historically regarded the murky concept of creativity.

Up until now the mentioned work has not fully acknowledged the creativity that groups can

exhibit. Prime examples of group creativity are music jam sessions and theatre improvisation. Sawyer (2003) identifies several key characteristics in which group creativity differs from individual creativity. Firstly, group creativity focuses on the process of creation rather than on the creation of a product. Specifically, Sawyer focuses on performances and sees the performance itself as the product. A key difference that follows from this is that there is not the opportunity to revise parts of the product, whereas revision is typically possible in other creative processes. Secondly, group creativity is unpredictable because those involved do not know what the others will bring to the process and, furthermore, they do not know how their own contributions will be interpreted. Related to this is the third characteristic of intersubjectivity, which refers to the fact that the meaning of a contribution to a group process cannot be determined until other participants have responded to it. Therefore, the creative process is seen as a joint activity of coordination of individual contributions. The fourth characteristic is that in group creativity the communication is complex due to the fact that the participants are both performing and negotiating their intersubjective understanding of that performance at the same time. The value of group creativity is emergent, meaning that is more than the sum of the parts. It is a property of the group, and not of the individuals. Other work that focuses on group creativity points out that there are other collaborative creative situations, such as joint music composition, where the performance is not the product (Nabavian and Bryan-Kinns 2006).

This overview of approaches to the study of creativity underlines the breadth of the research into the topic. Some aim to measure the creativity of artefacts, while others aim to measure it in people. There are those that see creativity as something that happens in a community, while others locate it in the human mind. There are others yet that investigate how creativity correlates to other personality traits, or how it emerges from groups of people collaborating.

As was set out in the introduction, our interest lies in creative practice specifically. In other words, we are interested in models of creative processes that include practically making some artefact. The next section takes a closer look at this aspect of creativity specifically.

2.1.2 Creative processes

Most of the authors mentioned up until now would agree with the proposition that the mind has some role in the conception and elaboration of a creative idea or artefact. However, they would have differing views on the interplay of mind and practice and how a practical creative process unfolds.

A large class of authors describe the creative process as consisting of a set of stages (Malinin (2016) presents a nice overview of such "process models"). This model of creativity originates with Wallas (1926) who describes four stages: preparation, incubation, illumination, and elaboration. A more recent example is Csikzentmihalyi (1997), who describes virtually the same process, consisting of the following steps: preparation, incubation, insight, evaluation, and elaboration. Preparation involves a person becoming immersed in a domain of knowledge and a set of problems. After this immersion one goes through a phase of incubation, which is when "ideas churn around below the threshold of consciousness" (Csikzentmihalyi 1997, p. 79). At some point there will be a sudden flash of insight and a creative idea will present itself. A person then evaluates that insight and decides whether to elaborate on idea or not. The final step, elaboration, consists of the realisation of the idea and requires the proverbial blood, sweat, and tears. Csikszentmihalyi adds that this view of the creative process should not be interpreted as being linear. Rather, it should be seen as recursive. For example, there can be many cycles of this process during an elaboration stage. He also notes that the duration

of the steps or the process as a whole varies greatly. This view of the creative process is of course compatible with Csikszentmihalyi's system model of creativity. That is, it focuses on the creativity of contributions to a domain and as such the creative process is the whole process of creating such a contribution. However, it does not explain exactly what the cognitive process of incubation is, nor what they typical structure of creative practice in the elaboration stage is.

Shneiderman (2000) proposes a view of the creative process that is based on Csikzentmihalyi's work. This view consists of dividing the process into four phases: collect, relate, create, and donate. In the *collect* phase a creator learns from existing works that are relevant in some way. In the *relate* phase one would consult with others. In the *create* phase the creator explores, makes, and evaluates possible solutions. Finally, in the *donate* phase the creator disseminates the results to the community. Shneiderman also emphasises that this is not a linear model and that a creator is at times required to return to an earlier phase. Also, the whole process of all four phases is cyclical, as the works that are donated to the community might be used in a collect phase by others.

These process models can be seen as descriptive models, rather than explanatory ones. That is, they take a top-down view of the process of creating an artefact and then recognise distinct phases in it. Because not all actual creative processes fit the distinct stages, the process models have to include the concept of recursion.

There is, however, a bottom-up approach to describing the creative process, which does result in explanatory models. It is perhaps best exemplified by Boden's work. The account of Boden's work presented earlier is already a model of a process. Her definition of the creative process as search in conceptual spaces aims to explain how the mind can produce ideas that are deemed creative. She draws heavily on knowledge from the field of AI and views the creative process as a computational process: an algorithm operating on representations. The algorithm navigates the search space and is guided by a heuristic, which directs the search in a particular direction.

Boden emphasises the importance of reflection. For example, she draws on the work of Karmiloff-Smith (1992) to emphasise the importance of meta-cognition in creativity. Through self-reflection the mind can re-describe knowledge that is implicitly present at a conscious level, thereby making it possible to explore a conceptual space at a meta-level. Boden mainly applies this concept to skills, noting that it is through this process that a person can explore their space of skills. Boden emphasises the importance of evaluation in this process, stating the following:

> The sort of consciousness that is essential for creativity, because it is involved in the very definition of the term, is self-reflective evaluation. A creative system must be able to ask, and answer, questions about its own ideas. (Boden 2004, p.295)

A common critique of Boden's model is that it does not explicate formally what conceptual spaces are nor exactly how they are traversed or transformed, as noted by Wiggins (2006a). Wiggins builds on Boden's model to do exactly this and proposes a framework that aims to be useful for the description, analysis, and comparison of creative systems, either natural or artificial.

Neither the process models, nor the models of creativity as search in a conceptual space, provide much insight in the role of practice in a creative process. The process models paint a picture of practice as execution of a plan, while the search models do not elaborate on the relation of creative thinking to practice. However, the ideas or artefacts in this world that are judged as creative are never the results of purely mental acts, but of an extensive interaction of mind and world (Gedenryd 1998). That is, even though we would like to pinpoint the origin of

creativity as lying in the mind, creative practice cannot exist without being in the world. This is a view that is more common in design research where models of the design process recognise the importance of interaction with the world more clearly.

2.2 Design and making sense of ill-defined problems

Several theories of creativity and the creative process have been described above, but none give a satisfactory account of the structure of creative practice. No creative artefact is the result of a purely mental creative act and only through a process of practically elaborating creative ideas can one shape materials into an artefact. What is exactly the relation between practice and creative cognition? Is practice just execution or does practice play a defining role in the cognition of creativity? Research into design typically looks more closely at these questions and views the creative or design process as an interaction between the design intent and the practical attempts at design solutions. This section introduces some work from the field of design research.

2.2.1 Schön's theory of reflective practice

The concept of reflection-in-action was developed by Schön (1991) in an attempt to give an account of the knowledge that professionals display in their practice. His theory is a reaction against the Technical Rationality view of knowledge, which consists of the belief that professional knowledge is inferior to knowledge resulting from basic or applied science.

Schön argues that this view of what practitioners do does not account for the knowledge that they display when solving real-world problems. That is, he believes that there is no set of knowledge that can provide ready-made diagnostic techniques and solutions for every possible problem. As a consequence, the belief that practice is the application of knowledge cannot hold.

Through close observation of professionals at work Schön developed the theory of reflective practice which characterised the knowledge of practitioners as a knowing-in-action. Part of this knowing-in-action is how problems are set, or framed, which is "the process by which we define the decision to be made, the ends to be achieved, the means which may be chosen". When one considers that many problem situations are unique and that each problem can be thought of in different ways, then it becomes clear that a set of generalised knowledge cannot not be the only thing that is required to come to a solution. Rather, there is a requirement for a process, and in this process is contained a knowing of how to go about shaping that process.

Schön further says of knowing-in-action that it is a form of tacit knowledge. A practitioner is not necessarily fully conscious of the actions, recognitions, and judgements that they carry out. In other words, their skills cannot be declaratively described, but can only be employed in action. Schön thus focuses on practice and the tacit forms of knowledge of practice. He goes on to describe how at times a practitioner's actions do not result in the expected outcome and therefore engender surprise. This surprise is what then causes the practitioner to reflect on "the outcomes of the action, the action itself, and the intuitive knowing implicit in the action". This allows them to reconsider their assumptions about the problem, how they framed it, and what new course of action to take. The concept of reflection-in-action is thus relatively simple: its core is the simple concept of act and reflect.

Perhaps the most valuable insight from Schön's work is that there is an underlying structure to the process with which any person deals with problems and goes about elaborating on

creative ideas, which he characterises as "a reflective conversation with a unique and uncertain situation".

If we juxtapose Schön's view of practice with process models of creativity (see Section 2.1.2) we see that there is a difference in scope. Whereas Csikzentmihalyi considers the process as a whole, Schön rather looks more closely at the moment to moment interactions. Although reflection-in-action does not aim to explain creative thought, it is similar to theories that do in that it is bottom-up. Whereas the macro-view theories of the creative process can only explain the micro-structure of practice through the use of recursion, reflection-in-action can only aim to explain the macro-structure by adopting the view that it emerges from practice.

2.2.2 Design Thinking

Design is typically seen as a holistic activity involving all aspects of a profession, for it is believed that, when it comes to designing, a reductionist approach to a complex problem cannot result in a full solution (Lawson 2012)[p.13]. The study of design therefore typically includes all types of activity, from sketching and drawing up plans to communicating with clients and manufacturers. Lawson (2006) as well as Darke (1979) describe early models of the design process that were developed from this macro-perspective. These models generally describe the process as consisting of a linear path through several stages, starting at the analysis of a problem, followed by synthesis of a solution, and finishing with its evaluation (we will call this the "analysis-synthesis" model). A similar linearity can be seen in the process models of creativity (see Section 2.1.2). A change in this linear view came when the conjecture-analysis model of design was introduced as a counter to the analysis-synthesis model (Hillier, Musgrove, and O'Sullivan 1972; Darke 1979). Over the course of many studies of designers at work a different picture emerged, where analysis and synthesis are not separate activities. Rather, designers analyse through synthesis, meaning that it is in their attempts to create solutions to the design problems that they come to know the problem better.

The dialogue between analysis and synthesis can perhaps be seen most clearly in sketching activity. Sketching has been a fruitful entrypoint for the study of design processes and the role of cognition in it (Goldschmidt 1991; Suwa and Tversky 1997; Verstijnen, Leeuwen, and Goldschmidt 1998; Goldschmidt 2003; Lawson 2012). Sketching and drawing are for many designers an integral part of the design process. In a review of studies of the role of sketching in the design process Purcell and Gero (1998) point out that as a designer works towards a final solution from initial ideas the sketches become more detailed and structured, indicating that their ideas are developing through the sketching process.

Design problems are characterised as being ill-defined (Simon 1977), meaning that the constraints imposed are not exhaustive and that they therefore do not constrain the problem space to a single solution. As such, designers will need to find a way to hone in on a solution, and the way sketching moves from ambiguous drawings to detailed ones provides a clue to how this is done (Goldschmidt 2003). As a sketch is completed it allows the designer to reflect on it and reinterpret it. This provides an opportunity to incorporate as yet unused knowledge from long term memory into the design process and to come up with new ideas. As another sketch is made, this cycle can occur again. Part of the reasoning behind the necessity of this dialogue with external representations is that design problems are too complex to solve purely mentally. Thus, external representations serve as memory aids as well as affording the generation of new ideas. Goldschmidt (2003) describes this quality of a work in progress as "backtalk" and she emphasises the importance of the human cognitive ability of representation.

Schön (1992) further highlights the role that perception has in the design process. He describes the general schema of the process as "seeing-moving-seeing". Seeing here is perceiving, both objectively (to see what is there) and subjectively (to judge what is there). Through seeing an intention of change is formed, which is effected through acting, or "moving". Subsequently, the designer sees the outcome of their move, which is either expected or unexpected, prompting further forming of intentions. Schön describes lucidly how an outcome of a move is not necessarily entirely expected or unexpected. A designer typically focuses on a small set of aspects of a design and intents to affect those. However, a move might change other aspects of the design to which the designer might not have been attending, and thus surprising the designer. Also, because of the complexity of a design, a move intended to change one aspect might cause emergent changes in all others. The point is that although these changes are frequently not intended, they are a vital characteristic of the design process. Schön writes:

> The sequential, conversational structure of her [the designer] *seeing-moving-seeing* enables her to manage complexity, and it harnesses the remarkable ability of humans to *recognize* more in the consequences of their moves than they have expected or described ahead of time.

Design thinking is often qualified as being a form of abductive reasoning (March 1984; Cross 2011). What is meant here is that the design solutions cannot be deduced or induced from the specification of the problem, but that there is an act of reasoning that simultaneously comes up with a hypothetical solution and an explanation of why the solution solves the problem.

Even though the interactive nature of the design process is now well understood, models of design thinking remain similar to the process models of creativity. They are still depicted as involving stages such as analysis, synthesis, and evaluation (Lawson 2006), similar to how they were drawn up a decade earlier (Candy and E. A. Edmonds 1994) and so they do not incorporate the cognition behind creative practice.

In sum, research into design highlights that the design process is not quite a sequence of stages, but a process that emerges from a proactive strategy of investigating a problem through attempting to create solutions for it. The cognitive capability of thinking of new solutions to try, and new ways of building them, is then what makes this strategy possible. Even though this is posited as the basic structure of design, it is recognised that it is not the complete picture, for there are a great many cognitive and practical skills that a designer will need as prerequisites (Lawson 2006, p.291). Furthermore, the question of how designers come up with new ideas is still open and it brings us squarely back to the question of creativity.

2.2.2.1 Engagement and reflection in writing

The interactive nature of practice that Schön wrote about is a core feature of design. It is, however, not only recognised in the prototypical design professions such as architecture. Sharples (1999) also recognises it in the process of writing.

Sharples relies on Boden's theory of conceptual spaces for an explanation of how new ideas are formed. An important aspect of Sharples' thinking is how constraints figure in the search for appropriate ideas. That is, concepts are subconsciously selected by constraints with respect the task at hand, the writer's audience, style, and the intent of the text, and so on and so forth. When an appropriate concept is found it comes into consciousness after which it can be written down. This view of the role of constraints is precarious, for it leads down the path towards creativity as an exhaustively constrained search for an optimal solution (Lawson 2006, p.76).

A noteworthy aspect of Sharples' work is the use of the concept of a Gestalt to explain how people are able to conceive of compositions, be it text or music, in their entirety without having to fully define all the details. He argues, from Gestalt psychology, that it is possible that the whole of a composition can be imagined without the need for imagining the individual parts. As such it would be possible to imagine abstract properties, and this does not necessitate imagining the individual words or notes.

Another important element of Sharples' work is the acknowledgement that writing is a practical activity that takes place not only in the mind. He does not solely mean that we cannot hold an entire text in our minds and that we therefore have to commit parts of it to an external medium, but also that through this externalisation the creative process is facilitated. That is, it facilitates exploration and elaboration and afford transformations that would be difficult or impossible to achieve without the help of external representations.

Sharples' model of the practical activity of writing consists of two levels. Firstly there is the level of interaction with the external medium. It is described as a cycle between short phases of engagement and reflection. In the engagement phase the writer gets the words down on paper and is, ideally, fully immersed in that activity. This means they devote all mental resources to transforming ideas in the mind into text and onto the medium. Engaged writing is followed by reflection. In this phase the writer rereads and interprets what has been written. By contemplating on the interpretation new ideas will form about what should or could happen with the text. A new phase of engaged writing starts once new ideas are clear enough. The reason that there is necessarily a cycle follows from the observation that engagement and reflection cannot happen concurrently. The length of each engagement phase is quite short and derives from the limits of working memory; it is about a sentence or a short paragraph. After each segment of engaged writing there is necessarily a phase of reflection, however short it might be.

The second level in the model describes phases of composition, revision, and planning, whereby a writer can move from one phase to any of the other two. The cycle between engagement and reflection as we have just described it would be an example of a phase of composition. In the composition step a writer generates text through engagement and reflection. The revising phase involves adopting an analytical stance and reviewing and annotating the generated material. In the planning phase a writer turns new ideas, which occurred during revision or composition, into broadly specified plans for the next composition phase. The cycle of engagement and reflection occurs not only in composition, but in revision and planning as well.

Sharples bases his work on various theories and findings from cognitive science and writers' own accounts. Pérez y Pérez (1999) formalises the model by implementing a software program. His efforts show that the engagement-reflection model can be made formal and that an implementation of it can produce stories.

2.2.2.2 Act and reflect

We have now seen several descriptions of the design process and creative practice as a cycle between action and reflection. Sharples (1999) brands it engagement and reflection, as well as composition, revision, and planning. Others focus on the dialectic natures of sketching and frame it as externalising and reinterpretation. Schön (1991) is perhaps the clearest in describing professional practice as a reflective conversation. Taking a step back we can be easily convinced of the validity of a model that stresses the dialectic between doing and seeing, for it is a natural

consequence of living in an uncertain world and having limited mental capacity that we have to go about navigating it interactively. That is, we have an idea, we act to externalise it, and then see and interpret the result and work from there. This is not a form of synthesising a solution and then validating it, but a way of understanding the problem. A crucial aspect of this cycle is that it allows to iteratively work from ill-defined problems to concrete and detailed solutions.

Although the models presented in this section are informed by cognitive science, they are not models of cognition. They describe aspects of creative practice, but do not explain how cognition brings them about. However, the past two decades have brought developments in cognitive science that can shed light on the cognitive underpinnings of the interactive views of creative practice. They are the subject of the next section, and they cast a different light on the nature of the action-reflection loop.

2.3 Creative practice and cognition

Organisms are typically said to be intelligent if they display behaviour indicating that they possess and reason with knowledge about the world and if they can make and execute plans to solve problems. Humans display a wide range of behaviour that is considered intelligent, while simpler animals are attributed a narrower range of intelligent behaviour. Creativity is often said to be a unique trait of the human intellect. Cognitive science is the field that attempts to understand how such behaviour, including creative practice, comes about. That is, it studies cognition: the process by which a system produces intelligence. In this section we introduce current strands of research into cognition and explore how they are relevant to creative practice as described in the previous section.

2.3.1 The scale of cognition

As we introduce different strands of cognition, we shall compare them mainly on two issues. The first is one of *scale*. Recent theoretical accounts of cognition posit that to understand cognition one has to first set the right scope. That is, cognition is a property of a system and the right scope is one that brings into view all the parts of the system that play a causal role. For example, early theories of cognition place cognition in the brain and therefore take the view that to understand cognition is to understand the role of the brain. More recent work looks at cognition at a different scale and posits that cognition is a property of a system where the brain is only a part and that, to gain a proper understanding, other parts, such as the body and tools, should be considered as well. The scale of things to take into scope is not only useful to differentiate theories of cognition, but also to properly locate the topic of this thesis.

The second issue is to do with how an agent internally represents information about the world that it operates in. The question typically revolves around what and how the world is represented in the brain and how representations are used in reasoning. Theories on one extreme end of this issue posit that the external world is viewed and internally reconstructed so that reasoning can take place without continuous access to the world. The other end is explicitly anti-representationalist and rejects the notion that the brain internally reconstructs entities that can stand in place of things in the world when reasoning. The issue of representations is particularly interesting when considering creative practice. When creating an artefact, to what degree does an artist or designer have an internal representation of that artefact before it actually exists?

2.3.2 Computational theory of mind

Computational theory of mind (CTM) is an early view of cognition that took the human mind to be an information processing system. In this view the brain is capable of computation and cognition is a computational phenomenon (Fodor and Pylyshyn 1988). Perception of the world serves to take information about the world and use it to build up internal representations of the world that the mind can then reason with. The product of such reasoning could, for example, be a plan of action to change the world in some way. Cognition is a continuing loop of perception-think-action whereby the key to understanding lies in the thinking. The scale of CTM is thus the brain itself and the mind is one that reasons with explicit and full representations of the external world.

There are parallels between early CTM and early models of the design process as analysis-synthesis (see Section 2.2.2). Both paint a picture of problem-solving as a process of internal reasoning. Information from the environment is gathered, a solution is arrived at, which is then attempted.

2.3.3 Embodied and extended cognition

Varela, E. Thompson, and Rosch (1993) took issue with CTM and held the view that cognition was shaped by aspects of the entire body in important and crucial ways. Cognition is thus not a property of the brain, they held, but one of a system comprised of brain plus body. The experiences of the world that are the basis of cognition are embodied experiences and cognition can only be properly understood by including the body as part of the cognitive system.

The enactive approach to embodied cognition goes one step further and highlights the importance of interaction with the world to bring about cognition. Not only does the body shape perception and therefore cognition, acting in the world through the body is part of cognition. Organisms take an active stance in shaping their environment. In doing so, they enact a world and their intelligence emerges in the interaction of those organisms with the world.

Extended cognition (Clark and Chalmers 1998; Clark 2008; Menary 2010) takes a similar view of cognition as the enactive embodied approach. It also emphasises the role of the body and interaction with the environment as constituent parts of cognition. More specifically, it posits that objects in the environment become part of the cognitive circuitry of the mind. The mind is extended into the world by coupling with objects, such as a mobile phone, in a cyborg-like manner.

The embodied, extended, and enactive approaches to cognition are all similar in that the scale of the system that is cognitive involves brain, body, and material environment. Rowlands (2010) sets out how these theses about cognition differ from each other exactly, but it will suffice to group them under the embodied moniker here. Embodied cognition emphasises that organisms use the world as part of their problem solving strategies. They recruit objects and tools in the world and part of the reasoning that takes place in the cognitive system is performed by the coupling of brain, body, and tool, within the environment.

The role of representations as described by CTM is called into question by the embodied approach to cognition. Reasoning is now not confined to the brain only, but it can be offloaded to the environment (Kirsh and Maglio 1994). For example, by trying things out in the world an organism can produce solutions without having to completely think them through internally. Representations still exist, but they play a different role, and they need to contain only enough knowledge to enable the organism to problem solve in the world. The closely related situated

and embedded notions of cognition (Brown, Collins, and Duguid 1989) even emphasise that the representations cannot be interpreted without being in-situ as they are coupled intimately with their contexts. Clark (1997) portrays representations as action-oriented, meaning that they are not action-neutral "copies" of the world, but rather descriptions of the world in such way that taking action is computationally cheap.

Embodied cognition fits well with theories of design that emphasise analysis-through-synthesis. For example, its take on the use of external representations and tools in design is that it is not just post-cognition elaboration, but part of cognition itself. The cognition of creative practice, according to embodied cognition, is brought about by a system comprising brain, body, and the tools and materials used in that practice. It also underlines that design itself can be understood in light of the designers enactive stance in the world. Firstly, the outcome of the design process is the shaping of the world and therefore the future cognitive process. Secondly, designers shape their environment (e.g. configuring their physical space and tools) within the design process. The embedded thesis of cognition highlights that the designer's representations often can only be understood within the context of practice, something that Schön called knowing-in-action. The embodied view of creative practice calls into question where the creative agency really lies. If the environment and tools are causally and crucially involved in creative practice, then their role in creativity should be duly pointed out, whether the artist consciously influenced them to some degree or not. However, notions of embodiment in cognition do not opine on the imaginative ability that is so crucial to creativity, nor the relation between that imagination and the embodied creative practice that turns imagined ideas into reality.

2.3.4 Distributed cognition

By taking another step up the scale ladder we find a strand of cognition research that focuses on the cognitive properties of assemblies of individuals and artefacts. This would fall outside of the scope of this work if not for the work that emphasises the effects of cultural practices on cognition (Hutchins 2008; Hutchins 2014). Hutchins posits that cultural practices, the learned way of being in the world, shape cognition in such an important way that certain properties of cognition should not be attributed to the individual, but to the social and built environment we inhabit. Hutchins conjectures also that cultural practices are not the result of our cognitive capacities, but rather the other way around: cultural practices produce new selection pressures that drive the development of cognitive capacities. The notion that cultural practice is a crucial part of cognition is extended to creative cognition by Malinin (2016), who provides a model of the interactions between a creative practitioner and the affordances perceived in relation to an architectural setting.

This work in distributed cognition looks at creative practice within the larger societal scale (much like Csikzentmihalyi, see Section 2.1.1) and is not directly useful for understanding the unfolding of a session of creative practice.

2.3.5 Anti-representationalist cognition

The anti-representationalist strands of cognitive science take the radical (Clark 1997) position that we need no such concept as representations to explain cognition. Instead, cognition is best understood as a system of tightly coupled interacting components (A. Chemero 2011). These systems are best described and explored through the use of tools from dynamical systems theory, such as differential equations. Notions of computation, the symbolic kind, and representation only serve to obfuscate. This represents the most radical break with CTM in that it claims that

whatever the brain is doing it is certainly not doing anything like building up representations that could be seen as a form of symbolic computing.

This radical stance has also been applied to the study of creative practice and tool use. Baber (2015) posits that tool use by craftworkers is best described as the selection and continuous control of parameters of the dynamical system in which craftworker, object, and tool are interlocked. The anti-representationalist stance tends to reject the notion of internal representations, and so Baber describes the craftworkers intention as being only "loosely defined". Baber does not so much explore the nature and content of the representations as reject that the representations are full specifications and plans for execution. Baber, Cengiz, and Parekh (2014) argue that while the high-level goal of a craftsperson might stay constant, the actual working out of that goal in practice needs very limited internal representations. Internal simulation is rejected also as it is speculated that trying things out in the world is simpler (Baber, T. Chemero, and Hall 2017).

There are some problems with the radical thesis. Firstly, not all actions can be cheaply tried or reversed. Marble is expensive you are likely to only get the opportunity to build an arc from the top down once. For example, Blender has just over 1,500 different commands that can be invoked with a countless number of parameter value combinations. Keyboard and mouse dynamics alone cannot explain goal-driven behaviour. Furthermore, tool use is seen as a dynamic coupling with the body at the kinematic level, and although this seems like a valid approach for a craftworker like a jeweller or sculptor, it is a less convincing approach for software-based practice. In such workflows the software user often uses a rather impoverished and discrete input devices with which they invoke any of a multitude of different commands. The command that is invoked is decided in large part by the contextual state of the software. In any case, it might turn out that the radical stance is the right explanatory model, but for now we need at least a higher level descriptive model that deals explicitly with a creator's intentions.

There is however an aspect of the anti-representationalist account of creative practice that fits well with the design descriptions set out in Section 2.2.2. Intents are not fully specified and designers do not have a concrete action plan before executing them. The question really is what the content of internal goal representations is and how it is represented. There are a great number of ways in which intent can be "loosely defined" and so this question deserves more attention. The radical approach does not bridge the gap between the high-level intentions and the activity that occurs in practice.

2.3.6 Predictive processing

Recent developments in cognitive science have taken the embodied and extended insights and backed them with a convincing explanatory theory (Clark 2013). This theory turns the typical CTM inside out. Its core claim is that sensory input to the brain does not get processed to ever-increasing levels of abstraction (e.g. from visual impulses at the retina to knowing which horse leads the race), but that it is the other way around. Higher level models predict what the sensory input should be, and the resulting error between what is expected and what the sensory input actually is then processed further. The bulk of information thus flows from the higher levels backwards, rather than forward from the sensors. The higher levels are seen as predictive models of the world that, over time, are tuned to minimize the prediction error. The predictive processing theory (PP) claims that brains are in the business of actively predicting the world, rather than passively describing it.

Each level of processing is a predictive model of the levels below. Higher levels deal with larger temporal and material scales. As one moves up the multilayered processing stack the levels therefore describe increasingly large invariants in the input. For example, lower levels deal with moment to moment expectations of sensory input, while higher levels deal with more constant concepts like identities of people.

The predictive model of processing is even extended to explain how action is caused by prediction itself. Low-level prediction errors that would result from a mismatch in prediction and sensory inputs drive motor action to cancel the prediction error. For example, expecting to see a cat but having none in sight would cause one to move their head so that the cat comes into sight.

Representations in PP certainly exist, but they are of a very different kind than in the original CTM. Each layer of the processing stack is a predictive model, at a larger temporal and spatial scale, of the inputs of the layers below. Higher levels thus describe the learned causal system, or latent factors, that explains the input to the lower levels. They do this as a probabilistic generative model, and so they do not predict a single possible input value, but the probability of a range of possible inputs.

One consequence of this view is that lower level perception is always contextualised by higher level predictions. Another is that representations are in a defined sense "action-ready": the cascade of high-level predictions all the way to sensorimotor states makes that there is a direct connection between high-level representations and being in the world. PP is therefore in line with embodied views of cognition.

Cognition is not seen as entirely model-based, nor entirely model-free, but as a mixture of strategies that take advantage of both the embodied nature and higher-level knowledge-based predictions (Clark 2015a). The on-the-fly deployment of this mixture is based on a notion of the prediction precision of each of the mixture components. PP thus posits that in being able to determine and use prediction precision the mind has the ability to employ meta-level knowledge.

Because the mind is adaptive in its use of different strategies, it can take a lazy approach to cognition. That is, if it is easier to engage the world than to internally simulate then it will. Such a strategy would only require representations in as far they provide a "grip" on the world (Clark 2015b). In other words, precision estimation allows the mind to incorporate the most reliable information, regardless of whether the source is external or internal (Clark 2017b).

2.3.7 Predictive processing and creativity

One criticism of PP is that it is not quite enough to explain human behaviours such as curiosity and creativity (Clark 2017a). PP is driven by minimizing prediction error and it is not immediately obvious why someone would not just seek a dark room and remain there to not encounter any prediction errors. This is counter to the curious and creative behaviour of humans that appears to seek out prediction errors intentionally.

Clark's answer to this apparent problem is based on the fact that the physiological state of an individual is part of the nervous system and therefore plays a role in the predictive processing (Clark 2017a). The state of the body therefore influences the perception of the world and the saliency of the affordances it offers. In a crude sense, an individual looks for food because it expects to be fed. Clark holds that this physiological link at least explains why organisms would explore the environment, as one expects exploration to lead to positive outcomes.

Another important factor, Clark holds (Clark 2015c; Clark 2017a), is the effect of our

cultural practices as described by Hutchins (see Section 2.3.4). That is, our cultural practices ensure that we are engaged in an ever-changing environment that is built and rebuilt over and over again, providing an evergreen supply of novelty to learn from.

Although it cannot be easily denied that the environment of the individual is like this, it focuses only on the consumption side of the equation. It is plausible that concept creation lies behind the prediction error minimizing thesis, but it does not lead immediately to a view of humans as creators, unless that is to be understood as an expression of the drive for exploration.

Elsewhere, Clark (2017a) explores the link between PP and creativity further. Prediction and imagination are not so far apart, it begins. The generation of sensor-like states by high-level generative models that plays a part in perception likely also plays a part in imagination. The individual would need to be able to surpress the prediction errors that would result from imagined states to not mistake them for prediction of perception. Clark further conjectures that spontaneous imagination of novel states could come about because of an inherent instability of the high-level generative models. This would cause different responses to low-level stimuli (or absence thereof). Not only would one see new things, but new ways of contextualising what's in front of them.

This is a convincing account that directly links concepts like spontaneous creativity and imagination with action and therefore practice. However, one element that is not worked through is how a creator would find an imagined state of affairs as an intention for practical pursuit, hold on to it while engaged in that pursuit, and judge the value of the changing world with regard to the imagined artefact throughout that process. It would seem an artist or designer would have to be able to willfully hold on to a subversion of reality, not as a hallucination, but as a parallel world, and in doing so start the flow of actions to make it a reality. PP does not provide, nor would it contradict, a cognitive architecture in which this is possible.

2.3.8 Spontaneous creativity

Wiggins (Wiggins 2012b; Wiggins 2012a) puts forward a more worked through theory for the cognition behind spontaneous creativity based on Baars' Global Workspace Theory (GWT) of consciousness (Baars 1988) and Shannon's information theory (Claude Elwood Shannon and Weaver 1948). GWT posits that there are many generators of information working in parallel to predict what will happen next in the world. If the information a generator produces is judged to be of high enough importance it comes into the Global Workspace, which implies that it comes into consciousness. Wiggins' theory aims to explain how exactly this importance is measured and judged. More importantly, with this extension of the GWT there is no need to search for an explanation of creativity. Rather, it is seen as a side effect of the mind's continuous activity of predicting the future. In an uneventful environment there is little input for the mind to predict, and as such the generators of predictions are free to come up with novel concepts. Furthermore, it allows us to distinguish between spontaneous creativity, which is the result of this cognitive prediction process, and planned methodical elaboration.

Wiggins' theory, like the PP approach, attributes sparks of internal creativity to the predictive nature of the brain. It is a more abstracted theory than PP in that it describes a cognitive architecture referring specifically to the role of, for example, consciousness and memory. However, where PP draws a direct link to action Wiggins' theory is not in itself embodied or extended. Although it is compatible with PP and its notions of embodiment, it is not concerned with placing itself within a larger framework that includes creative practice

as part of the creative process. For example, it is not directly clear how action follows from spontaneous creativity or how creative cognition is informed by ongoing action.

2.4 Artificial intelligence and computational creativity

The field of Artificial Intelligence (AI) is relevant to us for two reasons. Firstly, AI has long been concerned with studying systems that are capable of problem solving, either purely "mentally" or in the real world. Secondly, a subclass of AI research, called computational creativity (CC), focuses specifically on studying creative systems. We will briefly chart some relevant AI techniques that take the original symbolic computation approach. Then we will introduce the more recent developments that employ neural networks. After that, some relevant work from CC will be presented. Finally, we will give some special attention to works in AI that explain intrinsic motivation in information-theoretic terms.

2.4.1 Planning and reinforcement learning

The topic in AI that is relevant to us is that of learning, and planning, to act in an environment. Most early work on planning action is based on a symbolic approach to computation (Ghallab, Nau, and Traverso 2004). This comes down to designing the right representations so that certain search algorithms can find solutions in the resulting space. This includes temporal reasoning with temporal logic (Allen 1983; Allen 1984; Beek and Manchak 1996) and the event calculus (Kowalski and Sergot 1989; E. T. Mueller 2006). Although systems that use these techniques are widespread, for example in route planners, they are incompatible with modern views of cognition (see Section 2.3.6), nor do they have an answer to the question of how knowledge is learned in the first place. Planning techniques have been used to implement models of creative systems of which one good example is the Daydreaming agent by E. Mueller (1990).

Reinforcement learning (RL) is a class of AI techniques that is not concerned with logically planning action in an environment so much as learning behaviour from the environment directly. RL is a form of unsupervised machine learning as behaviour is learned only from the environment such that it maximises some reward metric. RL is therefore reminiscent of the behaviour of designers as observed in design research (see Section 2.2). Finding good solutions, according to some a priori metric, is a process of practically attempting action and learning from the outcomes.

Early forms of reinforcement learning operate in a space defined by manually designed representations and so are given a priori knowledge of the world. Techniques from deep learning have also been applied in the reinforcement learning context. Deep learning refers to the class of techniques that make it possible to train neural networks with multiple layers effectively, allowing representations to be learned from data (Yoshua Bengio 2009; Schmidhuber 2014; Goodfellow, Y. Bengio, and Courville 2016). A notable deep reinforcement learning (DRL) success was the recent highly publicised win of AlphaGo, against world-class professional Go player Lee Seedol (Silver et al. 2016). This showed the value of applying deep learning within a reinforcement learning approach. The AlphaGo architecture imagines and decides on potential future moves via a model-based search. Go is an environment where the rules are exact, but there are other environments for which precise models cannot be built and have to be learned. Examples of such environments that are often used in the evaluation of DRL systems include simple Atari video games (e.g. Pong and Space Invaders). Weber et al. (2017) introduce "imagination-augmented agents" that learn models of the environment through their interaction

with it and use that model to imagine potential trajectories. These trajectories are then used, in addition to outcomes of actual action, in learning a successful policy for its behaviour. Pascanu et al. (2017) introduce a similar "imagined-based planner" agent architecture, albeit with a stronger focus on meta-reasoning. It learns from its interaction with the world how to construct, evaluate, execute a plan, and when to switch between imagining and acting. In effect this system, as well as others (Wang et al. 2016; Duan et al. 2016), learn higher level behaviour enabling them to learn more effectively in particular tasks.

The problems to which DRL applies are similar to the problems solved by human creative practice. For one, DRL learns by acting in an environment, much like human designers learn about a design problem by attempting solutions. Also, some model-based DRL approaches incorporate something like imagination into their architectures, performing a similar simulation function as human imagination. On the other hand, the notion of goal-driven behaviour in DRL architectures is too simplistic to be a plausible model for creative practice. They pursue a given metric, such as the successful completion of a video game level, with a mindless relentlessness that is far removed from the intentional construction of an imagined artefact.

2.4.2 Computational creativity

Computational creativity is the subsection of AI that focuses specifically on the study of artificial systems that are capable of displaying creative behaviour (Colton and Wiggins 2012). This can be approached in two ways: engineering a creative system can be a goal in and of itself, or it can be a way of studying the cognition of creativity. The former tends to yield systems where the focus lies on the quality of the produced creative artefacts, whereby the cognitive plausibility is a secondary concern, if it is one at all (examples include Cohen (1995), Akkermans and Nispen Tot Pannerden (2008), Simon Colton (2012)). An example of the latter is the work by Wiggins (2012b) as discussed in Section 2.3.8.

Some work in CC takes an analytic philosophical approach to the study of creativity and puts forward formal models. The work by Wiggins on the Creative Systems Framework (CSF) is a prime example of a formal model of creativity. As the only formal model that also considers creative practice, it is the most relevant work from CC that we should consider.

2.4.2.1 The Creative Systems Framework

The CSF models creativity as search in a conceptual space as discussed in Section 2.1.2 (Wiggins 2006a; Wiggins 2006b). Forth, Wiggins, and McLean (2010) further develop the CSF by integrating it with Gärdenfors' theory of conceptual spaces (Gärdenfors 2004), thereby making precise how creative search can work at different levels of representation (i.e. symbolic, conceptual, and sub-conceptual). A. Kantosalo et al. (2014) employs the CSF to analyse human-computer co-creation scenarios, in which the computer shares in the creative responsibility. More recently, the CSF has been extended to include notions of creative practice. Specifically, Wiggins and Forth (2018) extend the model to live-coding of music (the creation of music in real-time by writing computer code). Because live-coding is a form of performance art, this extension of the model explicitly deals with creativity as part of creative practice. We describe the CSF in its original form first, after which we introduce the recent extension relevant to creative practice.

The CSF models a creative agent as a septuple $\langle \mathcal{U}, \mathcal{L}, [\![.]\!], \langle\langle ., ., . \rangle\rangle, \mathcal{R}, \mathcal{T}, \mathcal{E} \rangle$. $\mathcal{U}$ is the universe of possible concepts the agent might "imagine" and therefore corresponds to Boden's conceptual space. $\mathcal{L}$ is the language used to express the concepts from $\mathcal{U}$. $\mathcal{L}$ is also used to specify the rule

sets $\mathcal{R}$, $\mathcal{T}$, and $\mathcal{E}$. The rule set $\mathcal{R}$, as applied by the function generator $[\![.]\!]$, maps a concept $c \in \mathcal{U}$ to a real number that indicates how appropriate or relevant c is. It can thus be used to find the subset of $\mathcal{U}$ that are relevant by selecting all concepts above some threshold value. The rule set $\mathcal{E}$, as applied by $[\![.]\!]$, attributes *value* to a given concept. $\mathcal{E}$ differs from $\mathcal{R}$ in that it represents the preference an agent might have for one concept over the other, even though both concepts would be equally acceptable as judged by $\mathcal{R}$. The ruleset $\mathcal{T}$ specifies the traversal strategy for the conceptual space. The interpreter $\langle\langle ., ., . \rangle\rangle$ takes the rulesets $\mathcal{R}$, $\mathcal{T}$, and $\mathcal{E}$ and executes the search to find a new concept from a given starting concept. Outside of the search itself an agent is said to be capable of reflection on its own behaviour which can cause its rulesets to change. The universe $\mathcal{U}$ includes complete and partial concepts, including the entirely undefined concept $\top$.

The CSF is useful in describing and exploring, with precision, how possible agent rulesets compare. For example, different kinds of "uninspiration" (i.e. the search not finding suitable or valuable concepts) can be made precise by laying blame at the feet of the different rule sets, or interactions thereof. The concept of an "aberration" is defined as well: a member of $\mathcal{U}$ is an aberration if it is highly valued as per $\mathcal{E}$, but does not meet the requirements of being acceptable as per $\mathcal{R}$.

This brings us to the extension of the CSF to live coding and therefore creative practice. In live-coding the performer writes computer code which, when it is interpreted and executed, generates music. This makes that the practical activity of the live-coding agent consists of constructing syntactically valid computer programs, and the activity of the constructed program is the generation of musical events. The separation between the directly constructed artefact and the indirectly performed music is thus rather large, as opposed to an improvising pianist for example. Wiggins and Forth (2018) therefore propose to endow the agent with a personal conceptual space of computer programs $\mathcal{C}_\mathcal{P}$ as well as a conceptual space of music $\mathcal{C}_\mathcal{M}$. The set of programs $\mathcal{C}_\mathcal{P}$ is a strict subset of the set of valid real-world programs $\mathcal{C}_{\mathcal{TP}}$, and $\mathcal{C}_\mathcal{M}$ is assumed to at least overlap with the set of musics $\mathcal{C}_{\mathcal{TM}}$ that are yielded by the programs $\mathcal{C}_{\mathcal{TP}}$. A program $p \in \mathcal{C}_{\mathcal{TP}}$ is mapped to a music $m \in \mathcal{C}_{\mathcal{TM}}$ by the semantics of the computer code.

When live-coding, a performer iteratively constructs the computer code, often starting from a blank screen. The performance is in fact made up from a sequence of computer programs, whereby the code, and therefore the music, is crafted over the course of the performance. Wiggins and Forth (2018) propose to model the sequence of programs as consecutive applications, by manner of $\langle\langle ., ., . \rangle\rangle$, of the traversal ruleset $\mathcal{T}_\mathcal{P}$ for the conceptual space $\mathcal{C}_\mathcal{P}$. The consecutive applications yield a sequence of pograms and therefore a possible performance.

The application of CSF to live-coding practice in this way is a reasonable blueprint for the construction of an artificial agent capable of live-coding. However, as a model of a human live-coder's creative practice it throws up some problems.

One way to interpret the proposal is that the agent imagines the performance up front, and then performs it on stage later. In that case the model falls short on three points. Firstly, the model does not describe the live-coder very favourably. The traversal rules dictate the sequence of programs that is to be the performance, taking into account the live-coder's preference rules for programs $\mathcal{E}_\mathcal{P}$ and their sense of acceptable programs $\mathcal{R}_\mathcal{P}$. That is, in the creative search the live-coder is not evaluating what would be the performance, but each of the successive states individually. Whatever happens to be the route to a highly valued final program is taken as the performance. The live-coder cannot evaluate the development of the program over the course of the performance, something that is hard to accept given that the live-coder is surely aware that the sequence of programs correspond to a single "music" and not a sequence of "musics".

In other words, the evaluation $\mathcal{E}_{\mathcal{P}}$ should really be over the sequence of programs and the search should be over the space of sequences of programs. This brings us to our second point. In this reading the model is not a model of creative practice at all. Instead, a performance is conceived of and prepared beforehand and then performed, or practically elaborated, at a later time. The influence of practice on the creative process is thus not dealt with and the embodied and extended nature of live-coding falls by the wayside. Thirdly, because the search is said to occur in the space of programs the musical preference does not come into play in evaluating and selecting programs. Perhaps $\mathcal{E}_{\mathcal{P}}$ is thought to include an evaluation of the musical properties of the music generated by a program, but that would make the split between the program and musical conceptual spaces unnecessary.

The second reading of the proposal is that the search in fact happens during the performance and that each program visited in the search is performed as it is visited. In this case the objections already stated still remain. Additionally, there is no provision in the model for any from the actual musical outcome to influence the search. This does not fit well with descriptions of practice from the fields of design research nor cognitive science. Perhaps most problematic is that there is no provision for the agent to have some sense of a program (or music) that they are working towards and the way in which they would like to get there. This would firstly require a reification of the sequence of programs into a conceptual space of sequences. However, that is problematic in and of itself in that the agent would only be able to think in terms of valid and complete programs, and we would thus be back at the first interpretation: the agent a priori conceiving of a fully defined and faultless sequence of computer programs.

The CSF would perhaps fit better within real-time performance practice as the mechanism by which the performer takes imaginative steps, restarting the search after every performative action.

2.4.2.2 Evaluating computationally creative systems

Computational creativity that focuses on engineering creative systems needs to ultimately be evaluated by the creativity of the artefacts that are produced. Several frameworks have been proposed as standard methods of evaluation (Agres, Forth, and Wiggins 2016; Ritchie 2007; Jordanous 2012; A. A. Kantosalo, Toivanen, Toivonen, et al. 2015). Because these focus on the creative qualities of the output, they are of limited value to assess models of creative practice.

2.4.3 Information theoretic notions of motivation

In the section on predictive processing (see Section 2.3.7) the intrinsic motivation of an agent, be that creative or not, was described in information theoretic terms. Intrinsic motivation is the internal drive of an organism to engage in behaviour and opposes extrinsic motivation, which relates to external rewards (Oudeyer and Kaplan 2009). Roughly, PP states that an agent explores because it expects to find food and other useful things. It is the expectation itself that causes the agent to actively explore. This motivating device is information theoretic in nature as it is the core principle of the expectation versus observation dynamic (Oudeyer, Kaplan, and Hafner 2007). There are however other views of the creative motivations of agents that all belong to the larger class of information-theoretic insights into biology (Polani 2009). Even though the work in this thesis does not directly deal with the question of artistic motivation it is still helpful for the discussion in the final chapter to include an overview of these views here.

Schmidhuber tackles the motivation problem by proposing that an agent, in the absence of direct reward signals from the environment, pursues sensory signals that aid it in compressing

its history of observed data (Schmidhuber 2006; Schmidhuber 2007). An agent is said to store all the sensory input it ever experiences. The agent develops, over time, a mechanism to compress this data. Certain new inputs will cause the agent's compression mechanism to advance so that it can compress its data history better. The agent is motivated by choosing sensory input that allows for improvements in its ability to compress its data history. Beauty lies in perceptions that cause increases in compression performance and creativity is the search for such beauty.

Schmidhuber explains the motivation for creative behaviour as search for a better ability to compress, or understand, the experienced world. This therefore satisfyingly hypothesises that the driving factor in, for example, scientific and artistic pursuit is the same. However, the compression mechanism theory does not provide a way for an agent to know where valuable sensory input might be located and it therefore would have no way to plan accordingly. Some meta-knowledge, much like PP's precision concept, would be required.

Another information theoretic theory of intrinsic motivation is that of Empowerment (Klyubin, Polani, and Nehaniv 2005; Salge, Glackin, and Polani 2014). Empowerment is a quantification of the degrees of freedom an agent has, and is aware of, in a given state. The options open to the agent are seen as a measure of how prepared the agent is to deal with the world, which is in turn a proxy for the fitness of the agent. An agent's motivation would thus be to maximise its empowerment. The empowerment maximisation motivation theory could be applied to creative behaviour if that behaviour is seen as an exploration of the world that enhances the agent's readiness. In a less abstract manner empowerment could come into play in creative practice as well. It could provide a heuristic that determines an agent's preference of one course of action over another. That is, if one course of action achieves intermediate artefact states that leave more options open, then that should be preferred over alternative courses of action that might yield the same final artefact.

As we mentioned already this thesis does not directly consider the question of creative motivation. Even though the concepts of maximising compression capability and empowerment are plausible motivating factors, they are not enough to understand the cognitive process underlying creative practice.

2.5 Creativity support tools

At the intersection of creativity research and software engineering lies a subfield of Human Computer Interaction (HCI) that specialises in Creativity Support Tools (CST). This field studies how creative processes can be supported and improved by CSTs. Its main two areas of focus are best practices for software design and methods for the analysis of software-based creative practices. Both areas will be briefly charted in this section.

Creativity Support Tools (CSTs) are software tools that aim to support creativity and allow "more people to be more creative more of the time" (Shneiderman 2000). The development of more and better CSTs is seen as important from both a cultural and an economic point of view (Shneiderman et al. 2006). Supporting creativity is usually seen as different from supporting usability. For example, whereas typical software is meant to support productivity, a CST might aim to support associative thinking. Therefore, the design space of CSTs and the methods that are suitable for their evaluation are different to those of typical productivity software. Candy and E. Edmonds (1997) similarly argue that software that is to support innovative design processes have different requirements than software that aims to support more well-defined

processes. Although designs for software tools can often be well motivated, they can at times actually detrimentally affect the design process and so the development of evaluation metrics has received notable attention (Hewett et al. 2005).

Shneiderman (2000) presents many suggestions for better interfaces, but the category of suggestions that is especially relevant here is the provision of support for reviewing and replaying session histories. Shneiderman recognises the importance of reflection to improve the quality of the creative work and to learn from the process. Therefore, he argues, recording the user interaction could be beneficial in several ways. Plaisant et al. (1999) also argue that recording and presenting history would be beneficial to users. They conjecture that learning histories encourage students to monitor themselves and their progress and that this will lead to having more productive learning experiences. Furthermore, these histories are imagined to be first-class objects, meaning that they could be shared with others as simply as would be the case with the artefact itself. As such, histories could be shared with peers or mentors who could then provide feedback. It is also hypothesised that mentors will be more sensitive to the problems students might be having if they can be more readily inferred from the learning histories. Shneiderman (2000) extends on the notion of history as first-class objects and sketches the possibility of creating repositories of usage interaction histories that can be queried based on the properties of those histories.

Grossman, Matejka, and Fitzmaurice (2010) follow Plaisant et al. in their view that there is much valuable information inherent in the process of creation that others could explore and learn from. They envision that the history of a document is carried within that document and therefore shared if the document is shared. The potential applications of user interaction data are broken down into four categories. Firstly, the design history could help support team members understand how a shared document came to be. Secondly, any user who downloads a public document could learn new techniques and new software features. Thirdly, documents that have their creation history embedded can be used as tutorials. Finally, a document would document itself; that is it could help the creator to remember how they went about creating it.

An entirely different application of user interaction history is brought forward by Matejka et al. (2009) who state that CSTs potentially have thousands of commands and that most users only use a fraction of them. Therefore, there is a problem of awareness of the functionality of a software and this is a learnability problem. Matejka et al. propose that the user interaction history be used to determine which other, unfamiliar functionality could benefit a user and could therefore be recommended to them.

Klemmer et al. (2002) emphasise that understanding design history can be important to designers who have forgotten how a document has come about or take over a project from another designer, for students who are learning software or techniques, and decision makers. Terry, Kay, et al. (2008) suggest that users might want to see a representation of the progression they go through in learning or using a software.

Mendels, Frens, and Overbeeke (2011) emphasise the importance of keeping good documentation of design processes so that lessons can be learned later on. However, this can be difficult when the time pressure is high and, therefore, support for documenting could be helpful. The authors point out that having a representation of the design process helps in discussions in which that process is a topic of discussion. For example, this is the case in meetings with instructors or any time when a progression report has to be given. The assertion that it is harder to keep good documentation as stress increases is supported by Raskin (2000) who points out that people typically focus more on a few things and become less aware of others as stress increases.

W. Li et al. (2011) see several potential uses of history data related to teaching users how to use software. The data could be used to profile expertise levels and with that knowledge learning resources could be adapted to the user. Furthermore, the key teachable moments could be identified as they occur and thus the right learning resources could be presented automatically and at the right time.

Many works propose, or even implement, software tools to aid in the reflection of session history (Plaisant et al. 1999; Heer et al. 2008; Shneiderman 2000; Terry and Mynatt 2002; Hewett et al. 2005; Chen, Wei, and Chang 2011; Nakamura and Igarashi 2008; Terry, Kay, et al. 2008; Matejka et al. 2009; Grossman, Matejka, and Fitzmaurice 2010). These tools typically allow the user to scroll through a timeline and thereby replay the sequence of document states as they occurred in the session of creative practice. Most of these do not perform any analysis of the session history, although there are notable exceptions.

Denning, Kerr, and Pellacini (2011) segment session history of mesh construction sessions in Blender by way of regular expressions that rewrite the sequence of user action into groups of actions. Although the approach seems an effective way of clustering related actions it is ad-hoc and it is not clear in what way the clustering relates to the user's cognitive process that underlies the session history. Nakamura and Igarashi (2008) and Heer et al. (2008) also offer a superficial rule-based approach to grouping user actions.

Matejka et al. (2009) and W. Li et al. (2011) analyse six months worth of session histories from 16,000 users of AutoCAD. They do so to recommend commands to users that they have not yet used, but that they might find useful given that other users with similar command usage patterns use those commands with some frequency. However, the method of analysis does not look at temporal patterns, nor does it focus on the underlying cognitive process of the AutoCAD usage.

Grossman, Matejka, and Fitzmaurice (2010) use a segmentation method for session history that is based on the insight that the moment a user saves a document they most likely have finished a significant chunk of work. This is supplemented with additional rules based on the number of desired chunks and periods of user inactivity. The segmentation method is by no means the focus of the study, and as such does not consider the underlying cognitive process of the creative practice.

Akers et al. (2009) analyses session histories to see how undo and deletion events can be used as indicators of usability problems. Although a usability problem is clearly a cognitive phonemon, this work is limited in scope.

Kong et al. (2012) use an edit-distance based metric to calculate the similarity between pairs of session histories. Although this does take into account the temporal nature of session histories, it also does not explain the session history in terms of the underlying cognition.

From this overview we must conclude that there is no work in CST that attempts to analyse creative practice in terms of the underlying cognitive process on the part of the user, or of the cognitive system comprised of tool and user if you will. Understanding, at least to some degree, the cognitive process that underlies the observed session histories would surely aid in providing users with better reflection abilities. For example, it would make it easier to highlight segments of creative practice where someone achieved a particular high-level goal, or where someone might have been struggling and ultimately abandoned a goal.

2.6 Wanted: formal theory of creative practice cognition

From this overview of related work it can be clearly seen that there are no complete formal theories of the cognition underlying creative practice.

The models from creativity research either do not deal with cognition or not with practice. The design research field has studied the interactive nature of creative practice, and has certainly also focused on cognitive aspects of, for example, sketching. However, no complete theories are proposed that are formal and explain the cognition underlying creative practice. There are valuable insight from cognitive science, where the formal predictive processing theory of cognition provides a convincing basis for creative practice. However, its account of creativity and creative practice is nascent and it is not clear how it would explain the goal directed creative behaviour of constructing some imagined artefact.

In the field of AI the deep reinforcement learning techniques have been used very successfully, but they are not based on human cognition and the state-of-the-art architectures are not geared towards constructing artefacts. From the field of computational creativity the Creative Systems Framework, as applied to live-coding, is the only theory that deals with creative practice, is formal, and could be taken as a descriptive theory of creative cognition. However, it has some fundamental problems and is therefore unsatisfactory.

Given this landscape of related work we can now properly situate this thesis. If we want to be able to analyse sessions of software-based creative practice in terms of the underlying cognitive process, we will first need a formal model of creative practice that is based on cognition. It should see creativity as an embodied phenomenon in which the imaginative capabilities of the brain couple with real world tools to practically explore the consequences of an initial idea. That initial idea is never fully defined, although the creator might have in mind, as per Sharples' account, some sort of Gestalt. The cognitive system comprised of brain, body, and environment acts as one to fully define the artefact in mind, by making it real.

The next chapter is the first step towards the development of such a theory. It introduces a method for studying creative practice from up close and extracting from it a descriptive grounded theory.

Chapter 3

Method

3.1 Overview

In this chapter we present a method that takes video tutorials as data and develops from them a theory of what happens in practical creative processes in the Blender 3D modelling and animation application. This theory is then formalised and abstracted such that it applies to other creative domains as well.

The basis of our approach is the Grounded Theory method, which we argue can be applied to video tutorials. We further use Rhetorical Structure Theory for analysis of speech in the video tutorials and investigate its use as a method of process segmentation. Also, we present the novel Action to Speech Relation Taxonomy (AST) for the annotation of the relations between actions, and gestures, to speech in video tutorials. The theory is formalised and abstracted with the use of Category Theory. The final step in the method is to validate the formal theory through a computational model.

It is not immediately obvious why it is a good idea to take video tutorials as the primary source of data for our study. Therefore, we start this chapter with a discussion of why we have chosen them as a source of data. We then introduce the foundations on which we build the method (Grounded Theory, Rhetorical Structure Theory, Gesturing in communication, and Category Theory). The subsequent section present a novel way of preprocessing the video tutorials with the help of Rhetorical Structure Theory and a new action to speech relations taxonomy. We then move on to how Grounded Theory and Category Theory are applied to produce both a substantive informal theory and a formal theory.

3.2 Video tutorials as data

Tutorial videos are videos in which a tutor teaches viewers how to achieve a given task. One can find millions of tutorial videos on YouTube[1], on topics ranging from make-up to computer programming. We are here concerned with video tutorials in which tutors explain how to use a creative software application, specifically Blender, to achieve a given creative goal. For example, a tutor might demonstrate how to go about creating a 3D model for a cartoon character and discuss the practical and aesthetic considerations involved. Tutorial videos are a popular form of learning and there are many paid and free tutorials available on the World Wide Web for

[1] `https://www.youtube.com`

virtually every popular software[2].

The video in a tutorial is a recording of the tutor's computer screen and it displays the interactions of the tutor with the software. We can thus see the mouse pointer move across the screen, all interactions with the user interface, as well as any changes made to the artefact being created. The mouse pointer is sometimes used as a way to gesture and point at parts of the artefact or interface. It is worth noting that the view that the tutor and viewer have of the software is identical, allowing the tutor to demonstrate process in a natural setting.

Typically, the tutor cannot be seen, but they can be heard as they narrate what they are doing and why. Amongst other things, tutors will speak about the software application's features and interface, narrate the actions they are performing, and explain the technical and aesthetic reasons for them. The tutor's speech and actions are highly intertwined and synchronised as they demonstrate the practical process and explain the reasoning behind it.

For complex pieces of software, tutorial videos can be of varying degrees of difficulty. The simplest ones show how to take your first steps as a beginner and will focus on the tool at hand, while the more advanced ones comprise hours of footage and demonstrate the creation of a complex artefact from the ground up. Advanced tutorials are not so much concerned with the software tool itself, but rather with how to go about using the tool given the goal of creating a certain artefact. That is, the focus is on the application of techniques in relation to a creative goal.

From this description, we can see that the video tutorial is a rich medium and that both the video and audio modalities are vital to the tutor's communication. The fact that there is rich information does of course not necessarily entail that it is relevant or appropriate. So, what can we distill from these modalities?

Because the tutor's screen is visible to us, we can follow the practical process they go through as it unfolds. What we see on screen is the sequence of actions that cause the artefact to form, interspersed with gestures that serve to communicate rather than change the artefact. We see what they create and which features of the software are used and when. The screen recording modality thus provides a very clear view of the observable part of a practical creative process.

The speech modality in turn gives us information about the concerns of the tutor and the concepts that play a part in their thought process. We will not claim that this is some sort of direct verbalisation of their cognitive process. However, we will initially take the tutors' statements at face value. For example, if a tutor says their immediate aim is to give the cartoon character a bigger nose we have little reason to doubt that this aim, and the activity that follows it, is correlated to their cognitive process.

The two modalities of video and speech give us a moment to moment view of creative practice in the tutorial. However, if we cannot qualify how these moments relate over time we will not be able to say much about the process of practice at all. From direct observation we could relate two actions if one follows another, or if they are concerned with the same part of the artefact being created. However, this would not say much about the structure of the underlying cognitive process that we are actually interested in. For example, if a creator's aesthetic judgement led to a reformulation of purpose, in turn leading to a series of changes in the artefact, we would find no evidence of this chain of reasoning in just the sequence of observable actions. These relations and the structure they form only exist within the mind. If we cannot glean this type of information from the action sequence, is there anything else that

[2]YouTube returns over 7 million results for the query "Photoshop tutorial", and over 4 million results for "Blender tutorial".

contains this information in the video tutorials?

Luckily, the combination of the speech and screen modalities are more than just the sum of their parts. Frequently tutors bookend narration of their actions with explanations of the purpose of the structures they have created or talk of aesthetic intent. Because their speech is coherent, we can relate these statements of purpose with the narration. Because the speech co-occurs with the actions in a highly synchronised fashion we should be able to connect actions to individual speech utterances, and utterances to each other, allowing us to interpret the sequence of observable actions within its cognitive structure. The structure of the whole process, binding speech and actions together, thus emerges from the coherence of the tutors' speech and the tight relation of speech to action.

Not all Blender tutorials are appropriate for inclusion in this study. For example, there is a category of tutorials that focuses on explaining a single feature of the Blender software in isolation. In these tutorials, the tutors walk through all of the options of the feature and show how these options affect the result. However, they are not likely to show any preceding actions, and therefore do not show the context in which natural use of the feature might arise. That is, there is no creative trajectory and therefore no creative process. These tutorials are often aimed at beginning Blender users.

Other tutorials that are unsuitable are those that are heavily edited. In these, a tutor might show short snippets of on-screen activity, while explaining phases of the process in the abstract. The artefact is more a prop to explain what they are talking about than the focus of attention. It is easy to identify when a tutorial is heavily edited, because it is clearly noticed when an artefact has changed without cause.

Tutorials that are suitable are those in which the focus lies on creating an artefact. Additionally, the entire process of construction must be visible as a continuous recording. These criteria are more likely to be met by tutorials aimed at viewers of an intermediate skill level. Tutors do not have to spend much time explaining software features and can focus on their practice, while explaining it.

We have gained the most insight from tutorials that have a relatively long duration (20 minutes or more), or that are segments of a series of tutorials. In these tutorials it is more likely that tutors are exposing processes that are representative of real ones because the artefacts being constructed are not just toy examples.

3.3 Foundations

3.3.1 Grounded Theory

In *The Discovery of Grounded Theory*, Glaser and Strauss (1967) presented Grounded Theory (GT) as a method of developing theory from data. It provides a systematic approach to the scientific activity of infering theory from observation. Through an iterative process of coding (marking small pieces of data and using them as codes), writing memos, and sampling data, an analyst steadily works from observation to theory and in doing so leaves verifiable traces that lead from the theoretical concepts back to the data that provided their basis. The data in turn is meant to be described and explained by the theoretical concept. It is in this sense that the generated theory is called grounded.

GT is described as a method for sociological research (Glaser and Strauss 1967, p.6) and its most frequent use is in that field. In the description of GT that follows we lift it from sociology and adapt it to our own needs. We will argue that the structured approach to generating theory

from data that is advocated for sociological research is also applicable and useful to the inquiry into human behaviour of creative practice.

3.3.1.1 Constant comparative method

The basis of GT is the *constant comparative method*. It refers to the analytical process of constantly comparing new data with old, as well as comparing data to the theoretical concepts that are under development. This constant comparing allows the analyst to gradually develop theory (we will describe this process in more detail in the next section). The data can in principle be of any type, but it prototypically consists of interviews. The data become comparable because the analyst labels, or in GT terms, codes them. Codes are usually taken from the data themselves. For example, if a tutor says that it is important to "keep things simple", then that segment can be coded as "keeping things simple". Two segments become comparable by virtue of sharing the same code. The codes themselves suggest theoretical concepts. The dimensions of the concepts come to light by comparing the instances of the data that are linked to them. The analyst is able to develop the concepts, and the relations between them, from the data. As Glaser and Strauss write, it is through comparative analysis that theory is generated.

The constant comparative method is said to have three purposes, which are a) to provide evidence for, or refute, theoretical concepts, b) to establish the generality of a concept, or c) to specify a theoretical concept. We will describe each of these in more depth in a moment, but first it is important to note that, in GT, a theoretical concept is not necessarily a formally described exhaustive set of logical propositions as one would expect in other fields. Rather, they are seen as semantically flexible and, as such, there is room for them to develop and adapt to new data under consideration. This of course matches Popper's definition of pseudoscience (Popper 1959), for the theory cannot be tested if it adapts to any new data. However, the flexibility of the theoretical concepts exists while the theory develops. During this development the theory hopefully firms up to a state where it is verifiable. Even so, grounded theories are sometimes seen as substantive rather than formal, meaning that they are only to account for the data that was taken into consideration (Charmaz 2006, p.7).

As said before, purpose 'a' is to find evidence or to refute. In the process of generating theory one produces general concepts from observations. A new observation might then fit an existing concept, and thereby provide some evidence for it, or it might not fit, forcing the concept to be adapted or abandoned. If the concept is indeed problematic in light of the new observation then the concept is redefined by, for example, specifying different conditions. This primarily helps to develop the theory. However, if the conditions become so many that the theory does not generalise any more, then the concept has become useless.

This can work in the opposite direction as well, which is purpose 'b'. One can look to generalise a concept by checking if it is valid in situations that the concept's definition currently excludes or that have not been considered yet. If the concept fits observations from the new situation then that means the conditions can be relaxed and the generality of the concept is increased.

These are two cases of comparing data to existing theoretical concepts. The third purpose, 'c', is to compare between two or more observations, and to thereby form a new theoretical concept. The first step is for the analyst to rely on their interpretation of the data and to see that they are in some way similar. For example, in interviews different people might speak of the same kind of experience, or in video tutorials the same software feature might be used for

different purposes. By comparing the data the analyst can discover how they are the same and how they differ, thereby generating the necessary general theoretical concept and its dimensions. This is in fact the crux of GT: it provides the structure in which the human mind can create new theoretical concepts rigorously.

Although many Grounded Theorists would say that GT is an inductive method, a better description of it would be abduction (Reichertz 2009). That is, it is a mode of reasoning distinct from deduction and induction in that it involves the inference of the most likely explanation of what has been observed. In fact, the three purposes of the constant comparative method map nicely to these three modes of reasoning, whereby purpose 'a' fits with deduction, purpose 'b' with induction, and purpose 'c with abduction. Also, there is a pleasing parallel with the characterisation of design thinking as being abductive as described in Section 2.2.2; not only because it highlights the role of creativity in both design and scientific activity, but also because it allows the view that the way knowledge is produced in the design process is not all that different from how it happens in science: the creative mind lies at its core.

From this description we can see that theory is generated from data, and that it relies on the interpretation of that data by the analyst. That necessarily means that the GT process cannot be entirely objective. Thus, the use of the constant comparative method entails interpretation of the data by the analyst. The guidelines that are provided by Charmaz (2006, p.9) are meant to help interpret data in an open and rigorous manner. Here, open refers to the analyst having to take care to not use existing theories in the analysis of the data. It almost goes without saying that it impossible to interpret data without relying on experience, but it is possible to know when interpretations are speculative or come from existing theory. The practical process of the GT method affords to make interpretations explicit, which in turn affords the analyst to remain open to questioning their assumptions. In sum, the value of GT lies in the fact that it relies on the abductive mode of reasoning, and in the rigorous and constant way that it is guided.

3.3.1.2 The Grounded Theory method

The theoretical account of GT argues why it should work, but how does it work in practice? Charmaz (2006) gives a pragmatic account of the practical process of the Grounded Theory method. To start, the analyst will naturally approach a Grounded Theory project with a research interest. Even if this interest is still vague, it will rely on some existing concepts. Also, the analyst is not a blank slate, and they already bring certain assumptions and perspectives to the analysis of the data. Even though this initial framing will guide the analysis at the start of the process, there should be no blind commitment to them. Whenever a concept proves to not be useful, it can be simply ignored.

Collecting data With a research interest in mind the analyst sets out to collect data. The collected data should be rich, meaning that it consists of detailed descriptions of people's thoughts and actions. A primary example of this kind of data are interviews, but other kinds of data are not principally excluded, nor is quantitative data excluded from a GT study (Glaser 2008).

Coding After the initial data collection the next step in the analysis process is coding. This involves labelling short segments of the data, which are typically in textual form. The act of labelling a segment is to summarise that segment and to propose a category for it (Charmaz

2006, p. 45). By creating labels, the analyst makes explicit how they read the data. Although the data cannot be coded without interpreting them, the labels should stay close to the data. That is, the interpretative step has to be small, such that it can be said to have an objective quality. The analyst should code what they see, not what they think is behind what they see. On this topic Charmaz advices to look for actions and try to code the data using gerunds. This way the analyst can avoid the use of preconceived notions and concepts from existing theories.

Focused Coding After coding some data the constant comparative method comes into play when the analyst compares data to data. This partly happens as a result of being immersed in the data and coding subsequent segments. However, one can also explicitly compare data that lies further apart in some way. For example, the same code might appear in different interviews. The analyst takes the codes that seem most important and uses them to label larger segments of data. This process is called focused coding and it enables data to be summarised more abstractly. Because larger segments of data are coded with the same codes it also becomes easier to compare those segments and develop more insights. Charmaz (2006, p.60) writes the following.

> Through comparing data to data, we develop the focused code. Then we compare data to these codes, which helps to refine them.

In essence the analyst promotes codes to more abstract concepts. That is, whereas they were initially meant to represent small segments of data, they now represent larger segments and a larger number of them. By this process the codes gradually form better defined theoretical concepts.

Axial Coding After focused coding the analyst will have found the major concepts in their emerging theory. However, these concepts are not yet explicitly related to each other. Axial coding is the process of organising the relations between the focused codes. For example, the analyst determines what the major concepts are and what other concepts they subsume. In doing so the analyst formalises the concepts and brings together the results of the segmented analysis into a whole. Formalising here means making more explicit the specifics of the categories, rather than describing the theory in a formal language. Whereas initial and focused coding are an interpretative and abstracting process to generate concepts, axial coding is the process of integrating those concepts into a coherent system. Strauss and Corbin (1998, p.123) advocate the use of a specific analytical frame for axial coding, which involves asking specific questions about the conditions, dynamic qualities, and consequences of concepts occurring. In contrast, Charmaz does not find it necessary to apply such an analytical frame, and engages in the activity of producing an integration of concepts in a less formal manner. Because our focus is not a primarily social or experiential phenomenon we will not follow the analytical frame advocated by Strauss and Corbin.

Memo Writing While going through the process of coding the analyst will have tentative ideas about the concepts that are emerging. It is important to write them down and elaborate on them in memos (Charmaz 2006, p.72). Doing so will help the analyst think about and explore the data and codes. Whereas coding seeks to be more objective, memos are the place where the analyst allows themselves to be subjective, while recognising that they might change their view later on.

Written ideas are necessarily explicit. This provides an opportunity to investigate the vagueness that is inherently still present, which in turn leads to new questions and directions for the analysis. It is partly through answering these questions that observations are turned into theory. Also, memos can serve as explicit traces of how codes are developed into theory. The codes in turn are connected to the data, and as such there is a trace that leads from your theoretical concepts to the data.

Theoretical Sampling Through coding and memo-writing an analyst might find promising concepts, but they may also realise that they cannot fully describe them. Theoretical sampling refers to the focused data collection and analysis that occurs after these established concepts have been found to be underspecified. That is, one looks for data that can help in further specifying the dimensions of the concepts. More concretely, one samples data until no more new concepts or dimensions emerge. Where memo-writing leads to new questions as you try to make sense of data, theoretical sampling is the process of systematically answering these questions.

Saturation Data is coded and memos are written. The questions that follow from this lead to new data sampling and this in turn leads to more analysis. And so on and so forth until all concepts are fully specified. It is not easy to define when concepts are fully specified. Grounded Theorists consider saturation to be the stopping condition. Charmaz (2006, p.113) writes:

> Categories are "saturated" when gathering fresh data no longer sparks new theoretical insights, nor reveals new properties of these core theoretical categories.

Although we believe that this is a practical way of deciding when to stop, being convinced that there are no more theoretical insights to be had this of course does not make it so. Proving completeness, similar to proving correctness, would entail ensuring that no single context for a concept, in the domain of interest, now or in the future would provide new theoretical insight.

3.3.1.3 Grounded Theory for the study of video tutorials

As we have mentioned GT is typically used for the study of social phenomena. However, the questions we have asked are not social in nature, and the data we have set our eyes on primarily contain non-social phenomena (of course video tutorials can be construed as entirely social phenomena, but their social aspects are not our primary concern). This is somewhat at odds with the typical usage of GT, and we therefore have to show that GT is still appropriate.

In our description of the constant comparative analysis we have drawn the parallel to different modes of reasoning (see Section 3.3.1.1). The strength of the GT method lies in the application of these modes of reasoning within a rigorous structure, regardless of the type of data that they are applied to. The GT method can also be applied, therefore, to different types of data, such as tutorial videos. The constant comparative analysis is thus not restrained only to social phenomena, and we can in principle expect it to work outside of sociology. We have also said that the value of GT lies not only in reliance on abduction, but in the rigorous way it is guided. Having exposed that practical process we can make the case that it will also work for our interests and for video tutorials. We consider several of the GT steps that could be problematic.

Firstly, collecting data initially would not prove difficult, as video tutorials are available in abundance online. However, when collecting data as part of theoretical sampling we might

have more difficulty. If an analyst relies on interviews it is relatively easy to ask people directly about the topics they need more data on. It is likely to be much harder to find relevant data, or information contained in it, if that data is badly indexed, which is the case for video tutorials. However, although it is harder, it is not impossible. Also, there is no reason that we are confined to tutorial videos if we find them lacking in some way.

Secondly, the data that is typically considered when employing GT is such that it contains information about the topic of interest directly. For example, in an account of his PhD work (Furniss, Blandford, and Curzon 2011), Furniss describes that his research focus was to understand "why practitioners choose to use the usability evaluation methods they do", and so a GT approach was adopted and practitioners were interviewed about their choices. The conducted interviews were then direct descriptions of the topics of interest.

For our questions we cannot employ such a direct approach of interviewing creators about their practical creative process, for we would not expect people to be able to explain in full detail, or indeed accurately, an activity post-hoc. This is in fact one of the reasons why we looked towards video tutorials in the first place. However, video tutorials do not directly address the topics of our interest, but rather it is the behaviour displayed in them that is of interest. This has implications for the coding process. For example, if an interviewee addresses a topic, then the codes can be drawn directly from the text of the interview. In our case, the video tutorials are not already descriptions, and therefore it is necessary to make larger interpretative steps in the coding process. For example, if a tutor speaks of "keeping things simple", and we wish to apply the code "talking about strategy", then that code has not come directly from the data, but from our interpretation of it. To some degree this prevents the codes from being open to reinterpretation, because the codes are now interpretations already. Our response to this has been to keep the interpretations as objective as possible. We find this acceptable because a) the topics of interest are not experiential, and b) the video tutorials are direct recordings of the phenomena and as such we can describe them factually[3]. Although we import the word strategy in the summary example here, "talking about strategy" receives its meaning from the data coded with it, rather than from some imported concept of strategy.

3.3.2 Gesturing in communication

In analysing the videos, the analyst will quickly be confronted with the problem of distinguishing between on-screen actions that are part of the creative activity itself, and those that are part of the practice of narration and explanation. That is, we need to be able to make the distinction between actions that form the process of advancing the content, and those that are aids for communicating about that process. To this end, we here briefly describe some work on gesture theory.

3.3.2.1 Utterances: speech and gesture

Kendon (2004) and McNeill (2005) have laid out in considerable detail the different ways in which people use their bodies to communicate and how this mode of communication relates to speech. We can easily see how using the hands to point at objects can be classed as a communicative act. Other parts of the body, such as the head or torso, are also frequently used to gesture. However, not all bodily activity is considered gestural. After Kendon, we will consider gestures to be physical movements that are explicitly intended to convey information

[3]We should note that if our goal was to verify theory rather than generate it we would find it more important to measure intersubjectivity in the annotations of the data.

to others. In other words, even though we can consider every action to convey information, not every action is undertaken with the purpose of conveying information.

Typically, gestures go together with speech, and the two are often viewed as inseparable parts of an utterance. McNeill argues that they together embody underlying ideas. For him the tight integration and synchronisation of gesture and speech is evidence that thought is imagistic, and not only linguistic. Furthermore, Kendon (2004, p.7) defines an utterance as a unit of activity that is intended to provide information to other people and is composed of speech, bodily activity, or both.

3.3.2.2 Types and properties of gestures

Kendon (2004, p.84) gives a history of the study of gesture and the many classification schemes that have been proposed. He argues however that people gesture in so many distinct ways and contexts, and for so many expressive purposes that no single classification scheme can accurately capture this richness. Instead, Kendon advocates to study the dimensions of gestures, and to employ classification schemes as useful tools for particular investigations.

McNeill (2005, p.5) presents several such dimensions of gestures. In this model there are four dimensions: the relation of gesture to speech, the degree to which the gesture has linguistic properties, the degree to which the gesture is conventionalised, and the way the meaning of the gesture forms. McNeill explains the dimensions by placing several types of gestures along them. Of these types of gestures the most prominent type is the gesticulation, and it is the type that is of most interest here.

Gesticulations are defined as motions that "embody a meaning relatable to the accompanying speech". If we place the gesticulation type along the dimensions just mentioned we see that gesticulations a) are always accompanied by speech, b) have no linguistic properties (as opposed to, for example, sign language), c) are not conventionalised (as opposed to the thumbs up symbol), and finally d) their semiotic character is global and synthetic. The latter means that gesticulations have a meaning as a whole and that the meaning of their parts derive from the whole. This is opposite to a sentence of natural language where the meaning of the whole derives from its parts. A gesticulation is synthetic in the sense that it represents several things at once. An accompanying sentence would refer to those things in several parts. Not only are gesticulations accompanied by speech, they are also highly synchronised with it. That is, a gesture often co-occurs exactly with a part of the speech that is semantically related. McNeill further subdivides gesticulations into four non-mutually exclusive types, of which we will elaborate two: deictic and iconic gestures.

Deictic gestures are simply those that point at something. This is done typically with an extended index finger, but other parts of the body can be used. As the name implies, deictic gestures cannot be interpreted on their own, but only within the context of their performance. Kranstedt, Lücking, and Pfeiffer (2006) introduce the concept of the pointing cone, referring to the cone shape that is formed by extending the arm and moving the hand as if drawing a circle mid-air. The narrowness of the cone is then related to the specificity of the region being pointed to. Even though there is no 3D pointing cone to be formed in a 2D screen-based gesture space, we will show that the concept of the specificity being determined by the narrowness of the pointed to region is a useful one in interpreting deictic tutorial gestures performed with the mouse pointer.

Iconic gestures represent in some way a real entity or action. Imagine, for example, a person indicating the way a tennis ball spins if it is hit with a certain effect, or the shape of a vase

that they saw in a shop last week.

Deictic and iconic gesticulations are frequent in tutorial videos and the concepts of gesture introduced here will come in handy in Section 4.4.

3.3.2.3 Perceiving gestures

It is not immediately clear how people can distinguish between gesture and any other physical movement. It is, however, without question that we do. Without this ability we would not be able to interpret gestures. Kendon draws from a range of studies to posit the following.

> An action that is gestural has an immediate appearance of gesturalness. This means that a movement having this appearance will be discriminated and recognised as such directly. (Kendon 2004, p.15)

The recursiveness of this definition aside, the claim is simply that we perceive gestures without much effort. Although, it is not completely clear what qualities are meant by "gesturalness", there is much evidence that people are very capable of recognising gestures. In fact, it is often without any conscious effort that we interpret them. One gestural quality that is mentioned and will be useful to us is that gestures are excursions. An excursion is a physical movement that can be seen to start in a position and finish in that same position.

3.3.2.4 Gestures in tutorial videos

We cannot simply take what is known of gesturing and apply it directly to the rather impoverished communicative modality of a mouse pointer on a computer screen. Unlike face-to-face communication, in most video tutorials the tutor's hands, head, or torso cannot be seen. The mouse pointer and the interaction with Blender itself are the only ways in which the tutor could engage in gesturing activity. This begs the question if gesturing happens at all, and, if so, if we can tell them apart from other actions. We will show that gesturing does indeed happen and that it bears resemblance to the way in which people typically gesture with their bodies in face to face settings.

As said before, it is important that the analyst is able to recognise the difference between gestures and advancing actions, for if they were not they could not make the distinction between the practical creative process and communication about the process. The gestural theory described helps to sensitise the analyst to the properties of gestures, but it is not directly applicable to the relation of speech to non-gestural action. It is for this reason that we have developed a novel taxonomy of speech to action relations, which is presented in Section 3.4.1.3.

In our discussion of separating gesture from advancing action we have ignored the fact that a video tutorial as a whole is entirely a communicative act. In that light any of the tutor's actions can be seen as a gesture (according to our definition of gestures). However, even though the tutorials are demonstrations, they are demonstrations of how to achieve a certain hypothetical aim. That is, a tutor presents a process as if they had a goal. As viewers and as analysts we can adopt that hypothetical goal and view advancing actions as moving closer to that goal. In other words, we can maintain that within the context of a tutorial there is a distinction between gesture and advancing action. Furthermore, any normal action can be performed in such a way that it attains gestural qualities (Kendon 2004, p.7), but they remain only partly communicative. We can expect the same to be true for advancing actions. We thus have not a hard distinction between gestures and advancing actions, but a distinction between actions

that are purely gestural and actions that are part of the demonstrated process and that might to some degree be gestural.

3.3.3 Rhetorical Structure Theory

Grounded Theory helps an analyst investigate the tutorials with an open mind, but it is still a process of human interpretation. The video tutorial data is such that the analyst will have to look at it in very great detail. The topics that the tutor discusses, and the actions they perform from moment to moment, will be apparent to the analyst. This is helpful for understanding the practical creative process from a micro-level, but it does not readily reveal the macro-level. From watching a tutorial video, we can clearly understand what is happening from moment to moment, but it is hardly clear what the overarching structure of the process is. This is not only the case for creative practice; it is always difficult to see a process as a whole when one can only see a sliver of it at any given time.

How then do we get a handle on the structure of the practical creative process? This is where we look to Rhetorical Structure Theory (RST). RST posits that there is a hierarchic organisation in natural language texts. It brings this structure to the foreground by identifying the type of rhetorical relations that exist between segments of that text, such as sentences and paragraphs. RST posits that if there is no identifiable relation between two parts of a text then according to RST that text is incoherent, implying that it does not make sense. RST further specifies the form and types of text relations and how to go about annotating a text with those relations. Thus, RST is not only a theory, but a method to investigate the implicit propositions present in a text's structure (Mann and Thompson 1988, p.244).

RST is often used to analyse spoken or written discourse, but can also be a helpful model in the generation of text (Taboada and Mann 2006a). It has also been used, although not without extension, to the analysis of dialogue in tutorial settings (Taboada and Mann 2006a; Benwell 1999). However, we will not look any further into these extensions of RST as we are not dealing with dialogue.

How could RST be useful to the analysis of the structure of the practical creative process in video tutorials? We have to start with two assumptions. Firstly, we assume that a tutor's speech is largely coherent, for if it were not the tutorials would not make sense, and they would not be useful. Secondly, we also assume that the speech follows the practical process and that for the most part the speech will be concerned with the concurrent action. Thus, if the speech is coherent, and if the speech follows the practical creative process, then we have a good chance that the structure of the speech is related to the structure of the practical process. If RST can help us uncover the structure of the speech, then there is a good chance that we gain insight into the structure of the process.

The formal definition of RST involves the concepts of relations, schemas, schema applications, and structures (Mann and Thompson 1988, p.245). We will describe each in more detail below.

3.3.3.1 Relation definitions

RST revolves around the rhetorical relations that hold between two segments of a text. One of the two segments is defined as the nucleus and the other as the satellite. It is often the case that the nucleus can still be understood if the satellite is removed from the text. However, the satellite cannot stand on its own.

Name	Concession
Intention	The reader's positive regard for the nucleus is increased
Constraints	
on N	The writer has positive regard for the nucleus
on S	The writer is not claiming that the satellite does not hold
on N+S	The writer acknowledges a potential or apparent incompatibility between the nucleus and the satellite Recognizing the compatibility between the nucleus and the satellite increases the reader's positive regard for the nucleus

Table 3.2: The definition of the Concession rhetorical relation. The letters N and S stand for nucleus and satellite respectively. This definition was taken from the RST website[4].

Each relation between a nucleus and a satellite is of a specific type. The types of relations are defined by a) a name, b) constraints on the nucleus, satellite, and the combination of both, and c) the intention of the relation. When an analyst determines which kind of relation holds between the nucleus and satellite they have to evaluate for each defined relation whether in the case of the nucleus-satellite pair the constraints are satisfied and if it is plausible that the writer had the intention that is associated with the relation. It is important to note that it is specifically not the case that the intention can be inferred from the constraints being satisfied. Both are criteria that have to be judged to hold independently for the nucleus-satellite pair. For example, Table 3.2 provides the definition of the concession relation.

From the concession definition it is clear that the analyst has a interpretive role. Mann and Thompson (1988, p.246) call the judgements that the analyst is tasked with plausibility judgments. Any relation that an analyst finds is therefore a statement that begins with "it is plausible to the analyst that it was plausible to the author that" and ends with a statement such as "the reader's positive regard for the nucleus is increased" (in case of the concession relation type).

There are differing views on how many and which exact types of relations should be defined (Taboada and Mann 2006b, p.437). We have worked with the 32 relation types as presented on the RST website[5]. The definitions of these relations are given in the tables in Chapter A.

3.3.3.2 Schema definitions and applications

Because relations hold between a nucleus and a satellite they are binary relations. Although most relations follow the nucleus-satellite organisation, it is only one of several possible organisational schemas. Other schemas include the combination of two or more nuclei. These are so called multinuclear schemas and the relation that hold over the nuclei are called multinuclear relations. An example is the sequence relation. It holds between two or more ordered, but equally important units. A schema application is then the assertion by the analyst that certain units stand together according to one of the schemas.

3.3.3.3 Structure

We have said that schema applications apply to two or more units, but we have not yet specified what a unit is or how the units in a text are determined. Although Mann and Thompson qualify unit size as being arbitrary, they require units to have "independent functional integrity". Units are therefore typically clauses. That is, the units have to express a complete proposition.

[5] `http://www.sfu.ca/rst/`

Furthermore, the constituents of a schema application have to be adjacent. For example, the nucleus has to immediately follow the satellite or vice versa. The compositional unit that is formed by linking the nucleus and satellite can itself be part of another schema application as a new nucleus or satellite. Compositional units can be linked to other compositional units or to atomic units. Also, schema applications cannot overlap, meaning that each atomic or compositional unit can only directly be part of a single schema application. These constraints on structure make it so that an RST analysis organises a text into a tree structure.

3.3.3.4 Example

To illustrate the different elements described above we will here briefly look at an example relation. The text in Figure 3.1 is uttered by a tutor as he is evaluating the software feature which he has been explaining in the tutorial.

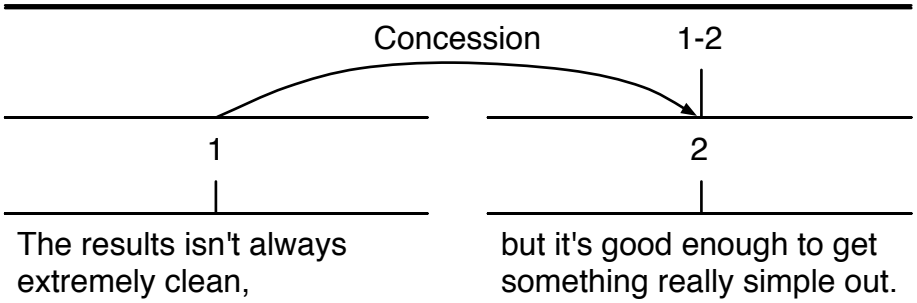


Figure 3.1: Two units marked 1 and 2 stand in a nucleus-satellite relation and together form the text-span 1-2. The relation that holds between the units is a concession relation. The diagram style is adopted from Mann and Thompson (1988).

Figure 3.1 also represents our analysis of this utterance. We here have a short span of text divided into two units. The first unit ("The result isn't always extremely clean") marked as text span 1 and is the satellite. The second unit ("but it's good enough to get something really simple out") is marked as text span 2 and is the nucleus. These two units thus stand together in a nucleus-satellite schema application, depicted by the arrow pointing to the nucleus. The compositional unit spanning 1-2 can itself be part of other schema applications.

The rhetorical relation is judged to be the concession relation. The intent for a concession relation on the part of the writer is for the reader to have a more positive regard for the nucleus. If we look at the constraints specified for the concession relation we can see that they are plausibly satisfied here. We have emphasised the constraints in the following few paragraphs. Note that the constraints refer to a writer, but that in our example we actually have a speaker.

Firstly, *the writer has to have a positive regard for the nucleus.* We can accept this because the tutor has spent an entire tutorial explaining the feature that he is referring to. Secondly, *the writer is not claiming that the satellite does not hold.* It is indeed true that the tutor admits that the feature is not perfect. Thirdly, *the writer acknowledges a potential or apparent incompatibility between the nucleus and the satellite.* The satellite refers to the quality of the feature not always being sufficient. The nucleus refers to the quality of the feature being sufficient for the cases where something simple is desired. Therefore, by stating both the apparent incompatibility is mentioned and resolved. That is, by acknowledging that the writer

knows the pros and cons of the feature we are more inclined to believe the pros that are presented.

3.3.3.5 RST and tutorial videos

As said before, we look to RST to uncover the structure of the video tutorials. RST is typically used for the analysis of written texts, whereas the speech in our video is more spontaneous. As said in our introduction of RST, as long as the speech is coherent we should be able to apply RST. Furthermore, our aim is not to study natural language or RST itself, but it is pragmatic: to attain a handle on the temporal structure of the practical creative process and its presentation.

3.3.4 Category Theory

Category theory is a branch of mathematics that can be used to investigate and describe formal structures. Typically, it is used to describe mathematical structures, but we will be using it as a way of formalising and representing our theory. Although category theory is a deep subject, we will here only touch on its basic constructs. With this basic introduction one should be able to interpret the diagram in Figure 5.4. Readers who appreciate the difference between sets and classes might prefer the introduction by Spivak (2014). The basic building blocks of category theory are categories, functors, and natural transformations. We will look at each in turn.

3.3.4.1 Categories

A category is quite simply a collection of objects of your choosing. You could for example define the category of tables in your local pub. Each object in the category could, for example, represent a specific table. Aside from objects, a category also has morphisms, which are links between the objects in that category. Links, as on the World Wide Web, point only in one direction, and not all objects are necessarily linked. Also, there might be more than one link between two given objects in a given direction. The morphisms that exist in a category is up to whomever is designing the category. Category Theory as a system of representation does not care what the objects are or what the morphisms mean.

Let us say the tables in our pub are arranged in a grid. A strange custom in our pub is that any patron may only ask for condiments from the tables directly to the left or to the right when seated (tables only have chairs on the long sides). We would like our morphisms to point from the table where the asking patron is seated to the tables that they may approach. If we were to draw the tables as objects and the condiments-rule as morphisms we would end up with diagram as depicted in Figure 3.2.

In the diagram we have named our morphism with lower case letters. Let us assume we are seated at table A and none of the tables have the mayonnaise we need for our chips, except for table B. We can follow the morphisms starting at either j or f. If the patrons at the other tables relay our request we can see that our request ultimately ends up at table B, from where the mayonnaise will hopefully be passed back. This chaining of morphisms on a path is called composition. With the concept of composition we can see that the two composed paths j, k, l and f, g, h are equivalent as they both map to the same table B.

One important concept to do with morphisms is that of the identity morphism, which is a morphism that maps from an object back to itself. In our example it would be asking the person opposite you at the table for the mayonnaise.

There could be any number of morphisms between two objects of a category, and there is a special notation to refer to the set of all morphisms between two objects. The set denoted by $\mathrm{Hom}_{\mathcal{C}}(A, B)$ is the set of all morphisms between object A and object B in category $\mathcal{C}$.

There is a lot more that can be said about categories and morphisms, but for the purpose of understanding the formal model in Figure 5.4, the above will suffice.

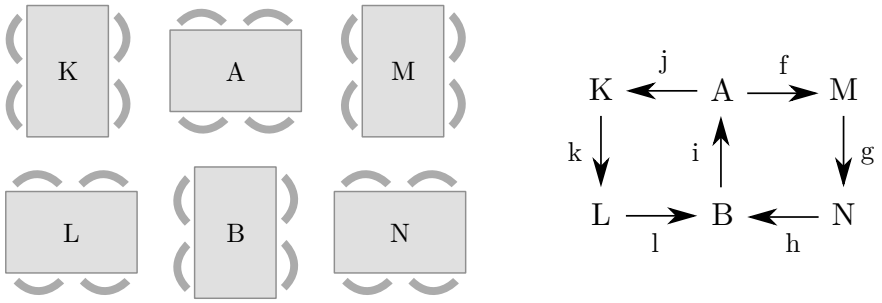


Figure 3.2: The tables in our pub are arranged like the diagram on the left indicates. A category representing the tables and the condiments-rule is shown on the right.

3.3.4.2 Functors

A functor is very similar to a morphism, but instead of mapping between objects within a category it maps from one category to another. If a functor F maps from category $\mathcal{A}$ to $\mathcal{B}$ then that functor must specify for every object in $\mathcal{A}$ to which object in $\mathcal{B}$ it will be mapped. Not only that, it must also pick a morphism in $\mathcal{B}$ for every morphism in $\mathcal{A}$.

There are two important rules to which a functor must adhere. Firstly, if our functor F maps any object X in category $\mathcal{C}$ to an object X' in category $\mathcal{C}'$, then F must map the identity morphism for X (i.e. the morphism mapping X to itself) to the identity morphism on X'. Secondly, functors must maintain the structure of morphisms under composition. That is, if we have a morphism q composed of first r and then s, and F sends q to q', r to r', and s to s', then it must hold that q' is composed of first r' and then s'. If we view the morphism as encoding the structure between the objects in a category, then these two rules force functors to maintain this structure in the translation of one category to another.

As an example, consider again our local pub. The six tables are waited on by two waiters, and so we will define a category Waiter with two objects W_1 and W_2, with identity morphisms and morphisms between them (see Figure 3.3). We can now define a functor F that maps between our two categories. For the objects, if a table is waited on by W_1 it is mapped to that waiter. For the morphisms, if a morphism maps between two tables waited on by W_1, then it is mapped to i_1 (the identity morphism on W_1). If a morphism maps from a table waited on by W_1 to a table waited on by W_2, then it is mapped to w_{12} (mapping from W_1 to W_2), and so on. F is a proper functor because it observes the two rules set out above.

As is the case for categories and morphisms, Category Theory puts no restrictions on what you want functors to mean. Your own functors may represent whatever is useful to you, as long as the defined functors follow the rules set out above.

Now that we know about functors we can introduce a special category called **Cat**, which is the category of all categories. The morphisms of **Cat** are themselves functors, for functors

map between categories.

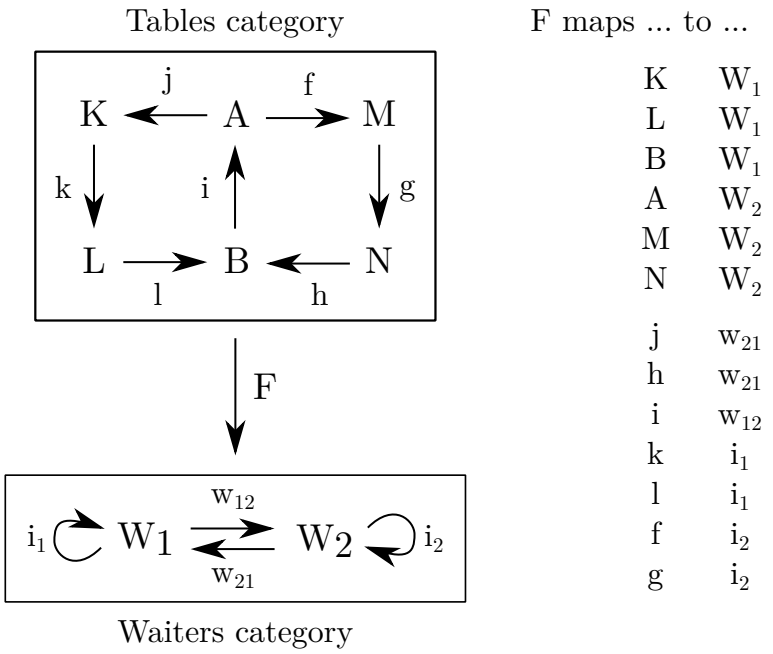


Figure 3.3: We now have two categories, Tables and Waiters. Our pub has only two waiters, W_1 and W_2. The functor F maps between Tables and Waiters as per the table on the right.

3.3.4.3 Natural transformations

Natural transformations are the third big topic of Category Theory. It is also one of the more complicated topics. Luckily we will not have to dive deeply into it. Let it suffice to say that natural transformations are maps between functors. For example, let F and G be two functors that map from category $\mathcal{C}$ to $\mathcal{D}$. A natural transformation n might map from F to G. Just like functors, natural transformations must maintain the structure of the morphisms in the underlying categories.

3.4 From preprocessing to validation

Having laid the foundations on which the method is built we can now dive into how they are put to use. As mentioned before, our aim is to work from our video tutorial data towards a formal theory of creative practice. The method we designed for this has four constituent stages: preprocessing of video tutorial data, application of Grounded Theory, formalisation with the help of Category Theory, and finally validation through computational modelling. Each of these stages will be described in depth in their own sections. We will, in addition, give a brief overview here to allow the reader to better situate themselves.

Firstly, the video tutorial data is preprocessed. This does not only involve the transcription of speech, but also the transcriptions of actions and gestures. Once transcribed the rhetorical structure of the speech is extracted, after which the relations between speech and actions are coded according to a novel taxonomy.

The second stage is the application of the Grounded Theory process, by which an informal, but substantive and descriptive theory is created. This follows a typical Grounded Theory process, with perhaps somewhat unusual input data.

Once the Grounded Theory has fully formed, the third stage of formalisation begins. With the help of Category Theory as a system of representation the informal grounded theory is moulded and shaped into a formal structure.

Finally, this formal theory is now such that it can be taken as a basic design for a computational model. An instantiation of it as a software implementation that closely follows the design of the formal theory can be taken as validation of at least its internal logic. Behaviour displayed by the instantiation and aligning with that observed in the video tutorials goes towards providing initial evidence of the formal theory.

The method is presented in diagrammatic form in Figure 3.4.

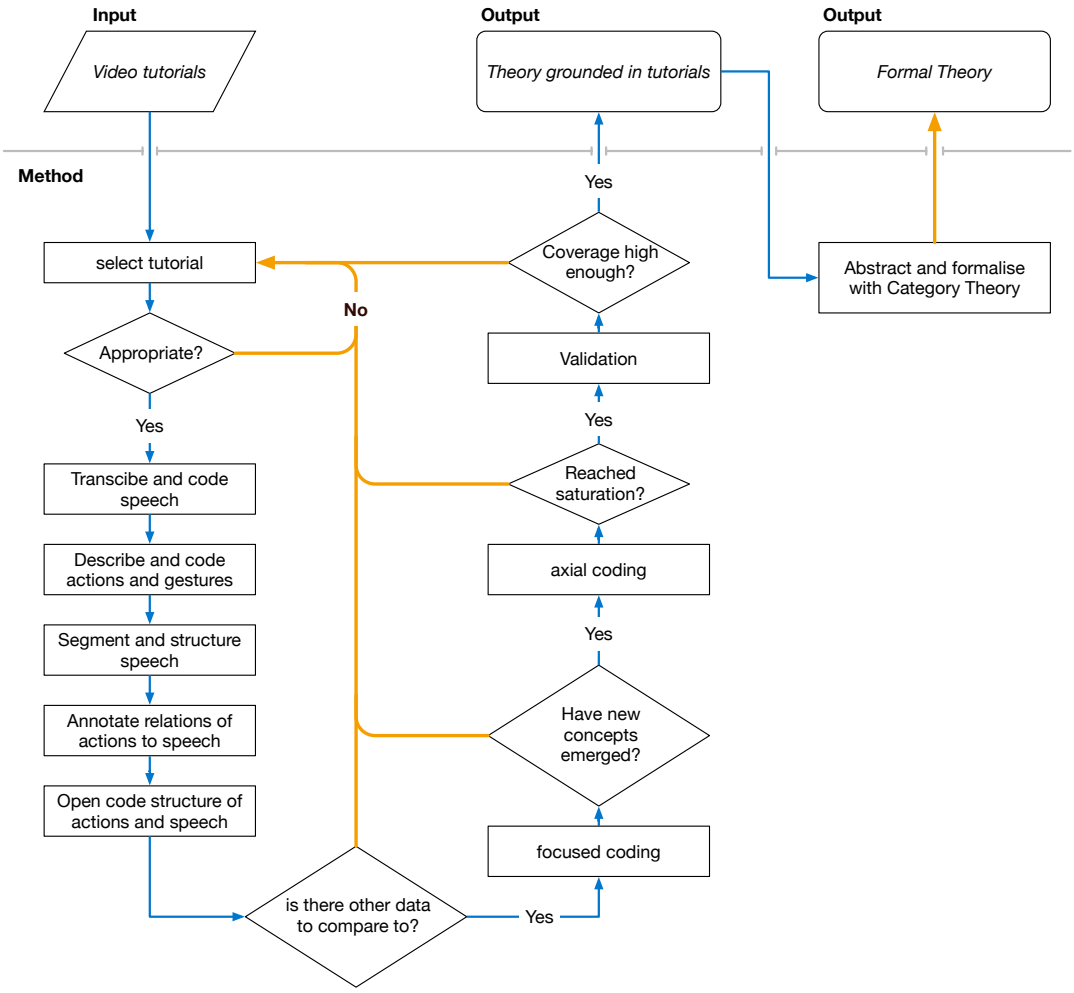


Figure 3.4: This informal diagram shows the steps taken in working from the video tutorials to a formal theory (preprocessing, Grounded Theory application, and finally formalisation with Category Theory). The final step of validating the theory through computational modelling is not shown here.

3.4.1 Preprocessing video tutorial data

The first part of our method is built around the Grounded Theory coding process. To start this process one typically takes transcribed interviews or other texts as the materials to be coded. However, video tutorials are not naturally in a form that lends itself well to coding. As such, we need to devise a way of representing the data in video tutorials such that it does.

The tutor's speech can be transcribed and their actions can be described. These modalities, in and of themselves, do not present any problems. However, there are aspects of the data that do not readily reveal themselves, but that we are nevertheless interested in. More specifically, the development of practical creative process displayed in a tutorial, the way it is structured over time in terms of how the tutor thinks of them, is of particular interest. This structure is not readily visible in transcribed speech or activity. A tutor's speech is mostly concerned with the co-occurring activity and does not explicitly signal how they conceive of or perceive their activity over the long run. The activity observable to us is of course just a sequence of actions and does not carry any information other than their temporal order.

How then do we tease out information about how tutors think of the structure of their own activity? How do we make this structure explicit so that it can be taken into account as we start the GT coding process? We look to RST to fill this gap. RST posits that the organisation of a text is implicit in that the constituent parts of a text do not explicitly refer to the rhetorical relations that hold between them. The structure of a practical creative process is certainly also implicit, for in the temporal unfolding of the process none of its constituent actions can explain explicitly how they relate to other actions. However, we expect that the tutors' speech explains this implicit structure, and we expect that we can uncover the implicit structure of the speech with RST. We expect the speech to follow the practical creative process, and therefore that the RST analysis of the speech uncovers the structure of the tutor's activity.

For the interpretation of a tutor's actions in a video tutorial the analyst relies partly on being able to distinguish between actions that are part of the practical creative process and advance the artefact and actions that are purely communicative. Gesture theory teaches us that, in face-to-face communication, people perceive gestures instantly and without conscious thought. The same is true for gestures in video tutorials. Gesture theory is then something from which we can draw confidence as analysts that actions and gestures can be studied in a GT manner.

A second question regarding action and gesture is how they relate to the modality of speech. We have developed a coding scheme, which will be introduced in Section 3.4.1.4, for the relation of action to speech that can be applied as another preprocessing step for further GT analysis.

3.4.1.1 Transcription of speech and actions

The transcription of the tutorial speech is achieved by using specialised software. A tutorial video is loaded into the NVivo software[6] and the speech is transcribed for the entire tutorial. Care is taken to keep the transcription as close as possible to the actual speech. However, for our purposes detailed information, such as the exact timings of each word, is not required.

Secondly, the actions and gestures are described as well as is practical. The textual descriptions are recorded also in NVivo and include both the use of features of the software, indications of which part of the artefact the tutor operates on, and gestures such as hovering

[6]NVivo is a software that aids in the transcription and coding process. It is specifically geared towards Grounded Theory. `http://www.qsrinternational.com/nvivo/nvivo-products`

or circling with the mouse pointer. The aim here is not to describe the tutor's activity to the level where it is reproducible, but to provide useful context alongside the transcribed speech.

3.4.1.2 Segmenting and structuring transcribed speech

Once the speech and on-screen activity has been transcribed and described respectively, the next step is to perform a Rhetorical Structure Theory analysis of the transcribed speech. This involves segmenting the text and determining relations between segments as described in Section 3.3.3.

Several specialised RST annotation tools exist[7], but these were deemed unfit as they do not allow for adding descriptions of actions with the speech, nor for adding additional codes as described in the next section. However, we have found that the flexible mind-mapping software Freeplane[8] can be used to great effect. With Freeplane it is easy to construct trees, and it allows for attaching arbitrary text nodes to the segments of the transcribed speech. This feature can be used to attach textual descriptions of the co-occurring activity. An example of what the analysis looks like after RST analysis has been completed is shown in Figure 3.5.

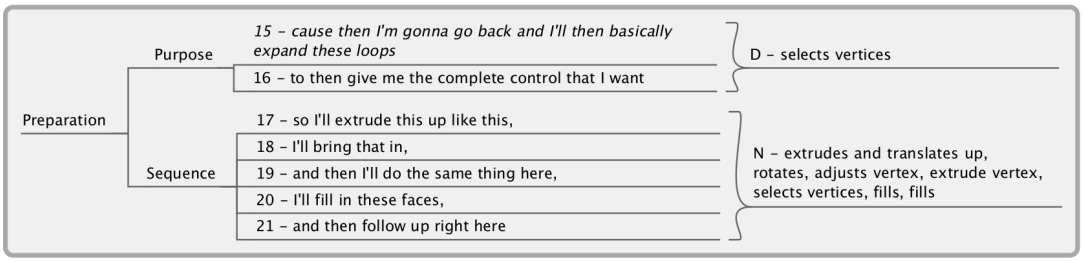


Figure 3.5: This image shows a short excerpt of the analysis of a video tutorial produced with Freeplane. At this stage of analysis the RST analysis has been completed, as well as the action to speech relation annotations. The transcribed speech is shown in the numbered nodes. To the right are the descriptions of the co-occurring activity with the letter code for the action-speech relationship type (e.g. D and N). To the left are the rhetorical relations between segments.

3.4.1.3 Coding the relations between action and speech

At this point of preprocessing we can see the rhetorical structure of the tutorial and the co-occurring activity as depicted in Figure 3.5. There is one more preprocessing step that needs to be taken before we can move on to Grounded Theory coding. This step aims to make explicit what the relations are between the speech and the on-screen activity. For example, a tutor's speech at times describes literally what is happening on screen. At other times the on-screen actions are accompanied by unrelated speech as the tutor explains briefly why a certain technique is considered best-practice. If these different types of relations of action to speech are not made explicit in the data it is likely that the data is interpreted incorrectly in the GT coding process where it is impractical to frequently refer back to the video. Moreover, if we rely on the rhetorical structure to be an indication of the practical creative process, then it is important that the analyst can clearly see how the rhetorical structure relates to that process.

For this reason we have developed a simple action-speech relation taxonomy (AST). The taxonomy can be used to annotate segments of activity with one or two letter codes that make explicit the relation between a tutor's actions and the rhetorical structure of their speech.

[7] https://www.sfu.ca/rst/06tools/index.html
[8] https://www.freeplane.org/

3.4.1.4 The Action-Speech relations taxonomy

We start the description of the taxonomy with the assertion that if it is so that the tutor is speaking, and there is on-screen activity, then there is a relation between the speech and the activity. The first question we ask is whether the activity consists of *advancing actions* or if it is a *gesture*. An advancing action is an action intended to make progress with regard to the artefact, whereas a gesture only has a communicative intent. Gestures are typically easily recognised as they a) do not change the state of the software or the content, or b) they do change the aforementioned state, but their effects are immediately undone. If the action is not a gesture then it must be an advancing action.

When the action is advancing then the speech can either be classed as narration or as being detached. Speech is narration when the tutor literally describes what they are doing on-screen. Narration is tightly synchronised with action, meaning that narration of one action will not overlap with the next. Also, action can be narrated at different levels of detail. For example, a tutor might say "I'll then rotate the hips a little bit" (while rotating the hips of a character), but also "You press shift A, and you go colour, RGB curves" (while walking through menu options). A sequence of actions is performed more slowly if the narration is more detailed and so the actions and narration stay synchronised.

Although a large part of all advancing actions are narrated, a tutor's speech is also frequently detached from their on-screen activity. For example, it might instead be concerned with elaborating on the use cases for a technique, while continuing to make minor tweaks to the artefact. In such cases it is important to realise that the speech is in fact not following the actual practical process.

On the gesture side of our taxonomy we have three subcategories: pointing, showing, and demoing. Pointing gestures consist of mouse pointer movements in a section of the screen in such a way that the gesture is recognised as drawing attention to and indicating part of the software application's interface or the content. Although there are variations to be seen, a typical pointing gesture would look like a repeated circling and a verbal deictic expression such as the word "here".

Unlike pointing, showing gestures change the state of the software. That is, to aid in a tutor's explanation they might want to show part of the content that is yet occluded. The tutor will then *change the view* by, for example, panning around the 3D model or hiding parts that are in the way. The content is not itself changed and after the explanation has finished the original view is typically recovered.

The main characteristic of demoing gestures, as opposed to showing gestures, is that they temporarily *change the content*. The goal can be to demonstrate a software feature, a technique, or that a certain hypothetical action indeed does not lead to the desired result. In essence, in demoing the tutor interacts with the software as if they temporarily have a hypothetical goal (much like is the case with the tutorial as a whole, but at a smaller scale). After demoing the consequences for the state of the software and content are immediately undone, either by deleting the result or by using the software application's undo functionality. The undoing part of a demoing gesture is never mentioned. The performance of a demonstration is often less precise than the performance of an advancing action. That is, the actions might be performed very quickly, or only partly. In sum, we have the following codes for annotation.

N for synchronised narration of advancing action.

D for speech that is detached of advancing action.

P for pointing gestures with a referent in the speech.

S for showing gestures with a referent in the speech.

De for demoing gestures, typically narrated.

R for reflection where speech and actions are both concerned with investigating and judging the artefact.

Segments where there is only speech, or only action, can be marked as having no relation with **NR**.

First and foremost the annotation helps to separate the practical process from presentation. We do not mean that a tutor's activity is either wholly advancing or wholly presentation, but that we need to look at any activity through these two lenses. The annotations will highlight what relations hold between the two, a key aspect of the questions we have asked.

Also, it is tempting for analysts to spend most of the time looking at textual representations of the video tutorials. However, if an analyst only considers transcribed speech then they will be extremely biased towards it. Not all action is narrated or explained and much speech is accompanied by gestures. Therefore, it is important for the analyst to immerse themselves in the speech to action relations and have an annotated version of the transcribed speech that represents what is happening in the other, equally important, modality of on-screen activity.

3.4.2 Application of Grounded Theory

With the preprocessing done the Grounded Theory process can start with open-coding. This is time-consuming as the granularity at which the codes are applied should be quite fine. Another factor is that it is very difficult to interpret and code the first tutorials, simply because the content of the speech does not always suggest literal codes. This is different from the typical case where Grounded Theory is applied. In these cases the text that is coded is directly descriptive of the phenomenon under study and so the codes can be taken from the text either verbatim or by paraphrasing. However, although speech in tutorials is mostly descriptive of actions taken, it is not descriptive of the process as a whole or its structure. It is the latter that interests us and it is therefore that the RST proprocessing step is taken. This makes it so that the larger rhetorical structure can be taken into account in the coding process.

The initial coding provides insights into the types of tutorials that exist. With that insight it becomes easier to avoid wasting time on transcribing and coding non-relevant tutorials or particular segments of tutorials. For example, some tutorials are much more focused on explaining features of the software, whilst others are focused on teaching techniques. The tutorials that are most useful are those that display the process of the creation of an artefact from start to finish. It is easy to recognise whether a particular video tutorial is suitable or not.

The first round of coding can be performed on the transcribed speech within a software like NVivo, while keeping an eye on the annotated rhetorical structures and action to speech relations. The second round of coding flips this around and focuses instead on the rhetorical structures and action to speech relations. This can be done either directly in Freeplane, but also with pen and printouts. An example of a printout after open coding can be seen in Figure 3.6.

As more tutorials are transcribed, annotated, and coded the codes naturally become more focused. That is, a smaller number of codes are used as the similarities between phenomena become more obvious. A looming danger is that subtleties and new phenomena are ignored. It

is likely that after analysis of several tutorials the collection of codes is almost unwieldily. By systematically going through the list of codes and grouping those that are similar it is possible to reduce the large number of codes to a few broader ones.

When grouping codes it is important to realise that a segment of data will have been marked with more than one code if appropriate. The grouping of codes should not be seen as categorising all the data under those codes, but only as categorising a certain aspect of that data. Thus, the grouping of codes is just the grouping of codes, not the categorisation of the underlying data.

The typical GT process of axial coding, memo writing, and theoretical sampling now applies. The theoretical concepts and their dimensions firm up and the theory can be written up. At this point the first output is produced: a substantive theory of practical creative processes in Blender, and presentation thereof, grounded in video tutorial data.

3.4.3 Formalisation with Category Theory

The theory that results from the application of Grounded Theory, as described in the previous section, gives an account of the practical creative process in Blender as well as how tutors go about presenting them in video tutorials. Chapter 4 presents this theory and the reader will see that it is not a formal theory. Grounded Theory does not result in theories that can be readily expressed in a formal language, and it certainly does not purport to do so. However, it is our aim to express our newly produced theory more formally, and we do so by the method explained here.

In addition, as we formalise the theory we will focus purely on the parts that are to do with the practical creative process as opposed to those concerning the presentation of the process by the tutor. We will also aim to make the theory more general by abstracting away from elements of the theory specific to Blender. The formal theory should apply to other domains in which software is used in the artistic or design process.

To achieve our aims we look to Category Theory (CT) for help. It will be both our target representational system, in which we will express our theory, and the tool we will use to help us get there.

As a system of representation CT is suitable because it is flexible enough to represent just about anything, while rigorous enough to allow us to put strong conditions in place where they are needed.

CT has certain properties that aid in the *process* of formalisation, as opposed to just being a good system of representation. Firstly, it provides a formal way of drawing diagrams. This greatly helps in sketching successive iterations of the developing theory, while leaving room to represent parts of the theory that have not yet been fully defined. The diagrams are easy to read and can therefore serve as a manner of communication with collaborators as a theory is developing.

One more advantage of CT is that, perhaps because it is closely connected to programming language design, a theory expressed in CT can be used as a high-level software design. This allows it to be used as a map for the implementation of a computational model that can serve as validation for the theory.

3.4.4 Validation through computational modelling

The use of CT, as described in the previous section, does not provide as rigorous of a process of developing a theory as Grounded Theory does. For example, it does not provide a roadmap of

steps to take or any guidelines to follow. The resulting formal theory represents a creative effort on the part of the researcher and, although it is formal and rigorous, must still be validated in some way.

A method often used to validate a theory is to implement an instantiation of the theory in software. If this results in a valid program that displays behaviour that the theory predicts then the implementation provides some evidence towards the intrinsic consistency of the theory. If, in addition, the theory describes some external phenomenon that the theory attempts to describe or explain then the implementation provides some evidence towards the view that the theory remains, until falsification, a valid one.

In this vein we will take the formal theory and attempt to implement one instantiation of it. The architecture of the software implementation will have to follow the structure of the theory closely so that the implementation is indeed an instantiation of the theory. The implementation will demonstrate the theory in action, and will provide supporting evidence for the theory. Furthermore, software does not allow for anything to be left undefined and an implementation will thus force choices for details left undefined in the theory. This will give insight into areas where the theory is lacking.

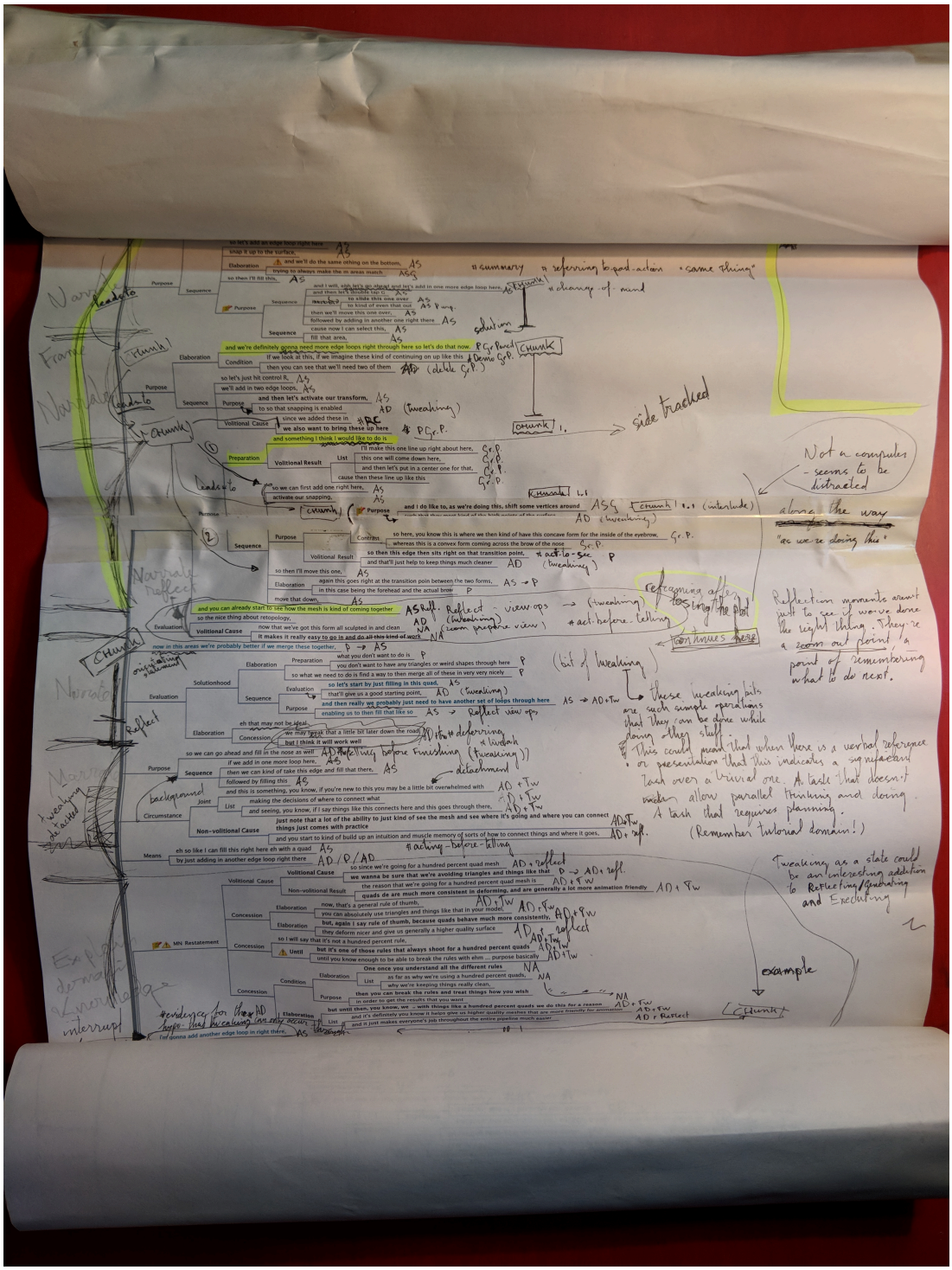


Figure 3.6: This is a photograph of an open coded segment of a video tutorial. The printout consists of several pages of transcribed speech, described actions, rhetorical relations, and action to speech relation codes. Re-occurring patterns are coded with pen and markers.

Chapter 4

A grounded theory of creative practice in Blender

In this chapter we present the results applying the method presented in the previous chapter. First we describe the data we have analysed and then introduce the major themes of the analysis. We then focus on results pertaining to the temporal structure of the practical creative process displayed in the tutorials. A short section focuses solely on matters of presentation and specifically on how tutors gesture. We end the chapter by tying together all themes in a descriptive model that characterises the practical creative process as one of iteratively working from vague idea to concrete artefact, facilitated by the software tool.

4.1 Data

In the previous chapter we proposed analysing the creative practice displayed in Blender tutorial videos. The tutorial videos we have analysed have an average length of 17 minutes. Some tutorials stand on their own, while others are part of a series comprising hours of recorded and commented work. Individual videos are generally not heavily edited; there are only cuts where there would otherwise be minutes of waiting for rendering to finish. Therefore, the videos show a fluent process of work. We have analysed eight videos by six different tutors, totalling 132 minutes of activity. Figure 4.1 shows the tutorials that were included in the analysis.

Tutorial Title	Author	Year	Duration	Blender
Intro to Character Modelling, Skin Modifier	Jonathan Williamson	2013	26m14s	v2.67.1
Intro to Character Modelling, Retopo Face	Jonathan Williamson	2013	36m13s	v2.67.1
Animation Toolkit, Workflow	Nathan Vegdahl	2012	11m15s	v2.64
Animation Fundamentals, Making Poses	Beorn Leonard	2012	6m49s	v2.60
Animation Fundamentals, Blocking	Beorn Leonard	2012	11m59s	v2.60
3D Printing, Texture Painting	Dolf Veenvliet	2013	10m4s	v2.66.6
Render Hair With Cycles	Andrew Price	2013	28m44s	v2.65.10
Bisect, Creating Stones	Gleb Alexandrov	2013	2m9s	v2.69

Figure 4.1: Tutorials included in the analysis.

4.1.1 What generally happens in tutorials

The main goal a tutor has when making a tutorial video is to explain how to achieve a given creative goal. The tutorial as a whole is an explanation of, for example, how to achieve a

certain stylistic effect when rendering a character's hair, or how to achieve effective poses for a character throwing a ball.

In the tutorials we have studied, the tutors explain the topic of the tutorial by means of demonstrating a process. We see their actions on the screen as they progress through that process. The tutor's speech accompanies those actions and typically gives more relevant information about the process. For example, a tutor might give background information on a feature of the software application, or the reason for deciding on a certain course of action. Characterising all speech as explaining helps to understand the goal of the speech, but we should rather look at the subject of the explanations.

We will form the main concepts of our model of creative practice from these subjects. Looking at the data we see that the speech concerns a wide range of concepts. For example, tutors will state how certain features of the software work, what to keep in mind during an activity, what is good or bad about a certain way of doing things, or what they are trying to achieve. We will present the themes that we have distilled in detail, but we first consider how the speech relates to the actions and the cognitive process that underlies them.

From the wide range of concepts being discussed by the tutors, we can be confident that they are sufficiently aware of the practical creative process and the concepts that play a role in it. In fact, tutors are able to reify those concepts and explain them to their audience. It is in fact because the tutor explicitly states their actions and explains what, how, and why they are performed that we can construct an understanding of what the important concepts are. However, it should not be assumed a tutor knows, or can express completely and accurately, whatever is in their mind. Even so, there are many examples whereby a tutor states a goal, a plan that will lead them to achieving that goal, and then can be seen to execute that plan, and be successful. That is, the reasoning that is described in speech is coherent and clearly corresponds to, and co-occurs with, the observed actions. Since the speech is not a post-hoc rationalisation, the explanations a tutor gives are representative of the processes in the mind.

We must ask if practice observed in tutorials is substantially different from that of real creative practice, and if so how. Practitioners might adopt creative goals for a variety of reasons, be they intrinsically or extrinsically motivated. Tutors, on the other hand, will adopt a creative goal for the purpose of producing a tutorial. As such, we might expect that tutors judge their work differently (e.g. less harshly), for ultimately they are producing a tutorial instead of an artefact. However, we do not expect that this causes tutors to follow significantly different practices than they would otherwise.

Secondly, tutors might also adopt different goals than they might do otherwise. For example, they might choose less experimental goals, resulting in less experimental practice. This could very well be a limitation of the chosen method. Even so, we expect that the goals tutors adopt can also be adopted in real practice.

Lastly, tutors interleave their practice in tutorials with communicative acts aimed at the viewer. Some examples of this can be seen in Section 4.2.2.1, Section 4.2.2.2, and Section 4.2.2.3. Clearly, tutors engage in additional communicative activity for the purpose of the tutorial. The pertinent question is whether this changes tutorial practice to such a degree that it is not representative of real practice any longer. We believe this not to be the case, for it would be much simpler for professional tutors to structure their practice like they would normally. Even from a didactic viewpoint, it would be counterproductive for tutors to teach students a way of working that is not representative of how tutors would work themselves.

In sum, we can expect that practice in tutorials and real practice is similar enough for tutorials to be informative.

4.2 Major themes: practice and presentation

4.2.1 Talk of state: construct and content

As a tutor progresses through the process of creating an artefact, they often talk of the past, current, and future states of that artefact. That is, in working from a blank document to the finished article, the artefact will be in a different state every time the tutor takes an action. When the tutor refers to the state of the artefact they do so in two distinct, but related, ways.

To explain the first of these requires the introduction of some elementary concepts from the world of 3D modelling and animation. In computer graphics 3D models are built up from simple geometric primitives. To create the illusion of a 3D body, like a cartoon character or a simple pyramid, one has to describe the outer shell of that body. This is done by first defining vertices, coordinate points in a 3D space. Next is an exercise in connecting the dots and defining which points are connected by line segments, or edges, from one point to another. These lines then form a surface, to which a 2D image can be applied. With the aid of linear algebra it is then possible to project this 3D model onto a 2D screen or image, given the position of a camera aimed at the model with some angle. To make a final visual artefact, such as a movie or image, is to create a 3D body with vertices, lines, and surfaces, and subsequently coloring in those surfaces.

The particular configuration of vertices, lines, and so on, is the way the final visual artefact is constructed. This configuration is what we will call the *construct*. Every single action undertaken in a practical creative process is aimed at modifying the construct. Of course the purpose is to create a 2D image portraying some topic with the illusion of being 3D. The way the construct looks, or what it represents, after it has been converted or rendered to its final 2D form, is what we will call *content*. The whole purpose of Blender as a tool is to make it is easy to produce 2D images by working on a 3D representation and thereby making it possible to achieve certain visual styles.

It might not be a surprise that this distinction between construct and content appears in the way tutors talk about an artefact's state. Although the distinction is clearly present, tutors talk about both levels simultaneously and use terms that typically describe content to refer to construct. The next few sections further elaborate the difference between content and construct.

4.2.1.1 Referring to constructs as representations

When tutors speak of constructs, we can discern several dimensions. Firstly, constructs are seen to be hierarchical. A 3D character has legs, legs have knees, and so on and so forth. What is important here is how tutors refer to those parts. In the following segment the tutor is working on the hand of a character.

Transcript 1 *So in this case you know I'll make the pinky a little bit smaller, I'll scale it down just a bit, I will scale down both the index finger and the ring finger such that the middle finger is larger as it should be.*

We see that the tutor refers to the constructs he is transforming by the names of the fingers they represent. The vertices and edges are referred to here as "pinky", "index finger", etc. Especially in the early stages of creation, the simple constructs are still far removed from what they will represent, but they are referred to in that way nonetheless. If tutors did not mention

what the constructs represented the viewers would be left guessing as to what the vertices, edges, and surfaces are supposed to be.

At other times, the relation between constructs and content is stated much more explicitly. In the next example, from the same tutorial session, the tutor makes the link between content and construct explicit by explaining how subdivided edges represent joints of the character's fingers.

Transcript 2 *Now, right now what I wanna do is I wanna scale each of these tips down, to represent the tips of the fingers, and then we're actually going to go and subdivide eh.. each of these edges a couple of times to then represent the actual joints within the fingers.*

It is not only the constructs, but also actions, that are referred to in relation to the content. Section 4.2.3 discusses actions in more detail. What warrants pointing out now is that the effects of actions are sometimes stated in terms of content as well. In the next example the tutor has nearly finished creating a character.

Transcript 3 *Ok, I think that's gonna be pretty close to what we want, now I might bring the arms back, just a bit.*

As the tutor says he wants to "bring the arms back" he lines up the view so that the character is seen from the side. He selects the construct that represents the arm and moves it to the right. It might seem inconspicuous, but the description that the arms are being moved "back" only makes sense considering the character is facing to the left and backwards for the character is to the right. That is, the way the tutor describes the action relies on an understanding of the content.

4.2.1.2 Constructs have behaviour

Another dimension of constructs is their behaviour. Constructs have behaviour if they are procedural, meaning that they have some algorithmic component that is to be interpreted and executed on rendering. Blender is typically not thought of as a procedural software application, but it has procedural elements still. In the following segment the tutor explains how the colour of each individual hair (in the segment "particle" corresponds to a single hair) of a character will be generated from the pixels in an image. As the tutor says "right there" and "right here" he points at specific pixels.

Transcript 4 *So if a particle is born, let's say right there, it's gonna be a light brown colour. And then if the particle is born right here then that particle is going to be a white colour, etc., for the rest of the mesh.*

As soon as a construct is procedural there is a greater divide between the construct and the content it forms. This divide is in fact always present in Blender, which becomes clear if you consider that the content is generally only visible after a time-consuming rendering step. When the content cannot be clearly inferred from the visible constructs then tutors will explain the behaviour and show the content. Section 4.2.2 considers this in more depth.

4.2.1.3 Constructs have potential

Constructs are not only seen in terms of what they are currently, but also in what they might allow later on in the process. Even though two constructs might have the same content form,

they are not necessarily thought of as equal. The following segment will serve to illustrate. The tutor explains that constructs that have "nice even quads", meaning that the surfaces from which the 3D object is made all have four sides, allowing "much more control". Elsewhere, he explains that this control consists of it being easier to add in detail and to animate the model.

Transcript 5 *You'll notice that if I bring it out we've got nice even quads just kind of flowing through here, which once I apply this and begin moving on to the next section where we're actually going to start splitting the pieces apart this geometry will give me much more control and it's going to be much easier to work with.*

4.2.1.4 Judging content and constructs

Now and then a tutor will stop to act and verbally indicate whether they are happy with the state of the artefact. In this judging of content and constructs tutors are remarkably succinct. Shorter periods of activity are often concluded with phrases such as "there we go", whereas satisfactory progress after longer stretches of activity is often marked by phrases like "that's looking good" or "that starts to work".

When a tutor notices a problem they will be more vocal and explain something is not as expected or desired. These problems can concern either the constructs or the content. For example, when a tutor says that the arms of a character are too big then that concerns the content, because it is a problem of how the content looks. However, when there are duplicated vertices where there should be none, or triangles instead of quads, then that primarily concern the constructs. Of course the two are intrinsically related, for there is no content if not for the constructs, but, in the way they are spoken of, they differ.

What exactly is content judged against? That is, if the arms are too big, what are they too big for? We infer that they are judged in relation to some goal the tutor has. Section 4.2.4 will discuss the concept of goals in more depth. Tutors also employ rules of thumb to judge content and constructs. Such a rule of thumb is concerned with either construct or content, but not with both at once. Evidence for the existence of such heuristics comes from examples such as the following.

Transcript 6 *Now, one thing to think about with ah poses is silhouette, and a good way of checking your silhouette in Blender is actually by changing the viewport shading mode into textured mode. Now if you haven't got any lights visible in your scene it will go black. Ehm, actually it'll go black in GLSL shading or white in multi-texture shading, but even so it turns off all the ... it means you can't see any inner detail, you just got a silhouette. So that's actually a really good way of checking your pose, because if you can't understand what's happening in your pose in silhouette, well, you know, generally it's not a very good pose.*

The tutor here explains a technique that makes it easy to see the silhouette of a character model. The general rule of thumb is stated in the last sentence of the segment. This specific example is concerned with judging content, rather than construct.

It is important to realise that tutors often see state within the process, and not only as what is current. For example, they will say on reflection on the content that they are "making good progress" or that the content is "closer to what we want". This demonstrates that content is spoken of as if it has a trajectory. There is then necessarily a comparison with past and future states. If a tutor signals progress, then the content must be better than in some previous state, but might not yet be as desired.

Whereas these comparisons are internal to the process, tutors will at times compare content with external entities. This includes knowledge of human anatomy, behaviour of real world objects, or cultural references.

4.2.2 Talk of seeing

At times, it is not obvious to the viewer of a tutorial what the state of the artefact is and tutors have to elaborate to make it clear. At other times the state might not be clear to the tutor themselves, and they have to inspect from up close to get a good look. In this section we will introduce the ways in which tutors *act to show* to help viewers understand, but also how tutors *act to see* so that they can better understand state themselves.

4.2.2.1 Acting to Show

In Blender, it is almost never the case that the content and constructs are entirely or simultaneously visible. For example, the artefact might be partially occluded, too small to see, or visualised in a way that prevents seeing certain details. It is in these occasions that tutors might *act to show* by zooming in or taking a view at a different angle. When a tutor acts to show, it is usually accompanied by a phrase that explains what they are showing.

In the following segment the tutor explains the construct of the head of a character created by means of the sculpting tools in Blender.

Transcript 7 *So for this case, since we've sculpted the [1] head with dynamic topology, you know if we [2] go into edit-mode we can see that it's [3] very very dense [4] and there's no real, you know, pattern or anything like that to the [5] face and so it's gonna be very very difficult beyond just render it as is.*

In this short segment the tutor selects the head at marker 1, switches to edit-mode at marker 2 and thereby shows the individual vertices from which the head is constructed, zooms in from 3 to 4 such that the vertices become distinguishable, zooms out again, and switches back from edit-mode at marker 5. The sequence from marker 2 to marker 5 serves to demonstrate the constructs of the head, which are not visible otherwise. These kinds of demonstrations are typical when a tutor acts to show constructs more clearly. For a moment a different point of view is taken or the constructs might be visualised differently, but the tutor returns to approximately the original point of view to resume the process.

Acting to show aims to explain what the constructs are, often because they are otherwise not visible. A different type of showing involves not *what*, but *how* to see. To do so a tutor points at a part of a construct and explains that it forms a certain type of construct, for which they provide a domain specific term (e.g. a mesh made up out of quads).

4.2.2.2 Acting to see

Tutors will frequently act to get a better understanding of the state of the artefact themselves. A tutor might need to get a closer look at something for the same reasons a viewer might not be able see the current state: not everything is visible at once. Tutors cannot fully predict the precise outcome of every action and as such have to continuously evaluate whether the artefact is in the desired state. Because this evaluation requires the tutors to actively move around the artefact and zoom in and out it is possible for us to recognize such moves.

Tutors often act to see the details of constructs that might be hidden. If those details are not visible one can act to change the view, such that those details can be observed. This type of evaluation happens when a tutor wants to see if a result of an action was as intended, or to see if there are problems with any of the constructs.

A different type of evaluation happens at the content level, usually after a longer stretch of activity. For example, in one of the sessions a tutor works on the nose of a character for a minute or so. Before moving to the next part of the face he zooms out and views the face from different angles. This kind of evaluation is concerned with the content as a whole.

It is typical that evaluation of constructs happens at the level at which they are created, while evaluation of content happens from an overview level. This is, however, a matter of degree.

When a tutor changes the view, it is not always an act of seeing. More frequently a view change is aimed at lining up the artefact to act on it from a particular angle. This *approaching* of the artefact is done in a precise and quick way, whereas acting to see involves slow panning and trying different angles.

4.2.2.3 Example of showing, evaluating, and approaching

We include a short video[1] that has instances of all three types of viewing actions. It is taken from a tutorial session and we have added annotations along the top and the right.

The annotations mark several instances of approaching, one instance of showing, and finally a single evaluation at the end of the segment. In a very short period the tutor changes frequently between the front and the side views. This he does not as an evaluative act, but as a way to approach the constructs. Although we have marked only one instance you will see that the tutor also frequently switches between wireframe and solid mode: wireframe for selection and solid for moving vertices. In the segment marked as *showing state* the tutor demonstrates very briefly what the state of selection is. These kinds of demonstration often involve this particular repeated motion. Section 4.4 returns to the topic of gesturing in tutorials. Finally, the tutor zooms out, switches to solid mode, and rotates around the torso and arms. This latter kind of action pattern generally indicates the taking of an overview position and evaluating the content, as the speech here also indicates.

The majority of view changes are not mentioned by the tutor. Generally, only when *approaching* (see Section 4.2.2.2) or *showing* (see Section 4.2.2.1) the artefact does a tutor sometimes explicitly state that they will rotate, pan, or zoom to change the view they have of the artefact.

4.2.3 Talk of action and result

Talking of state is important, but if a tutor does not explain how to get from one state to the next, their tutorials would be hard to follow. In this section we will present our findings on how tutors speak of action. First of all, describing *talk of action* is problematic in the sense that in the context of tutorial videos this kind of talk forms a large class.

We distinguish between three classes of talk of action: past, current, and future actions. The way in which tutors talk about each of these classes is significantly different and it is therefore helpful to subdivide talk of action in this way.

[1] Please see the DVD or view the video at `https://archive.org/download/jSbm2viWiJ`.

4.2.3.1 Past Action

We have found several ways in which tutors refer to their past actions.

Reification to explain action, result, or reason. A common case of referring to past action is to be able to explain its results or the reasons for taking it. Before doing so the tutor has to reify the action, that is to make it explicit such that it can be commented on. We will start with an example.

The following fragment of speech is uttered by the tutor after constructing a part of a 3D mesh. While he was constructing the mesh he did not narrate the action, but explained the general importance of keeping meshes simple and clean. That is, while working on the construct the tutor's speech was not directly concerned with his actions. However, after finishing the activity he feels the need to elaborate on the immediately preceding actions.

Transcript 8 *Alright, now I can start [1] eh filling in this area, [2] so you may have noticed what I was doing is [3] I was starting to position each of these loops [4] based on their corresponding neighbours like this, so I tried [5] to get all the vertices to kind of line up.*

The tutor first frames what his next concern is [1], but before continuing he refers back to the previous actions and explains what he was doing [3], and in what specific way [5]. What you may notice is that the actual actions are referred to abstractly. That is, when he says "positioning each of these loops" he is referring to the repeated translations and rotations of several parts of the mesh. He simply uses the phrase "what I was doing" to signal that he is referring to the immediately preceding activity. This manner of unspecific referencing of action is typical, especially if the speech immediately follows the activity.

In the above example the tutor reified the activity to be able to explain the considerations involved in the action. Reification occurs at other times to explain the link between the action and the resulting construct more explicit, or to elaborate on the reasons for taking that action in terms of the content.

Repetition. A second use of referring to past action is to succinctly explain the next action. Consider the following example where the tutor is creating the hand of a character.

Transcript 9 *[1] Because now I have basically a vertex for each joint, I can just bring it out kind of like this [2], and I'll do the same thing on the pinky.*

The tutor here selects the joints of the index, middle, and ring finger and positions them as he utters span 1-2. He then performs a selection action and a positioning action for the "pinky" as well, and refers to this as "the same thing". In all cases where a repetition like this is signalled the past activity is referred to by phrases such as "do the same" or "once more". The part of the content to which "the same thing" is to be done is simply stated. Once more we see a very unspecific expression referring to an immediately preceding activity.

Summarisation. At times a tutor will refer to longer stretches of past activity comprising many actions. The tutor will do so abstractly, by grouping concrete actions together and referring to them in terms of what parts of the construct or content they were concerned with, or by stating the goal they intended to achieve. We will call this kind of talk of past action *summarisation*. It is important to note that this kind of summarisation is not just a matter of leaving out detail, but rather concerns the grouping and labelling of a sequence of actions.

The past activity that is referred to in summarisation is of a much longer duration than in the previous examples. That is, summaries do not refer to the immediately preceding activity, but can refer to much larger segments of activaty and even the majority of activity in an entire tutorial session.

In the following example the tutor has just finished posing a superhero character such that it appears he is about to pitch a baseball.

Transcript 10 *I mean, it's not it's not a brilliant pose I've made here, but I'm really just demonstrating the eh demonstrating the process. [1] I start with the feet, then move on to the torso and hips, and then work on the hierarchy up there up the upper body and the arms and so on. [2] And that's basically my blocking process. [3] I'll set out a pose and then keep working through the pose. [4]*

The tutor first evaluates the outcome of his activity ("not a brilliant pose") and provides a reason for why that is not problematic ("just demonstrating"). He then summarises the activity of posing a character and in fact does so twice. First in span 1-2 and then in span 3-4. This instance of summarisation occurs near the end of a tutorial and includes almost all the activity displayed in it.

In this particular case, the steps in the summarisation (first the feet, then the torso, etc.) do not refer to the actual actions with any specificity. Similarly, to the other examples the tutor does not say what the actions are, but only mentions what they concern: the feet, the torso, and so on and so forth.

4.2.3.2 Current action

Talk of current action, we will label it *narration*, is a very narrow class that we reserve for speech that is synchronised with actual action and that simply names action as it happens.

The following example of narration is a typical one. The tutor's speech quite literally describes his actions: he types Shift-A on the keyboard, picks the *Mesh* option from the menu that pops up, then picks the *Plane* option, hits the Tab key and the W key on the keyboard, picks *Merge* from the menu that pops up, and finally picks *At Center* from the subsequent submenu.

Transcript 11 *So if I just hit Shift-A, add a new mesh, and a plane.. And this plane, I mean, we're just gonna hit Tab to go into edit mode and just hit W and merge and merge at centre.*

The speech here is tightly synchronised with the actions performed. That is, "and a plane" is uttered as the *Plane* menu option is clicked.

However, not all actions are narrated. Firstly, activity that is marked as tweaking, fine-tuning, or fiddling, is often accompanied by speech that elaborates on some other topic than the actions themselves. These actions can be categorised as repeated small changes that can be clearly interpreted from the video. Similarly, view changes (zooming, panning, and rotating) happen frequently, but are almost never mentioned explicitly. Also, acts of demonstration such as pointing at parts of the content are similarly never mentioned explicitly, possibly because they are part of the communication itself.

4.2.3.3 Future action

Having defined talk of past action and the narrow class of narration we are left with talk of future action, or planned action. Very rarely do tutors speak of future actions with the level of

detail we have seen in the previous examples. That is, instead of mentioning hitting keys, menu options, or moving objects around they talk of what they would like the content or construct to be like. We consider the following example.

Transcript 12 *Alright, one more loop that I would also like to put in is I would really like to put in a loop that then follows around the nose like this that's gonna make it really easy to get this almost kind of like little button nose of sorts that the character has.*

Reading this it appears that the tutor's planned action is to "put in a loop". However, there is no mention of any concrete actions, but only of the desired result of there being a loop. This is in fact typical. Instead of saying what concrete actions they will perform, tutors will state that they will do *something* such that they get the desired result.

Talk of future activity is similar to the practice of summarising discussed earlier in the sense that the details of the actions are not mentioned. The two are fundamentally different, however, because when stating future activity they are unknown, whereas when talking about past activity they are left out or forgotten.

4.2.3.4 Framing

There is a particular way tutors will start a segment of narrated activity by first framing it. This happens typically at the start of larger segments of activity. In this framing tutors say specifically what is next and thereby mark a new segment of activity. The framing of the ensuing narration of actions in the following transcript is the phrase "next let's bring this on around". It is accompanied by pointing with the mouse pointer at the corner of the mouth of a character, which is the "this" the tutor refers to.

Transcript 13 *Next let's bring this on around, I'm just gonna take these vertices here, just hit E to extrude, S to scale out, bring 'm down, and then these can just fill like that.*

At times framing is preceded by a reflective statement on the preceding segment of activity. Consider this example.

Transcript 14 *And that gives us the majority of the face, but let's continue this up on here.*

Here a segment of activity just performed is wrapped up by the phrase "and that gives us the majority of the face" and is immediately followed by the framing phrase "but let's continue this up on here". Section 4.3.1 discusses the use of these *connective pairs* as indicators of segments in the overall structure of creative practice.

4.2.3.5 Action Schemas

All the above examples are orderings whereby one action follows another and the speech happily follows along. The ordering of actions is perhaps not explicitly marked in speech, but it is explicit in that it actually happens. There are, however, moments where tutors are explicit. For example, tutors will say some activity will have to be left till later, or that before doing X, one first has to do Y. We have distinguished between several such general *action schemas*, that is ways of talking about activity.

The first is the most common and is that of *dependency*: doing some activity before any other. A second form is similar to the first, but specifies the other activity. That is, instead of saying X before the rest or first X, it says X before Y. It is also possible to simply say an

activity should be *deferred*, that is left till later, without saying that it is to be preceded by another activity. A more complicated schema is that whereby an activity is said to be done *along the way*. We interpret this as a type of activity that is to be interleaved with another. For example, one might tweak or fine-tune as one progresses. Another schema is *working through*, denoting the repeated application of a sequence of actions. An example of this is posing a character frame by frame. Finally, we consider the schema *working from a default*, whereby there is a sequence of actions that is done first and the result copied, such that one does not have to repeat the same exact sequence of actions over and over.

We should note that these schemas are mentioned by tutors, and our aim here is not to describe a true ontology of action schemas, but to uncover ways in which people speak of action. It would be interesting to look further into such general action schemas, but it is outside of the scope of this work.

4.2.4 Talk of goals and reasons

In the previous section we claimed that talk of future action is in actuality talk of goals. We also introduced the concept of framing. In this section we will look at both of these in more depth. We will also look more closely at how the concept of goals relates to that of reasons to act, as they are not entirely identical.

First we define what a goal is. We saw earlier that talk of future action is often stated as *doing something such that we have X*, where X is a description of the desired state of content or construct, or quality thereof. Whenever a tutor says that they want the content to be a certain way, then we interpret that as stating a goal. That is, goals are desires of state.

At the start of a tutorial the tutor will typically say what the tutorial is about and broadly what they will attempt to do in the tutorial. For the purpose of the tutorial some goal is adopted. That is, the tutor would not have the goal if it were not for the desire to make a tutorial. Even though *real* goals would likely be more complex, we assume the *tutorial* goals to be similar enough to not render our analysis irrelevant.

Goals can be stated either at the content or the construct level, whereby construct level goals are seemingly derived from those at the content level. Goals are likely to be stated in abstract terms. In the following example the tutor has just finished a segment of activity that ended with rendering the image to see the current state of the content. He states clearly what his content level goal is for the next segment of activity: having the light shine through the hair from behind the character. Then he immediately progresses to act and adds in shaders that change how the hair is rendered such that the light indeed shines through the hair.

Transcript 15 *Now the only other thing that I'm gonna do is I'm gonna make it so that the light that shines through our hair here actually passes through to the other side. [1] Ehm.. and so the way I'm gonna be doing that is by adding in a* translucent *shader [2] and then adding in an* add *shader, not* mix *shader, it's an* add *shader.*

At the position marked [1] the tutor starts the actions necessary to add in the translucent shader, and at [2] he starts adding in the *add* shader.

The visual style that the tutor is after is stated as the goal, and by doing so he frames the next segment of activity. The goal at the construct level is to have a different type of shader to take care of the way the hair is rendered, and so he progresses to narrate the actions required to get to that construct state, which in turn should yield the content state he desires.

This is in fact a general pattern: segments of activity are framed by stating a content goal, after which there are statements of construct goals, followed in turn by narration of actions. Furthermore, content goals are stated in terms that include human interpretation (e.g., a mesh representing hair), whereas construct goals are explained in terms of the construct itself (e.g., add, mix, and translucent shaders).

4.2.4.1 Chaining goals

We consider another example. The next segment of speech is uttered by a tutor near the start of a tutorial.

Transcript 16 *And in this case what we're gonna do is [1] we need to create a nice animation-friendly version of the face here, [2] so for this video we're just focusing on the face and then in the next one we'll do the ear, and followed by the head, so the general outline that we want. [3] If you're not familiar with facial topology I do encourage you to check out some of my other training, I have some a dedicate collection on understanding topology including a topology overview of the human face that I encourage you to look at because it'll definitely go into far more detail than I'm going to here, [4] but the basic premise that we want, we want rings of faces around the eye, a ring of faces around the mouth, and then a ring of faces coming from the bridge of the nose going around the mouth down to the chin, and then ideally I would also like to see a ring of faces coming from the forehead going around the outside of the eye and then coming in here to the bridge of the nose, kind of commonly referred to as the racoon mask. [5] And so by following this general structure, what this is going to enable us to do is have a very nice animation friendly model.*

In this segment the tutor clearly frames the next segment of activity by stating a goal. At marker 1 he states that the construct level goal is to have a nice animation-friendly face (we will label it goal A), meaning simply that the 3D model is structured in such a way that it can be easily manipulated later on when it is animated. The tutor proceeds to mention, at marker 2, that doing the face is not the only goal there is: the ear and the rest of the head have to be done as well. However, that is not the current concern. The tutor then introduces, at marker 3, that there are other tutorials available; we will ignore that fact here. At marker 4 the goal is elaborated on and what we view as *sub-goals* are stated. There are certain parts of the artefact that need to be constructed: loops around the eyes, mouth, etc. This set of goals, also stated at the construct level, we label B. The specific construct that is described is referred to as a racoon mask. The conclusion of this segment, at marker 5, is that the sub-goals of building a racoon mask will satisfy the goal of having a "very nice animation-friendly model".

There are two things that deserve to be pointed out here. The first is that goals chain together, or rather than A breaks down into B. Given goal A, the tutor reasons that achieving goal B will achieve goal A. This is an act of inductive reasoning. It is likely that the tutor believes from experience that constructing racoon masks results in animation-friendly faces, therefore it should be so in this instance as well. It could however be the case that he is wrong and that the face is different in such a way from all other faces that came before that his strategy will not work.

In working from abstract goals at the content level to concrete actions, there should, at some point, exist a goal at the construct level. Once arrived at the construct level goal, actions can be imagined that yield that construct. We then have a reasoning process whereby abstract goals are made ever less abstract until they are so concrete that a sequence of actions can be

committed to that is believed to achieve the concrete goal, which in turn is believed to achieve the abstract content level goal.

4.2.4.2 Reasons

Most explanations that tutors give for taking actions are that they achieve some goal. Moreover, the reason for a particular goal is that it achieves some higher level goal. This is of course aligns with the common sense definition of reasons. However, not all reasons for action are equal, and they are not all stated by tutors in the same manner.

The most frequent kind of reason to act is to advance the content. That is, the content structure is transformed into a state that is closer to what is desired, or into a state that allows a future activity to be easier to accomplish; the difference here being that the first aims to influence the final perceivable state, whereas the second instead aims to transform state such that it allows further action.

Another type of reason to act is to prevent an unwanted consequence that would be caused by future action if not for some action that can be taken now. Tutors will often speak of "preventing" or "countering" future unwanted side effects.

There is also frequent mention of administrative reasons for acting. For example, by naming objects and organising project structure and layers things can be "kept clean".

Finally, there is a type of reason that does not result in action, but in the refraining from action. There are many occurrences of tutors considering a setting or possible action and deciding not to change or act. In these cases they either mention that the default is what they want, or that changing it would be inconsequential.

4.2.5 Talk of software and features

A large part of speech in tutorials is concerned with explaining how the software and its individual features work. The previous sections dealt with how tutors explain the goals and actions taken and the reasoning behind them. Explaining the software helps to understand why certain actions have certain results. In this section we will show that tutors have clear ideas about the software, that they reason about it, and explain it with the use of hypothetical situations.

To define what features are, we first look at some properties that tutors endow them with. Tutors often know clearly the *provenance* of a feature. This includes knowledge of when it was introduced, the software developers that worked on it, and whether it is still in development or not.

Features are *judged* mostly in terms of a) quality, that is whether it works correctly and produces good results or artefacts, b) ease of use, that is whether it is easy to understand, confusing, or complicated, and c) utility, that is whether it is handy, powerful, or fast.

Furthermore, features have *alternatives* as there are frequently other ways to achieve something. From this it follows that they can be compared to each other, and tutors indeed frequently do so. And features of course have *interfaces*; their buttons, and settings are located somewhere in the user interface.

Although tutors talk of the above properties of features, first and foremost they explain that features have *uses*, that is they allow to create a certain class of things. Tutors either refer to a feature's use in a general context or in a specific context. For example, when a tutor says that the strand rendering feature can be used to make hair, grass or carpets the tutor does not

refer to a specific instance in which that feature is used. However, in a specific context the use of a feature is the means by which a goal will be achieved. Consider the following example.

Transcript 17 *So if we want to animate the face or to rig it or pose it or do anything with the face basically we need to retopologise it, we need to create a nice clean structure to the mesh that we can then work with. So the way that we're gonna do this is using the snapping tools in Blender. We're actually going to generate a new mesh and basically just trace out along the sections that we want.*

Here we see a mention of the means by which the goal of having a nice and clean mesh structure is to be achieved: by using the snapping tools. The specific use of a feature allowing a goal to be achieved, regardless of the goal's specificity, is a dimension of talk of action we have not yet mentioned. There is of course no other way to act in software if not by means of using one feature or another.

4.2.5.1 Explaining the workings of features

Given that each action in a tutorial is an interaction with a feature, and that each goal is to be achieved by means of a feature, we should then ask how and when tutors explain how these features work?

The first of two answers to this question is that tutors simply show how it is used throughout the tutorial. Short explanations of the behaviour of features are intertwined with narration of actions. The following segment shows two such examples.

Transcript 18 *Next let's do go ahead and add in a mirror modifier so that we can be working symmetrically, and we will enable clipping and then what we want to do is just turn on surface snapping so by default when we hold down control anytime we move something we're snapping to grid increments.*

First the tutor states that he adds a "mirror modifier" and why. The reason given here, "working symmetrically", is an explanation of the workings of the modifier. Although it is not explicit the viewer can easily induce that mirror modifiers allow for working symmetrically.

The second feature explanation in the above example is much more explicit and it illustrates the second answer to our question posed. The tutor explains that if the surface snapping setting is on, and if you hold down control, then whatever part of the content you move will snap to a grid. What we want to note here is the use of hypotheticals. This type of explanation first clearly states the conditions for the feature's settings and the content, an action, and then the expected result. That is, features are explained as having a certain behaviour.

We infer that tutors have predictive models that describe the behaviour of software features, and that these models play an important part in their reasoning. That reasoning is seen to go two ways: from goal to action, and from result of action to feature behaviour.

Going forwards, from goal to action, we can see how given a goal, and by means of a software feature, actions could be envisioned that would result in a state that achieves that goal. Thus, the tutor's model must be predictive.

Going backwards, from result of action to feature behaviour, we can see how, given an unexpected outcome, a tutor would have to reason about why a feature behaved like it did. Firstly, there could be modal errors, whereby the user was not aware of some state either in content or feature settings. Secondly, there could be model errors, whereby the user has an incorrect or incomplete understanding of how a feature works and therefore makes a an incorrect

prediction. Finally, there could be performance errors, whereby the wrong button was clicked by accident, or, for example, the tutor failed to perform a delicate physical movement, such as a precise painting stroke.

4.2.6 Trying and expectations

From the previous sections it might appear that, given a content level goal, a tutor can infer, with confidence, the construct level goals, and actions, that will yield that content level goal. That is, it might seem that tutors reason about the steps of the creative process with exactitude and certitude. However, this description of the tutors' reasoning would not be an adequate account. Tutors can sometimes be seen to be unsure that the course of action that they have decided on will actually achieve what they want. They will speak of "trying", or changing their goals after a certain approach has failed. If we consider that the creative practice displayed in video tutorials is likely simpler than that of creative practice outside of the tutorial context, then we can infer that acting with uncertainty is even more frequent in the non-tutorial setting.

First we take a closer look at an instance of *trying*. In the following segment the tutor has been "fiddling" with the torso and shoulders of a character. His goal is to make the torso less shallow. However, the constructs behave in such a way that he cannot achieve the desired result.

Transcript 19 *[1] So sometimes it can, it can take a little bit of experimenting to see the kind of what works and what doesn't. (pause) In this case [2] I'm gonna try someth.. I'm gonna just delete this vertex. [3] Sometimes I've noticed that it's it is hard time to [4] get e eh or you'll have a hard time getting your branches to work exactly, so you might just redo that one portion and as soon as I reconnect these edges [5] to the roots then it generates. [6] Now in this case that hasn't really helped me at all, so I can scale down the radius to be something like this, [7] and that will probably be alright, because again that's something that I can adjust as I go, no problem. Ok, I think we'll probably leave it just about like that, that starts to look pretty good I think.*

At marker 1 the tutor scales the control point up and down, but it does not respond. He deletes the point from marker 2. At marker 3 he recreates the deleted point, and at 4 and 5 he recreates the connections between points that were lost on deletion of the original point. At marker 6 he continuous fiddling with the control point, but it behaves similarly as before. Finally, at 7 the tutor zooms out and pans around the model to evaluate.

We see here that the tutor has a problem with the construct; it is not behaving as he wants. It occurs to him that recreating part of the construct might help. It could not be said he is certain it will help, because that is what his speech indicates. Indeed it turns out that his attempt to circumvent the problem does not help. Subsequently, he drops the goal of making the torso less shallow, does what he can, and defers obtaining the result he wants.

What is interesting here is not only the uncertainty with which is acted, but also that the content goal is dropped, or at least deferred, in light of the fact that the required construct is not possible.

4.3 Temporal structure

Section 4.2 introduces a few themes related to the temporal structure of the practical creative process, such as talk of past and future action, and more specifically that of framing.

However, these themes do not readily reveal much about the temporal unfolding of a creative process. This section focuses on the results found by looking closely at the data from the RST preprocessing step.

In the analysis of the rhetorical relations, we find points in the speech that do not easily fit within the RST framework. That is, working from the clauses up it is relatively easy to find small branches that span around 10 to 15 segments. However, once this level is reached it becomes difficult to relate these branches and in such a way form larger trees. That is, the contents of the branches, as interpreted by us, are either difficult to relate, or they cannot simply be related to a single other branch. It is at these points of analytical tension that questions about the structure of the tutorials arise. In other words, the RST analysis, as a pre-processing step for GT, provides us with a handle on relevant data.

We will discuss some of the analytical tensions with the help of an example tutorial analysis. The tutorial analysed is part of a series in which the tutor is creating a cartoon character. In this part of the series, the tutor creates the lips and the jaw of the character's head. The figures in the following sections show small parts of the analysis. Figure 4.8 depicts a larger, continuous, segment of the analysis, which includes all the smaller parts. After we have highlighted the points of tension, we will discuss what conclusions we can draw regarding the structure of the tutorial process.

4.3.1 Points of tension

4.3.1.1 Pivot points

We start our discussion with the 14 phrases shown in Figure 4.2 below. In segment 1 the tutor introduces the activity that is to follow: creating the lips of a cartoon character. Then in span 2-5 he attempts to construct a large part of the lips at once, but reflects, in segment 5, that his effort does not lead to the desired result. He then goes on to formulate an alternative way of going about creating the lips, which is executed in span 6-14.

As depicted we have marked segment 5 as an *evaluation*[2] of the narrated sequence[3] of actions in span 2-4. In that evaluation the tutor remarks that there is a problem with the method, after which he offers the solution to "do these manually".

So, we have segment 5 looking backwards and segment 6 looking forward. Segment 5 is interpreted to be an evaluation of the effort that precedes it, and segment 6 is taken to be a *preparation*[4] of the effort that comes after. However, segment 5 and 6 can also be related by the RST *solutionhood* relation[5]. It would be hard to argue that this relation does not exist between the two utterances, for 6 clearly presents a problem and 7 clearly presents a solution in response to it. However, the analysis problem with which we are then faced is that the solutionhood pair is equally related to the activity in span 2-4 and to that in 7-14, for 5 still reflects on the preceding actions and 6 still frames the next sequence of actions. As such, we can only decide arbitrarily how to form the relation tree.

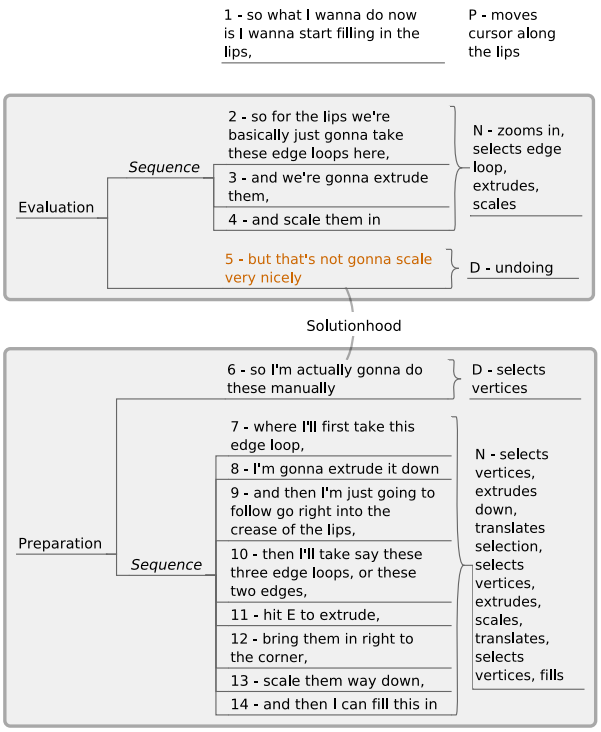


Figure 4.2: Example RST analysis - span 1-14

An alternative analysis would be to have a solutionhood relation over 5-6, have this pair be an evaluation of the sequence 2-4 before it, and then have the whole of 2-6 be a preparation for the activity that follows. That is, we could read the simple attempt at creating the lips as a demonstration of why it is necessary to do the lips manually. This is certainly a plausible hypothesis, but we have no way of determining whether this was in fact the tutor's intention. We are inclined to believe that the attempt is genuine, and that the discovery, that it does not

[2]The intention of the evaluation relation is defined as: the reader recognises that the satellite assesses the nucleus and recognises the value it assigns

[3]The sequence relation is multinuclear and its intention is defined as: the reader recognises the succession relationships among the nuclei.

[4]The intention of the preparation relation is defined as: the reader is more ready, interested or oriented for reading the nucleus

[5]The intention of the solutionhood relation is defined as: the reader recognises the nucleus as a solution to the problem presented in the satellite.

have the desired results, brings about the next activity.

Thus, we have found a point of tension. Our analysis of pairs like 5-6 is that they are *pivot points*. In a pivot point the first half reflects on the preceding activity and the second half frames the succeeding activity. Pivot points are, as it were, wedged between two chunks of activity. The halves of a pivot point, here segments 5 and 6, are intimately related, not just because of their proximity, but because the framing is formed in light of the reflection.

The concept of a pivot point is not easily expressed in the binary and hierarchical framework of rhetorical relations. However, what is important here is not that we cannot find a solution to the question of which relations exist, but that the RST analysis has brought the question forward. It has in this case allowed us to see that there is a reflecting utterance followed by a framing one. We see also that they are interposed between two separate chunks of activity, or rather, it has made us perceive the actions of the tutor as two separate chunks of activity.

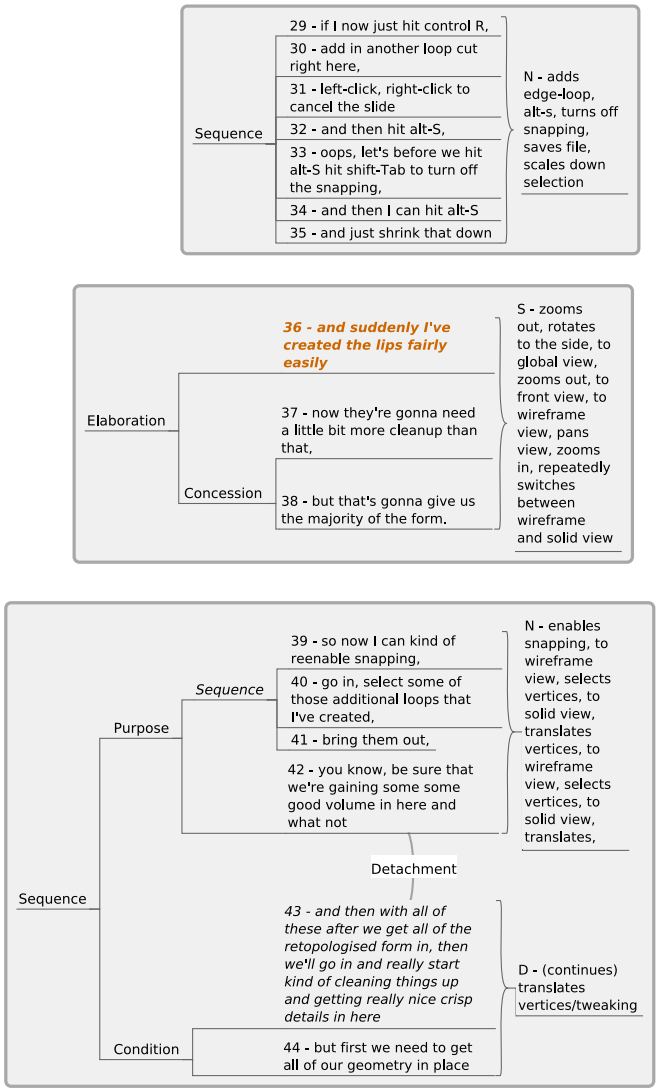


Figure 4.3: Example RST analysis - span 29-44

Another pivot point can be seen in Figure 4.3 on the right. The span 36-38 makes up the pivot point, connecting the activity in span 29-35 and span 39-44. Here the reflection on the preceding activity and the state of the artefact is in itself the framing of the "cleanup" that follows. That is, the evaluation of the state of the artefact and the framing of what needs doing next are expressed as the same thing. That is, the perception of the state of the artefact is the stating of the next concern. From span 36-38 we can see that the framing and reflecting halves of a pivot point are separate functional components, but that they are not necessarily syntactically separate.

4.3.1.2 From narration to detached speech

We now turn our attention to span 22-28, shown in Figure 4.4. Here, in the interaction with the software during segment 22, the tutor first positions the view such that he is able to operate on the lip construct as he wants. He then modifies the model to create "overlap" in span 23-27. He narrates both the view change and the actions that follow it. What we see then is that he, in speech, gives background[6] information about this activity in span 26-27 by contrasting the result he is obtaining now with a past state of the construct where it was not possible yet to create the overlap. While talking about this past state he continues to work on the lips, making a fair few small adjustments. What occurs from segment 25 to 26 is a detachment of the speech, meaning that the tutor goes from narrating concurrent action to talking about another concern. Then in segment 28 the tutor has stopped tweaking and reflects on the current state, summarising the preceding activity and rotating the view around the face such that it can be viewed from all angles.

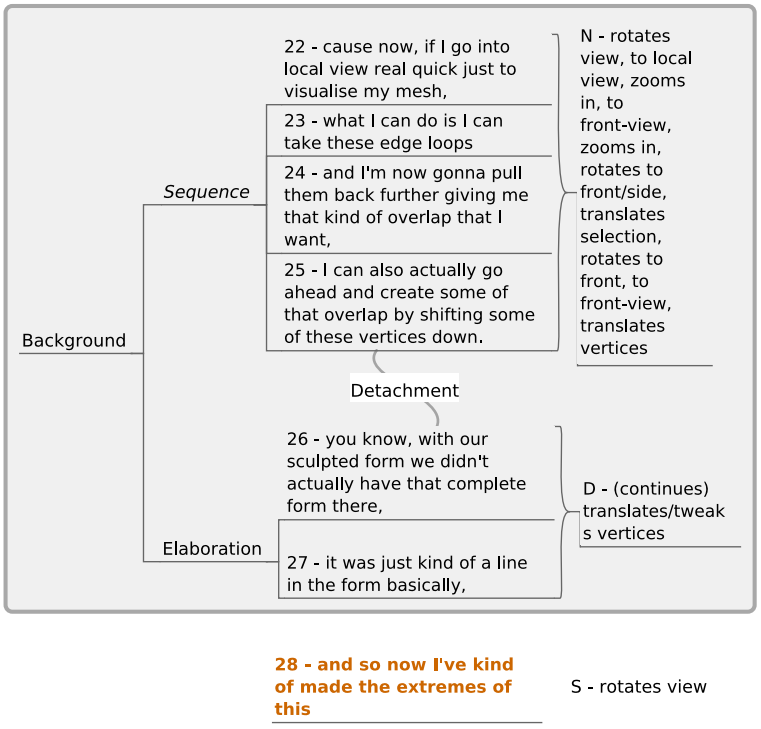


Figure 4.4: Example RST analysis - span 22-28

What is key here is the detachment of the speech from narration to giving background information, while the activity in the interaction with the software continues. There are then two separate coherence relations in the two separate modalities: a) the background relation in the speech, and b) a relation in the activity to indicate it continues. If we had not thought the relation between action and speech important, and if we had only been working with the transcript and looking at the rhetorical relations, then it would have been difficult to see the detachment. Because we have noticed the detachment, we can now group the activity in 22-27 together. This grouping is now based not only on the rhetorical background relation, but also on the continuing activity. Thus, RST analysis of the transcription alone would have provided

[6]The intention of the background relation is defined as: the reader's ability to comprehend the nucleus increases.

an incomplete picture of the aggregate activity of speech and interaction with the software. We see then that the application of the speech-action taxonomy provides a more comprehensive view.

The chunk formed by span 22-27 is followed by a reflecting utterance in segment 28. The action-speech relations, apart from simply helping to represent continuity in activity, now also allow us to question the relation of the reflecting utterance to the entirety of the preceding activity in 22-27, rather than solely to the interjected background information.

From other observations of detached speech we find that it is typically accompanied by activity that consists of making many successive small changes. This type of activity is sometimes introduced by the tutor as "fine-tuning" or "tweaking". We have found only one case where a tutor made larger changes to the construct while attending to other concerns in the speech, only for the tutor to immediately revisit and explain what he was doing. Tweaking seems to occur mostly near the end of a chunk, and so detachment does as well.

We cannot know what the reason for the frequent co-occurrence of tweaking and detached speech is. It is, for example, hard to know whether the tutor wants to tweak a construct, which is difficult to narrate, and therefore talks about something else instead, or whether the tutor wants to talk about something else and therefore can only do something that does not need narrating. Both could be of course the case. However, we can entertain the hypothesis that the reason for not making large changes during detached speech is that it would too severely disrupt the understanding of the process on the part of the viewer. That is, if the actions involve any complex or lesser known feature, or hard to see interactions, then viewers are unlikely to be able to infer what they were from the resulting changes to the constructs. This is not the case with simple changes (simple in both action and effect), such as is the case with tweaking activity, for it uses solely well-known features in clearly visible interactions and therefore the viewers should be able to infer the actions.

More importantly, under this hypothesis the tutor is seen to reason about what the viewers are capable of understanding, given the tutor's own performance. It is likely that a tutor reasons about the audience when preparing a tutorial, but that is not quite the same as the meta-reasoning that seems to happen simultaneously with the performance of the tutorial.

To conclude this section we would like to highlight the detachment that happens after segment 64, as shown Figure 4.5. The "fine-tuning" that is introduced by the tutor in segment 63 is continued in span 65-77. The narration of the tweaking is then skipped entirely, and the tutor proceeds to explain that the tactic to "keep things simple" generally applies in 3D modelling. Interesting here is that the tutor reiterates it four times, while being partly occupied with the fine-tuning. Whereas the example of detachment we gave before had two different types of coherence relations, here we only have a coherence relation in the continuing activity.

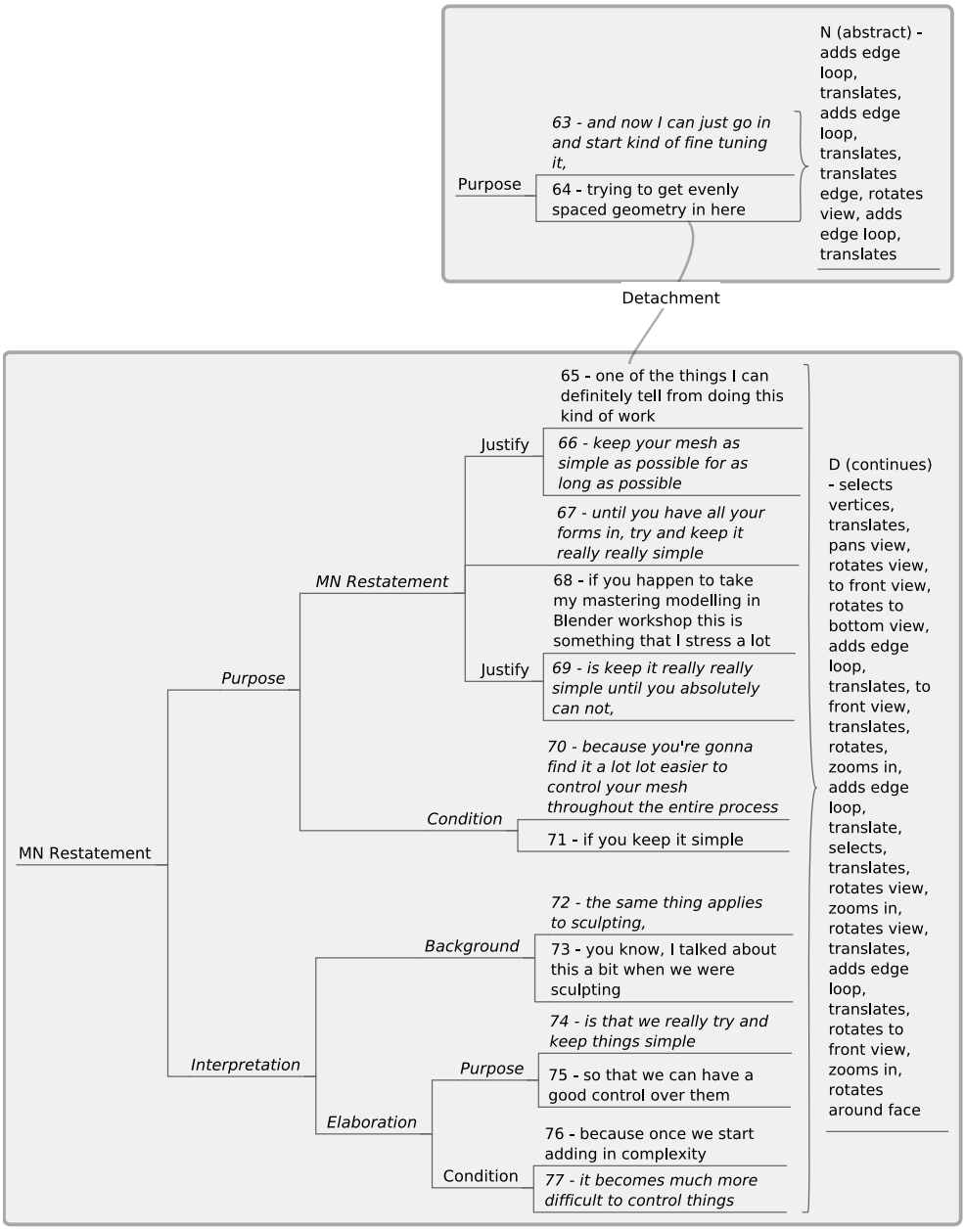


Figure 4.5: Example RST analysis - span 63-77

4.3.1.3 Backward reach

During the discussion of span 22-28, the reflecting utterance in segment 28 was not related to the preceding segment. Segment 28 is shown, again, in Figure 4.6, but with more of its surrounding context. We could have very well interpreted utterance 28 as an evaluation of the activity that directly precedes it, but we could have equally reasonably taken segment 28 to be an evaluation of the two preceding chunks, or three, and so on and so forth, for surely the current state was achieved through a sequence of a great many actions and not just the directly preceding ones. It is perhaps true that the tutor has a definite sequence in mind when reflecting, but as analysts we cannot know what it is.

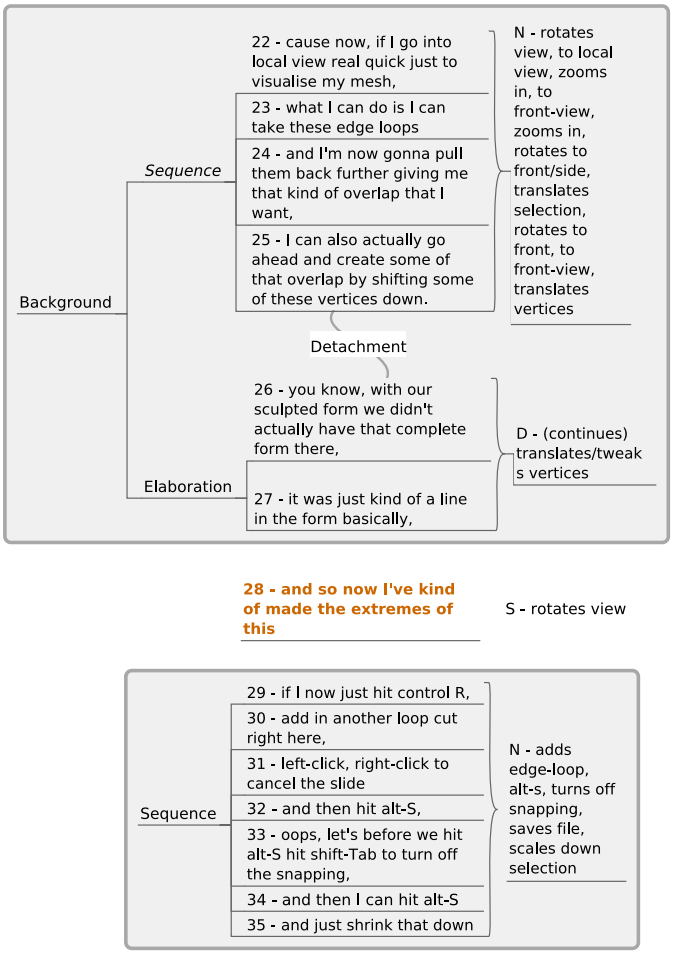


Figure 4.6: Example RST analysis - span 22-27

A similar, but separate, issue is that many utterances of reflection are best understood not as relating to the previous activity, but to the state of the artefact. For example, if we contrast segment 28 and 45 we see two different types of utterance. The first can be taken to be a summary of the activity before it, in that it characterises the outcome of it: "the extremes of this". Segment 45 ("alright so that's looking really good"), shown in Figure 4.7, cannot be interpreted in the same way. Rather, it is better understood as a reflection on the state of the artefact. That state is of course the outcome of the activity that came before it, but therein lies the problem, for it is the outcome of *all* activity that came before it. We therefore cannot, or should not, relate these utterances to any spans before or after it.

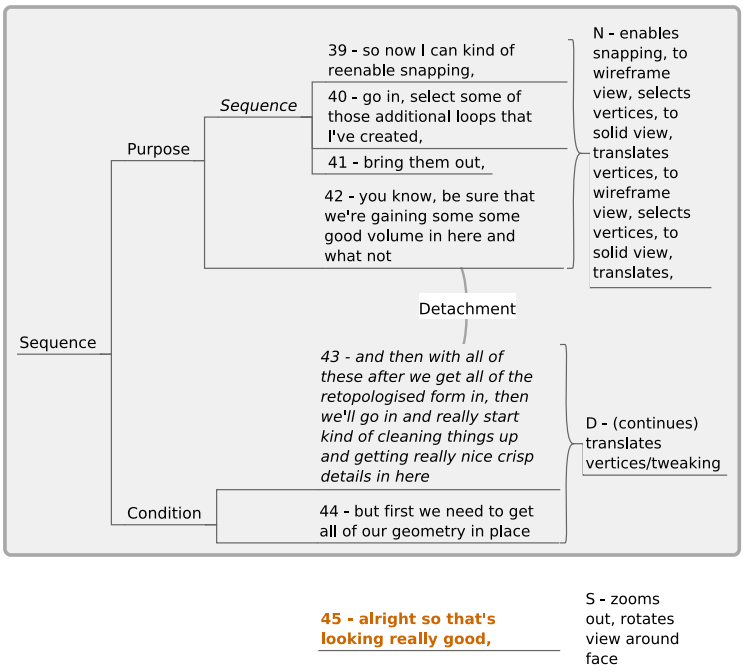


Figure 4.7: Example RST analysis - span 39-45

In fact, it would make more sense to also view segment 28 as a reflection on the artefact being produced, rather than on the content of the preceding speech. If viewed that way we cannot relate 28 to the preceding chunk of activity, because it is simply not concerned with it. Rather, segment 28 and the preceding speech only indirectly share a subject matter: the construct being worked on. That is, although the subject matter is the same, they do not stand in a rhetorical relation to each other.

The same argument can be made with the distinction between content and construct that was introduced in Section 4.2.1. When a tutor reflects, like in segment 28, a pertinent question is whether it concerns the content or the construct. In the case of segment 28 we are inclined to believe that this concerns the construct rather than the overall content, for the view is zoomed in and the individual vertices are displayed. If we contrast this to segment 45, then we can start to see the distinct difference in quality between the two forms of reflection. Whereas reflection on constructs is rather matter of fact, reflection on the whole of the content is more subjective. That is, we can view reflection on a construct along the lines of making sure that the preceding action indeed had the intended mechanical outcome, whereas we can view reflection on content as further removed from the immediate terms of the software domain (e.g. vertices, quads, textures, etc.) to more abstract terms related to expression, emotion, quality, and intention. Then, as we argued before, reflection on content is reflection on the emergent properties of constructs, and as such this kind of reflection cannot be solely concerned with any directly preceding local changes of constructs. Again, this makes it so that it is difficult for the analyst to know what the reflection regards, or how far back it reaches.

Aside from the distinction between reflection on content and construct, we should point out that we have observed reflections specifically concerned with activity, such as in Transcript 19 on page 75. In such cases the speech is clearly related to the preceding activity as an evaluation or summary.

4.3.1.4 Forward reach

We could argue along the same lines that the reach of forward-looking utterances, such as those in segments 1, 6, 15, 46, and 63 (please see the full analysis in Figure 4.8), cannot be clearly determined. Similar to reflecting utterances, forward-looking utterances clearly concern the immediately succeeding activity. These forward-looking statements typically occur at the start of a chunk of activity. We can interpret them as *framing* utterances that help the viewer understand the purpose of the narrated activity that follows it. Now, if we return to segment 1 "so what I wanna do now is I wanna start filling in the lips", then we could argue that this statement frames all the subsequent chunks that are concerned with the lips, and that its reach finishes when the tutor starts work on the cheek bones in segment 46. However, rhetorical relations are defined such that they require the writer (here the tutor) to have the intention and belief that the reader (here the viewer) will understand the rhetorical relation. In our example that would mean that the tutor has a very clear idea of the future activity (span 2-45) that he is framing with segment 1. Similarly, the viewer would have to somehow realise at segment 46 that, when the concern of the process switches from the lips to the cheek bones, segment 1 framed the entire preceding activity. Both are unlikely to be the case.

However, this strict interpretation of rhetorical relations is somewhat unfair. For one, in the case of a written text, a reader does not necessarily remember, after reading a section, all the sentences that were written to prepare the reader for reading that section. Also, a tutor can prepare a viewer for some activity by simply stating that they have turned their attention to creating some specific part of the artefact. This does not require the tutor to have in mind or to have prepared beforehand all the activity that is to follow. In a sense it is an open-ended statement of concern, to be completed at a later point in time. In other words, the stating of concern could be seen as generative, rather than descriptive, of future activity. Thus, we interpret these framing utterances in two ways. Firstly, they indicate genuine shifts of the tutor's attention to a different part of the construct or content. Secondly, by making these shifts explicit the tutor maintains coherence throughout the tutorial.

The question then arises how these shifts of attention are organised. In the example we can see a hierarchical organisation. First there is the overall concern of the lips, and then several more specific concerns related to creating the different parts and details, such as the overlap. Because we have a general and more detailed concerns we can imagine a forward hierarchy, whereby if a concern is stated then it is the preparation of all the more specific, but still related, concerns that come after it. So, in our example that would mean that the chunks in span 2-44 can be related as a multi-nuclear sequence, for which the viewer is prepared by segment 1. Similarly, we would have segment 46 as a preparation for the span 47-77.

We have been careful to say this is a hierarchy of concerns, rather than a hierarchy of actions. Our definition of action is that it is the concrete events the tutors intend and directly cause in the software. As such, actions cannot be organised hierarchically, but only sequentially (ignoring parallelism for now because there is little of it in the interaction with computers). Even in imagined form actions remain sequential. We thus have a hierarchy of concerns, or perhaps goals, and not of actions.

We should also be careful in what we read into this supposed hierarchy of concerns. That is, we should make a distinction between its conception and perception. As analysts, we see this hierarchy in the analysis of the tutorials and, furthermore, we see it in its entirety. It would be tempting to think that the hierarchy, as we perceive it, is conceived by the tutor as the structure of their practice before they embark on it. However, that would mean the hierarchy

would have been conceived in its entirety beforehand. We cannot accept this, because it is about as plausible as the tutor having planned the sequence of all actions before they were taken.

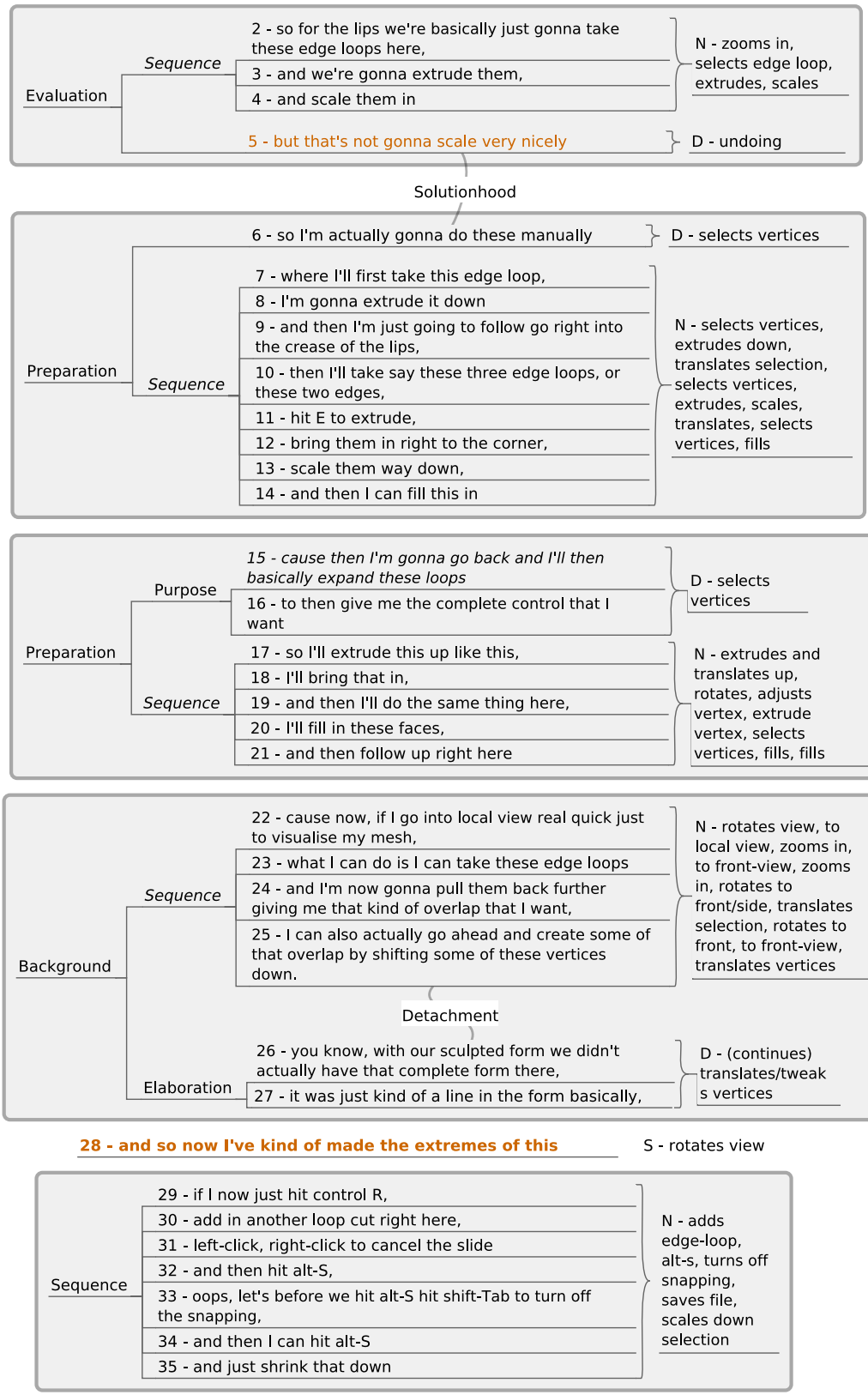
More plausible is the notion that the hierarchy we perceive is a description of how the sequence of concerns have unfolded. In a practical creative process a creator will not often start with the smallest of concerns, but rather with more abstract ones. There are exceptions, of course: ideas can spring from the mindless doodling on the back of an envelope or tinkering on a piano. However, turning those ideas into artefacts much more sizeable than a simple melody cannot be achieved from the bottom up; a more structured approach is required. However, the actions that lead to the creation of the artefact are concrete and as such the abstract concerns need to be turned into similarly concrete ones. In this line of reasoning we can view the hierarchy of concerns as a description of the tutor's attention moving from abstract to concrete concerns and back. This movement would be analogous to the movement of content to construct.

A tutor's concerns are partly dictated by the type of content they have in mind. For example, a cartoon character has a body, that body has a face, and that face has lips. Starting at the lips would be impractical, and perhaps cognitively unnatural. Another way the concerns develop will be dictated by reflection at the pivot points. A third influence on how the concerns unfold might be prior knowledge: the way a creator has been taught to do it or has found useful or efficient over time.

We will not speculate further on this notion, but instead reiterate that we are able to find a forward hierarchy of concerns. Also, we cannot be sure this hierarchical structure is the generative structure of practice, although we can take it to be descriptive of it. Furthermore, even though our RST framework is able to provide us with the mechanism to form this hierarchy, it is still difficult to fit in the reflecting utterances, for reasons discussed in Section 4.3.1.3.

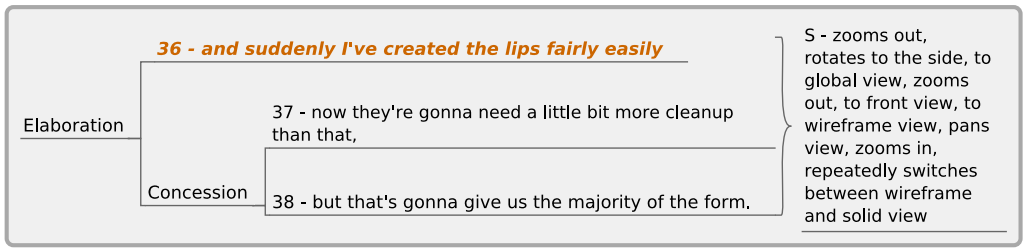
1 - so what I wanna do now is I wanna start filling in the lips,

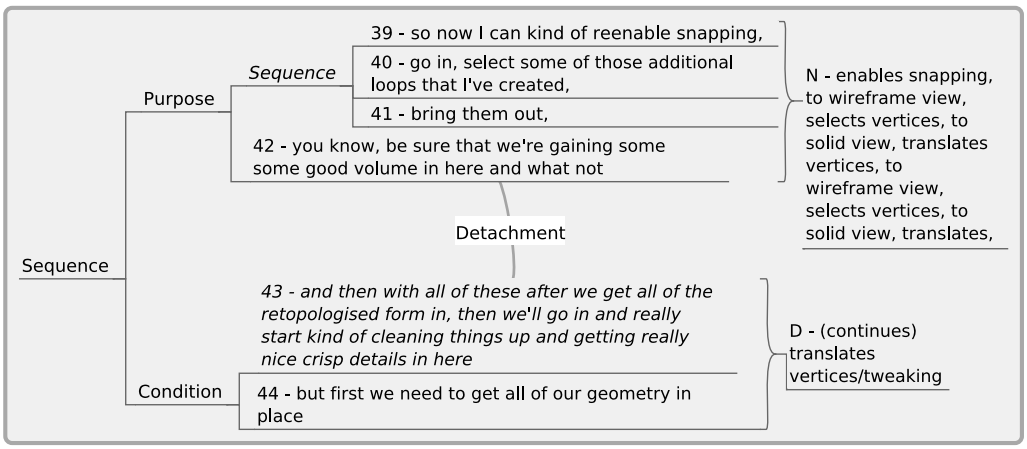
P - moves cursor along the lips



28 - and so now I've kind of made the extremes of this

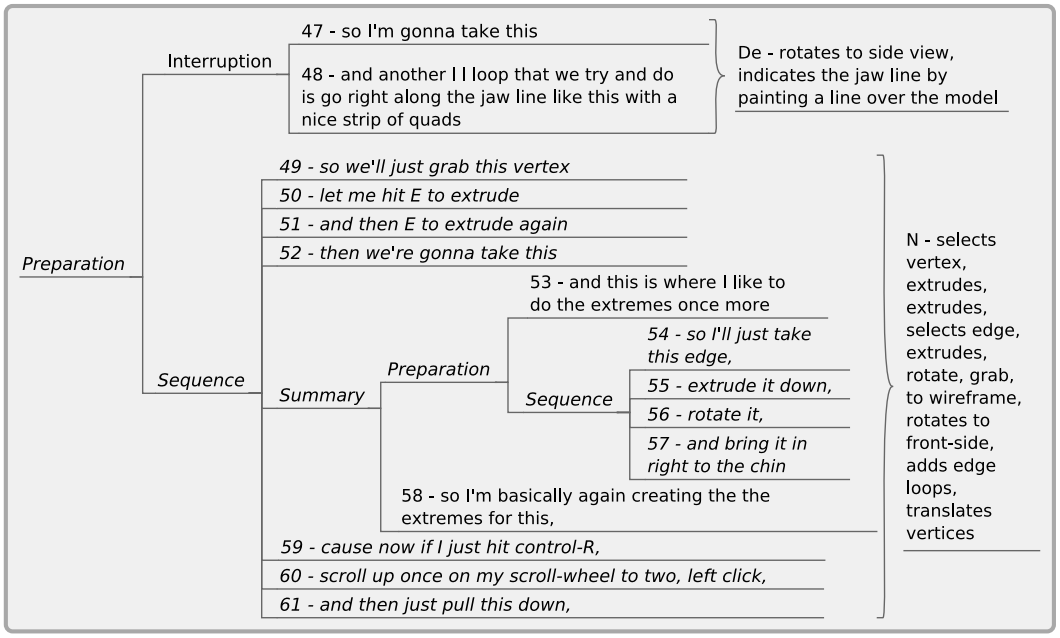
S - rotates view





45 - alright so that's looking really good, — S - zooms out, rotates view around face

46 - let's now bring down the cheek bones, — D - selects vertices



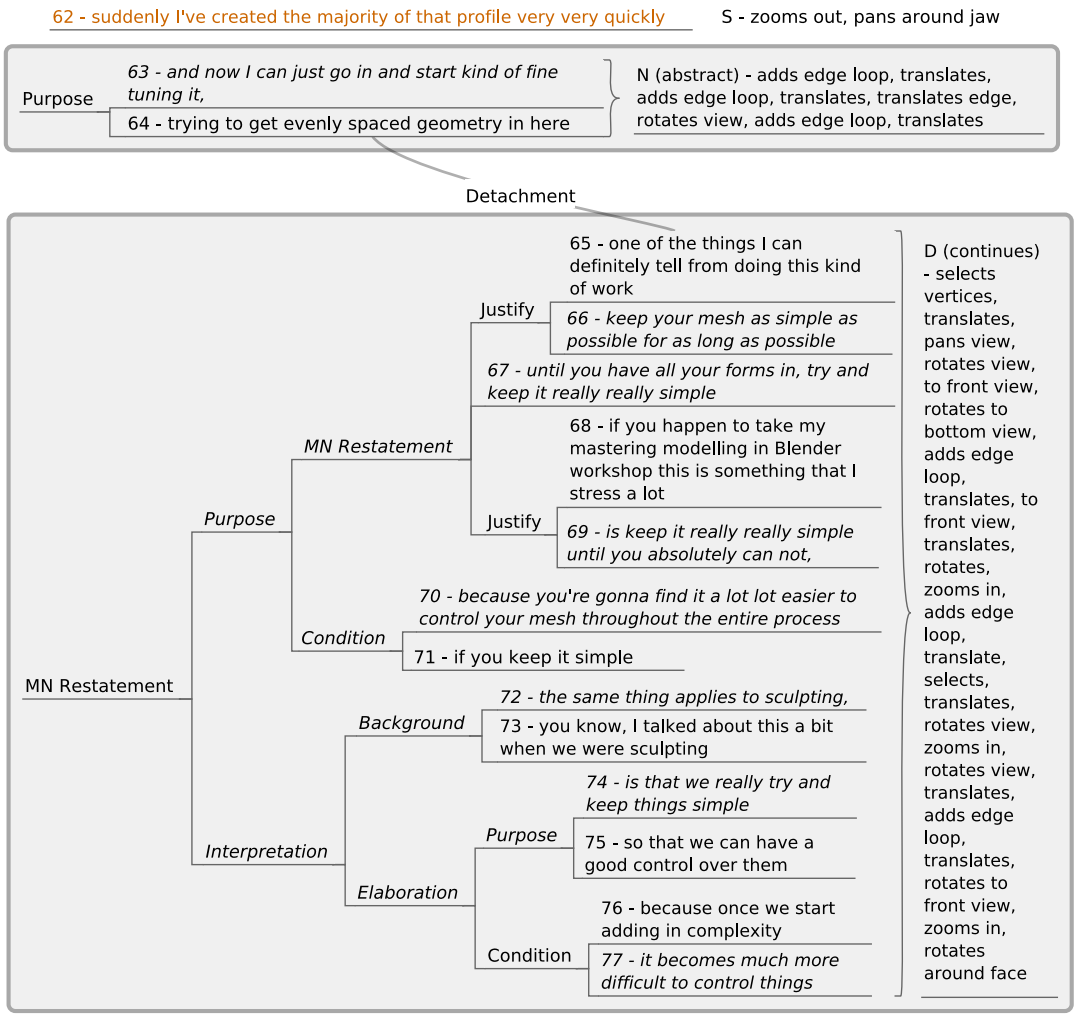


Figure 4.8: Part of a tutorial analysis of a tutorial transcription. The speech segments are numbered. To the right of the speech the actions of the tutor are transcribed as well. The relations between speech and action are marked by the letters N, D, P, S, and De. The rhetorical relations that hold between spans of the transcription are depicted to the left of the speech segments. The nucleus of each speech relation is marked by italic text. The speech segments marked in orange are reflecting statements. The chunks that are formed by the rhetorical relations are circumscribed by the grey boxes. Finally, the curved links with the text *detachment* and *solutionhood* between segments indicate some special points in this example.

4.3.1.5 Forming chunks

All the analytical tension points (pivot points, detachment, reflection, and framing) that we have discussed in the previous sections cause us to separate segments or group them together, and this allow us to perceive chunks of activity. To reiterate, in the case of pivot points we find that there are segments of the tutorial that are both backward and forward-looking at the same time, whereby the pivot is equally related to the preceding and succeeding chunks. Also, detachment allows us to group together activity where there is an apparent incoherence in the speech, finding coherence across modalities. Furthermore, framing statements introduce chunks, but they do not relate back to the preceding chunks, and reflecting statements are at times related to preceding activity, whereas at other times they relate to the state of the artefact only. In Figure 4.8 we have marked all the chunks we perceive in grey.

With this in mind we can now present an idealised chunk schema. That is, a chunk of activity starts with a framing, then the activity is narrated, after which there might be some tweaking and detached speech, with the chunk being completed by a reflection. The framing of a chunk might be extremely succinct or not marked at all. If we, for example, look at segments 15, 22, and 29, then we see that the starts of the chunks are marked with utterances such as "cause now…" or "if I now…". The speech and action are then synchronised during the narration. We also do not see a detachment in every detected chunk, but if it happens it is accompanied by tweaking activity. Reflective utterances are necessarily detached from concurrent action because they concern either the constructs, content, or past action. Also, chunks are not always followed by a reflection. For example, the shorter chunks 6-14 and 15-21 lead right into the next framing utterances. So, we see then that detachment typically occurs near the end of a chunk of activity, whereas it attaches with the start of the narration of activity. This schema explains the coherence within a chunk of activity, but it does not explain the relation between chunks.

In RST there is an incoherence if there is no rhetorical relation, and vice versa. We have not related the grey chunks to each other, but does that mean there are no relations between them? The potential relations of the forward hierarchy of concerns have already been outlined in Section 4.3.1.4. However, this does not account for the relation between chunks 6-14, 15-21, 22-27, and 29-35. Here we see chunks that are minimally framed at their starts and reflected on at their ends. We certainly cannot say that they are related by pivot points. However, these chunks are simply related by their order, that is they can be related by a sequence relation.

In fact, taking into consideration all the RST analyses performed, and not just the example here, the predominant rhetorical relation that can be applied is the sequence relation. In other words, the structure of the tutorials is such that one thing happens after another. This is unsurprising given that the creative practical process, as can be observed from the interaction with the software, is simply a sequential process. The tutors that teach this practice are therefore bound to a presentation that largely follows this sequential structure, with the consequence that the communication is structured mostly sequentially instead of hierarchically.

4.4 Gesturing in presentation

In Section 4.3 we introduced our findings with regard to the temporal structure of the process. The analysis was done by looking at the structure of the presentation of the process. However, because our focus was on the larger temporal structures we disregard the finer details of the manner in which the process is presented by tutors. In this section we will focus on those

details, specifically the way tutors gesture.

It is important to keep in mind that tutors themselves are not visible in the tutorial videos, but only the Blender software and the mouse pointer. Some tutorials will have an additional overlay that will show a textual representation of the keyboard shortcuts the tutor presses. If the tutors are not themselves visible, do tutors employ gesturing in their communication and if so how? We break this question down along the lines of the Action-Speech relation taxonomy presented in Section 3.4.1.4.

As layed out in the taxonomy, not all actions a tutor takes with the mouse pointer are done to advance the state of the artefact. Frequently the movements of the mouse are purely communicative. The clearest of these are pointing gestures. Much like pointing in face to face communication these gestures serve to draw attention to a particular part of the artefact or user interface. They are accompanied by a verbal deictic expression such as "here" or "there". In the tutorials pointing is always done by repetitively moving the mouse pointer along the area that the tutor is talking about (e.g. circling or moving backwards and forwards along a line). We presume that the pointing gesture has this repetitive quality because it would not stand out as a gesture enough if the mouse pointer was simply held still in a single position. Although an extended arm in face to face communication is quite noticeable as a gesture, the equivalent in a tutorial video would possibly not be. It is worth noting that pointing in a software interface can be done quite precisely as tutor and viewer are certain they have the same viewpoint.

Showing gestures are similar to pointing gestures in that they serve to draw the attention of the viewer to a particular part of the artefact. Instead of hovering the mouse pointer, however, they will change the view on the artefact such that a particular part or quality of it is better visible, while referring to it in speech. The analog equivalent would be to pick up an object, turn it round, and present it to the interlocutor. Showing gestures are excursions; they will start from a particular view to which the tutor will return after the gesture finishes. Once returned to the original view, the tutor will resume the practical process. Showing gestures focus on the artefact, but do not change it.

Demoing gestures on the other hand are focused on action and will, in fact, change the artefact. The start of a demoing gesture is sometimes accompanied by a hypothetical in the tutor's speech (e.g. "If instead we wanted to …"). The tutor then proceeds to change the artefact, after which the tutor undoes all the changes and returns to state the artefact was in when the demoing gesture began. The actions are often performed quickly without much care and the undoing is never narrated. Demoing gestures, because they return to the original state of the artefact, are also excursions.

When the tutor narrates advancing action we cannot characterise the mouse movements as gestures, as they are intended primarily to change the artefact. However, some gestures are done in such a way that they can be seen to have a gestural quality to them. For example, they might be performed extra slowly or in an exaggerated manner while the speech might stress a particular point. However, when speech is detached from advancing action there are no gestural qualities to be found.

Finally, it is noteworthy that tutors never explicitly signal in speech that they are moving from action to gestures.

4.5 Discussion

In this chapter we have introduced some major themes resulting from the analysis of the video tutorials. How are we to understand these themes, when put together, as a coherent descriptive system?

At the core lies the distinction between *content* and *construct*. Content refers to the way the artefact looks once it has been rendered and is explained in terms of what it represents (e.g. a character with large ears). On the other hand construct refers to the way the content is built up. Users of Blender can only ever modify the construct level, from which the content level is functionally determined. In describing construct tutors refer to objects explicitly represented in the software (e.g. vertices, texture) or the patterns of their possible compositions (e.g. quads, edge loops).

Content and construct together make up the state of the artefact. It is precisely the goal of a practical creative process to modify the state of the artefact towards some end goal. Actions are invocations of software features to modify the construct, and in turn modifying content. Past, current, and future actions are all presented differently by tutors. Past actions are frequently summarised abstractly, while current actions are narrated nearly verbatim. The stating of planned future actions frames subsequent narrated actual action.

A statement of planned future action typically does not include references to software features, but is instead concerned with either a goal content or construct state. The framing of activity is therefore best understood as explaining the reason for the subsequent concrete actions. Goals at the construct level are subordinate to goals at the content level, as a desired content state leads to a desire for a construct state.

It is clear also that tutors have a clear conscious notion of Blender's workings and features. As such they can explain clearly how, given a construct goal, which commands in the software need to be invoked.

This then gives us a hierarchy of goals, where content goals break down into construct goals, which in turn break down into action goals. Action goals are concrete enough that they can be executed, yielding a change in the construct and thereby also in the content.

It would however be a mistake to think of this chain of reasoning as being perfect. Even tutors are often seen to make mistakes in their reasoning. For example, a construct goal they imagined would yield some content in fact does not, or a particular course of action does not yield the construct that they thought it would. The chain of reasoning is perhaps best understood as a chain of hypotheses, and therefore abductive reasoning, on the part of the creator. We can see evidence of this uncertainty in the behaviour of *acting to see* also: evaluation and judging are only necessary if creators have partial predictions of what construct certain actions would result in, and what content certain constructs would yield.

To finish this chapter we will take a step into the hypothetical and consider a question for which the answer cannot come from the analysis of video tutorials. In framing planned activity tutors describe the activity that is to follow in abstract terms. The question is, do they have in mind all the subsequent activity, and they are stating a summary, or do they only have a vague (vague as in underspecified) idea of what is to follow, and they will work it out on the way? It is the latter that seems more likely to us, especially taking into account the uncertainty present in the predictions tutors make.

If we were to assume that any creator, tutors included, start a practical creative process from a vague idea at the content level, then we would have to characterise the entire process as fleshing out the detail one concrete action at a time. From a vague idea at the content level a

creator would hypothesise that it could be attained by some idea, potentially also vague, at the construct level. From the vague construct idea, they would have to hypothesise action goals. The concrete action goals would then be executed and the resulting artefact state interpreted and judged against the goals at the content, construct, and action levels. In this process Blender is the tool that allows the creator to take these iterative steps. With each step the vagueness of the idea is reduced as they are committed to the artefact.

It is the characterisation of the practical creative process as a process of reduction of vagueness by means of a tool that we will formalise in the next chapter.

Chapter 5

A Framework for Software-based Creative Practice

In this chapter we will formalise the grounded theory presented in Chapter 4. We will abstract away from the domain of Blender and in doing so produce a formal theory, named the Software-based Creative Practice Framework (SbCPF) that applies to creative practice mediated by a software tool in any domain.

The formal theory is expressed as a Category Theory construction as per the method described in Section 3.3.4. The first section describes a model of a software tool. The subsequent sections extend this model by introducing components describing a practically creative agent. Having introduced all the necessary components we give a formal definition of an agent within the theoretical framework introduced in this chapter. The final section discusses the model and compares it to related work.

5.1 An abstract model of creative software tools

Four concepts from the grounded theory from Chapter 4 are relevant to the abstract description of a creative software tool in general: actions, software features, construct, and content.

Actions are events instigated by the user and interpreted by the software to be the invocation of some command with values for its parameters. For example, in Blender you could move an object +120 units along the X axis. This would be an invocation of the "translate" command with the value 120 for the x parameter, and 0 for the other axes. The "translate" command is the software feature and the action is an invocation of that command.

The tool interprets a user's keystrokes and mouse movements and determines what type of command is being invoked and with which parameters. The interpreted action is included in the model of the tool's state. In category theoretical terms let $\mathcal{S}_a$ be the category which has all possible invocations as objects and has no morphisms.

It is typically the case that a user works on a document that can be loaded from disk, changed with the software tool, and then saved again. The document corresponds to what the grounded theory calls construct (see Section 4.2.1). Let $\mathcal{S}_3$ be the category of possible constructs, which has an object for every possible artefact that could be made with the tool. Given any two constructs $s_3, s_3' \in \mathcal{S}_3$ there is a morphism in $\mathrm{Hom}_{\mathcal{S}_3}(s_3, s_3')$ for every action that transforms s_3 into s_3'.

For each *construct* there is a corresponding *content*. For example, in the case of Blender

a particular document will result in a particular image or a movie. In the case of a music sequencer a document might correspond to an audio waveform. Let $\mathcal{S}_2$ be the category of all possible content level objects for a given tool.

Because there is a corresponding $s_2 \in \mathcal{S}_2$ for each $s_3 \in \mathcal{S}_3$ we can define a functor $R : \mathcal{S}_3 \to \mathcal{S}_2$. In Blender R would be rendering the document to an image, whereas in a music sequencer it would be bouncing down the audio to disk or playing it back in real time. Because the same content can be achieved by different constructs, we cannot define a valid functor from $\mathcal{S}_2$ to $\mathcal{S}_3$.

The morphisms of $\mathcal{S}_2$ represent the possible transformations that the tool allows, exactly like the morphisms in $\mathcal{S}_3$. That is, if there is a morphism between $s_3, s_3' \in \mathcal{S}_3$, then there is a corresponding morphism between $R(s_3)$ and $R(s_3')$.

The value of a creative software tool lies in its ability to transform constructs quickly. Each parameterised command corresponds to the execution of some functionality on the current document state. So, given the combination of the current state $s_3 \in \mathcal{S}_3$ and an invocation $s_a \in \mathcal{S}_a$ the software can produce a new state $s_3' \in \mathcal{S}_3$. Let $E : \mathcal{S}_a \times \mathcal{S}_3 \to \mathcal{S}_3$ be the functor that represents all the tool's construct transforming features. Let $\pi_1 : \mathcal{S}_a \times \mathcal{S}_3 \to \mathcal{S}_a$ be simply the projection of the tuple of invocation and construct to only the invocation.

With the categories $\mathcal{S}_a$, $\mathcal{S}_3$, $\mathcal{S}_2$, the product category $\mathcal{S}_a \times \mathcal{S}_3$, and the functors R, E, and π_1 we can draw the diagram as depicted in Figure 5.1 to represent the software tool.

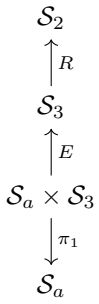


Figure 5.1: Diagram representing the different types of state and functionality of a creative software tool.

The model of the tool is fairly straightforward and the model of the creative agent will roughly mirror it. That is, each of the levels of state in the tool at the content, construct, and action levels ($\mathcal{S}_2$, $\mathcal{S}_3$, and $\mathcal{S}_a$ respectively) will be reflected in the agent. The functors $R : \mathcal{S}_3 \to \mathcal{S}_2$ and $E : \mathcal{S}_a \times \mathcal{S}_3 \to \mathcal{S}_3$ also have counterparts in the creative agent. In the following sections model of the agent will be built up bit by bit.

5.2 Views, representations, and schemas

5.2.1 Views

The different ways in which creators view and judge content and construct were introduced in Section 4.2.1.4 and Section 4.2.2. One salient point to highlight here is that content and construct are two separate levels at which the artefact is seen and judged. Content is not just an abstraction or compression of construct because content contains information imparted by

the render functionality E. Also, tutors have different concerns depending on which of the two levels they focus on. It is therefore that we must model the way creative agents view and represent these two levels of state separately.

We start by stating that an agent has a means of viewing state and that the act of viewing state results in the agent having some internal representation of that state. Representations-in-mind are not necessarily equal or isomorphic to the tool states because agents might see incorrectly, incompletely, or disregard some information present in the states. We say the agent has a view $V_3 : \mathcal{S}_3 \to \mathcal{R}_3$, where $\mathcal{R}_3$ is the category of representations the agent could have of tool constructs. In similar fashion we have $V_2 : \mathcal{S}_2 \to \mathcal{R}_2$ that maps content state to an internal representation $r_2 \in \mathcal{R}_2$. In addition, let $V_a : \mathcal{S}_a \to \mathcal{R}_a$ be the view that the agent has of any performed action as interpreted by the tool.

5.2.2 Representations

The representational categories $\mathcal{R}_2$, $\mathcal{R}_3$, and $\mathcal{R}_a$ do not only serve to model the states of the tool as observed by the agent. Their objects are also what an agent holds as goal states. An agent might have $g_2 \in \mathcal{R}_2$ as a goal at the content level, and derived from it a $g_3 \in \mathcal{R}_3$ at the construct level, resulting in a $g_a \in \mathcal{R}_a$ at the action level.

Before we formally define each of the representational categories, we have to consider what their constructions are supposed to encode. Firstly, any concept should be able to be expressed and therefore a representational category should contain all possible representations. That is, the framework should not impose any restrictions on what an agent can hold in their mind as a result of viewing an artefact nor should there be any restrictions on what an agent can imagine as their creative goal. There are of course limits to what an agent, human or artificial, can hold in their minds, but that lies outside of the current scope. Also, representations should allow for underspecification. There are two reasons for this requirement. Firstly, when an agent views the tool state they might not be able to see everything as some details might not be visible. In Section 4.2.2 this is described as *acting to see*. Secondly, when an agent imagines a goal artefact at the start of a creative process they might not know every single detail yet. The goal will be vague in some respects, meaning that certain details, perhaps most, are still undefined.

5.2.3 Schemas and instances

One can only be aware that a detail is undefined if there is something that says that detail should or could exist. In other words, there must be some prior knowledge of a schema that encodes the general structure of representations. Roughly, a schema defines what types of objects exist and what relations hold between them. In our case we do not wish to limit what can be in the mind of the agent and so there is not just one schema, but any number of them. For each schema there might be an infinite number of possible instances that follow its structure.

What a schema is exactly and what we mean by underspecification can be made precise with the help of the Grothendieck construction (Spivak 2014, p. 363). Let any schema be a category $\mathcal{C} \in \mathbf{Cat}$. An instance of the schema $\mathcal{C}$ is a functor $I : \mathcal{C} \to \mathbf{Set}$. I maps each type of object $c \in \mathcal{C}$ to a set $X \in \mathbf{Set}$ and so instantiates the inhabitants X of type c. Furthermore, the morphisms in the schema $\mathcal{C}$ are the types of relations between the inhabitants of two types. For example, a relation $r \in \mathrm{Hom}_{\mathcal{C}}(c, c')$ defines that every object in the set $I(c)$ is mapped by $I(r)$ to an object in the set $I(c')$. There can be any number of relations between two types, including none at all.

As an example imagine that a schema $\mathcal{C}$ contains two types of objects: Table and Waiter. Let the relation $a \in \text{Hom}(\text{Table}, \text{Waiter})$ represent that a table is assigned a particular waiter. Let $I : \mathcal{C} \to \mathbf{Set}$ be an instance of $\mathcal{C}$ that represents the set of tables and the set of waiters in our local pub, as well as the assignments of waiters to tables. That is, $W = I(\text{Waiter})$ is the set of waiters in our pub, $T = I(\text{Table})$ the set of tables, and $I(a)$ picks out the assigned waiter from W for each table in T.

This way of letting functors encode instances of schemas is the Grothendieck construction. Because a category can be anything it is easy to see how our first requirement is fulfilled: any thought can be thought. However, this does not yet give us underspecification generally.

To achieve this second requirement we place an extra condition on instance functors. An instance $I' : \mathbf{Cat} \to \mathbf{Set}$ maps each type $c \in \mathcal{C}$ of a schema $\mathcal{C} \in \mathbf{Cat}$ to a set of inhabitants $X \in \mathbf{Set}$. That set X must always contain the special object $*$ which means *undefined*. That is, if in an instance I' the value for a relation $r : c \to d$ of a particular inhabitant $x \in I'(C)$ is to be left undefined then $I'(r)(x)$ must map to $*$. Furthermore, for any relation $r' \in \text{Hom}_{\mathbf{Set}}(c, d)$ it must hold that $I'(r')(*) = *$.

5.2.4 Representational categories

Having defined what a valid instance is for any schema we are ready to define the representational categories $\mathcal{R}_2$, $\mathcal{R}_3$, and $\mathcal{R}_a$, starting with $\mathcal{R}_2$. The objects of $\mathcal{R}_2$ are the representations and are simply all valid instances for all possible schemas. Let $B_2 : \mathcal{R}_2 \to \mathbf{Cat}$ be the functor that picks out the schema on which the representation is based.

The representational categories are now defined in a way that allows representations to have undefined details. It is not in itself very useful to define a category that contains any and all possible representations, but we can put the morphisms between them to good use. The morphisms in $\mathcal{R}_2$ are to indicate abstraction. That is, given any two representations $I, I' \in \mathcal{R}_2$ there will be a single morphism $f \in \text{Hom}_{\mathcal{R}_2}(I, I')$ if and only if I' is an abstraction of, or equal to, I. Informally, I' is an abstraction of I if, for any detail in I, I' has the same value as I or if the value is $*$.

Formally, whether $I \leq I'$ (I' is an abstraction of I) is defined as follows.

$$I \leq I' \quad \equiv \quad I' \subseteq I \quad \wedge \quad I' \approxeq I \tag{5.1}$$

$$I' \subseteq I \quad \equiv \quad \bigwedge_{c}^{\mathcal{C}} . \; I'(c) \subseteq I(c) \tag{5.2}$$

$$I' \approxeq I \quad \equiv \quad \bigwedge_{c,d}^{\mathcal{C}} \; \bigwedge_{r}^{\text{Hom}_{\mathcal{C}}(c,d)} \; \bigwedge_{x}^{I'(c)} . \; I(r)(x) = I'(r)(x) \; \vee \; I'(r)(x) = * \tag{5.3}$$

There exists exactly one morphism $f : I \to I'$ *iff* $I \leq I'$. There are two necessary further conditions for abstraction. Firstly, $I' \subseteq I$ expresses that I must have all inhabitants that I' has. The fact that I can have additional inhabitants makes that we are working under the assumption of an open world, meaning that the absence of an inhabitant in a representation does not mean that it should not exist. Secondly, $I' \approxeq I$ expresses that I must have the same values for its relations as I', unless I' specifies a value of $*$.

The morphisms between representations point only in the direction of abstraction. All representations map to the entirely undefined representation $*$. According to this definition $\mathcal{R}_2$ is a partial order, meaning that the morphisms indicate an ordering. In this case the representations are ordered by abstraction relations. The order is called partial because not

all pairs of objects can be compared; an ordering is only defined for a pair of objects if there is a morphism defined between them. Because $\mathcal{R}_2$ is a partial order we use the notation $I \leq I'$ if I' is an abstraction of I. It is worth noting that the partial order morphisms are natural transformations as they map between representations that are themselves functors. The diagram in Figure 5.2 depicts an example partial order for a simple schema.

The categories $\mathcal{R}_3$ and $\mathcal{R}_a$ are constructed in the same way as $\mathcal{R}_2$. This, in fact, makes them all the same category. We maintain different labels because there might be specific constructions for each of the representational categories in future versions of the model.

Because it will come in handy later, we define the subcategory of instances for a particular schema $\mathcal{C} \in \mathbf{Cat}$. Let $R^f_\mathcal{C}$ be the subcategory of the representational category $\mathcal{R}$ that contains only fully defined representations of the schema $\mathcal{C}$. The subcategory $\mathcal{R}^*_\mathcal{C}$ contains only instances that have at least one relation that maps to $*$. The objects of $R^f_\mathcal{C}$ and $R^*_\mathcal{C}$ are defined as follows.

$$\mathrm{Ob}(\mathcal{R}^f_\mathcal{C}) = \{I \in \mathcal{R} \mid B(I) = \mathcal{C} \wedge \mathrm{full}(I)\} \tag{5.4}$$

$$\mathrm{Ob}(\mathcal{R}^*_\mathcal{C}) = \{I \in \mathcal{R} \mid B(I) = \mathcal{C} \wedge \neg\mathrm{full}(I)\} \tag{5.5}$$

$$\mathrm{full}(I) \equiv \bigvee_{c,d}^{\mathcal{C}} \bigvee_{r}^{\mathrm{Hom}_\mathcal{C}(c,d)} \bigvee_{x}^{I(c)} . \; I(r)(x) \neq * \tag{5.6}$$

With these initial categories and functors we can extend the diagram in Figure 5.1 to include categories and functors that make up the agent. Figure 5.3 now include each of the representational categories and the views that map tool state to them. Also, included is a functor $a : \mathcal{R}_a \to \mathcal{S}_a$ that represents an agent's ability to effect change by acting out an action-in-mind.

5.3 Moving between representations

Now that all the main categories have been presented we can move towards explaining the functors in the model that play a key role in creative practice. Taken collectively, these functors model the ability of an agent to work from an abstract idea to an actual artefact. In doing so the agent reduces the entropy of the artefact-in-mind by adopting information from the developing artefact.

5.3.1 Imagination

Imagination plays a vital role in the model and it is seen as the primary way in which a practical creative process is driven forward. When an agent imagines they take a representation, at any of the levels, and fill in details that were left undefined previously.

Informally, if we were to visualise a representational category as an acyclic graph, with the completely undefined representation at the top, and the completely defined representations at the bottom, then we could see imagination as traversing the graph from top to bottom. Because the representational category is a partial order the edges of the graph correspond to the morphisms of the category, directed upwards.

Formally, imagination is modelled by an endofuctor T on a representational category R. Because there are three representational categories ($\mathcal{R}_2$, $\mathcal{R}_3$, and $\mathcal{R}_a$) we also have three imagination functors $T_2 : \mathcal{R}_2 \to \mathcal{R}_2$, $T_3 : \mathcal{R}_3 \to \mathcal{R}_3$, $T_a : \mathcal{R}_a \times \mathcal{R}_3 \to \mathcal{R}_a \times \mathcal{R}_3$. An imagination functor $T : R \to R$ is valid if and only if $T(I) \leq I$, where $I \in R$. That is, an imagination functor

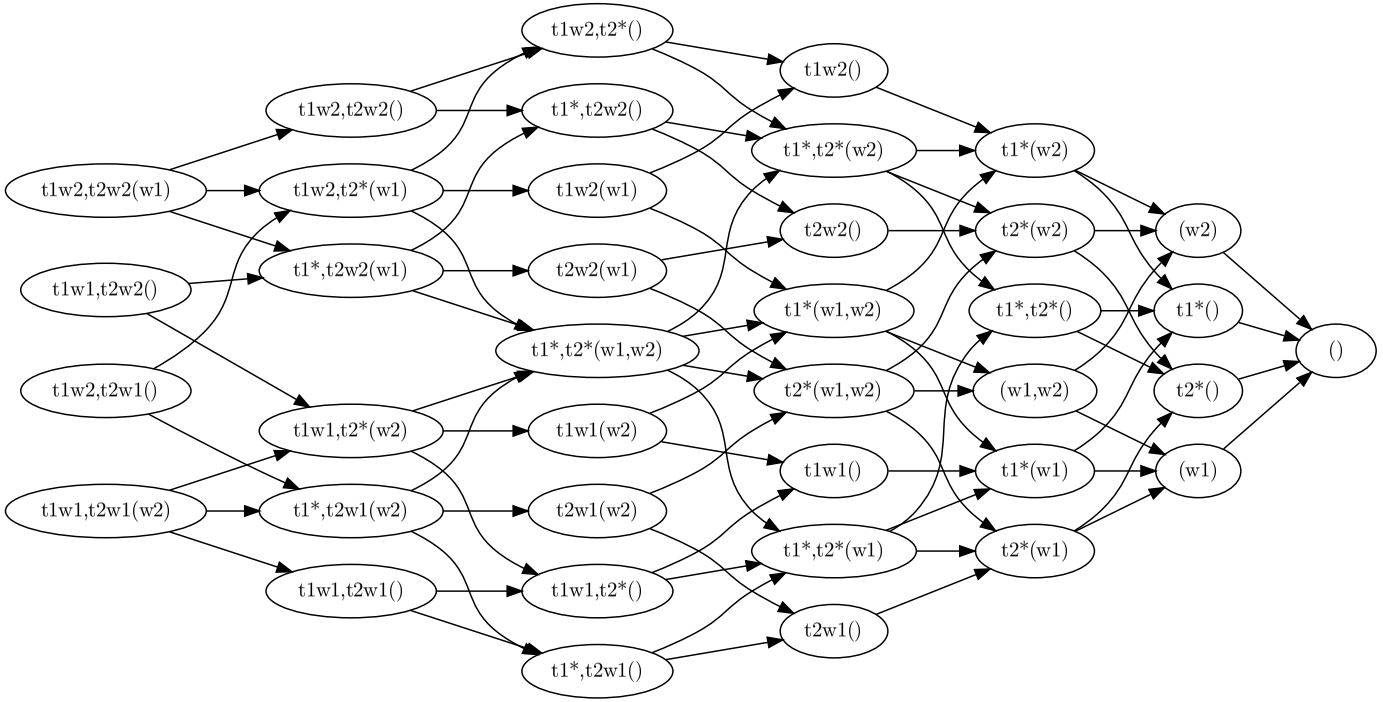


Figure 5.2: This graph depicts a small part of the representational category for a single schema. The schema has two objects, Table and Waiter, and a single morphism $a : \text{Table} \to \text{Waiter}$ that assigns a waiter to a table. Each node in the graph represents an instance of the schema with the titles encoding the specific instance. Inhabitants of Table are named t1, t2, etcetera and inhabitants of Waiter are named w1, w2, and so on. The text before the parentheses shows which tables exist in the instance and to which waiter they are assigned. The text "t1w1,t2w2" means that waiter 1 is assigned to table 1 and waiter 2 to table 2. The text "t1w2" means that there is only one table and it is assigned to waiter 2. The symbol $*$ is used when a table's assignment is undefined. The text in parentheses encodes the idle waiters, which are those that exist but have not been assigned to any tables. The nodes on the left are fully defined, while the node on the right is fully undefined. Nodes in the middle have at least some information defined. The edges point in the direction of abstraction, and they are transitive. That is, if a node A points to a node B, and node B points to a node C, then C is an abstraction of A, even though it is not depicted explicitly with an edge in the graph. To keep the graph small it only contains instances with a maximum of two tables and two waiters. It was generated with the code from Listing B.1.

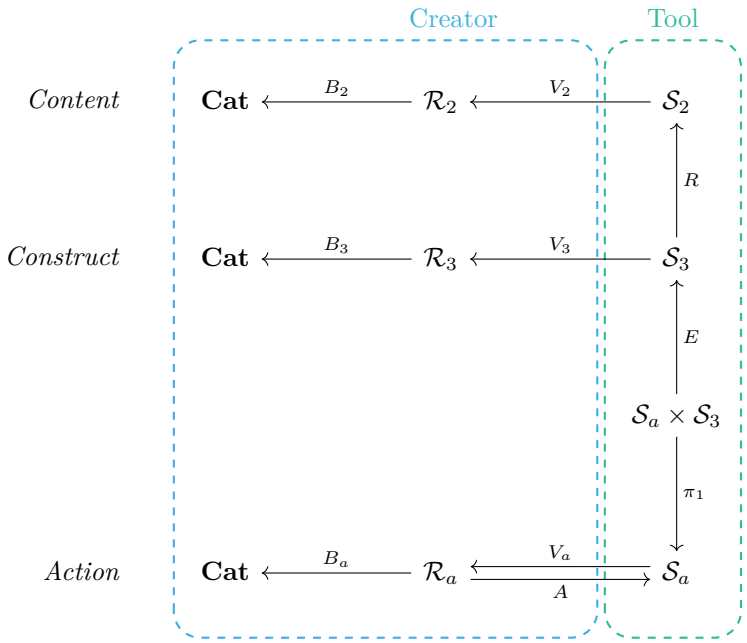


Figure 5.3: The extended model diagram includes tool states, tool functors, agent view functors, and representation categories. The categories and functors describing the agent are circumscribed by the blue border. Tool categories and functors are circumscribed by the green border. The text labels on the left indicate what is being represented by the tool states and agent representations.

may not reduce information. This is the only restriction placed on imagination functors. The model does not specify how the representational categories should be traversed, and indeed an agent might know several ways that could be employed when deemed appropriate.

5.3.2 Information content and entropy in representations

Although the model does not specify the manner of traversal, it does allow to describe some aspects of a particular traversal functor. Representations are expressed in terms of the schema that they are instances of, and it is therefore that we can say something about the amount of information and entropy present in a representation. The information quantifies how much of the detail is defined, while the entropy quantifies how much detail is yet to be determined. If these measures of information and entropy present in a representation could be quantified, then the amount of entropy reduced by an imagination step could simply be calculated.

To give the definitions of the entropy and information content functors we need a new partial order on a category of representations $\mathcal{R}$. This order is denoted with $\preceq$. It indicates whether an instance is a *strict* abstraction of another and it is defined as follows.

$$I \preceq I' \quad \equiv \quad \bigforall_{c,d}^{\mathcal{C}} \bigforall_{r}^{\mathrm{Hom}_{\mathcal{C}}(c,d)} \left[I'(c) = I(c) \ \wedge \ \bigforall_{x}^{I(c)} \left[I'(r)(x) = I(r)(x) \ \vee \ I'(r)(x) = * \right] \right] \tag{5.7}$$

An instance $I \preceq I'$ if it has the same inhabitants as I', but only for those types that are the subject of relations, and if I' assigns the same values as I or $*$. As an example, the instance **t1*()** from Figure 5.2 is a strict abstraction of **t1*(w1)**, **t1*(w2)**, **t1*(w1,w2)**, **t1w1()**, **t1w2()**, **t1w1(w2)**, and **t1w2(w1)**. Intuitively, **t1*()** has a single relation undefined and it is the abstraction of only the instances that define precisely that relation.

We first define a functor $H : \mathbf{Cat} \to \mathbb{R}$ that maps a schema to a real number representing the amount of entropy inherent in the schema. The entropy $H(\mathcal{C})$ of some schema $\mathcal{C} \in \mathbf{Cat}$ is the expected amount of information that needs to be provided to produce a finished artefact, as per the schema. The entropy functor follows Shannon's definition (Claude E Shannon 1938) and so $H(\mathcal{C}) = \mathbf{E}\left[-log_2\left(P^f(\mathcal{C})\right)\right]$, where $\mathbf{E}$ gives the expected value and $P^f(\mathcal{C}) : \mathcal{R}^f_{\mathcal{C}} \to \mathbb{R}$ is a probability density function (pdf) that defines the probability of any fully defined representation of the schema $\mathcal{C}$ occurring. In this case the schema $\mathcal{C}$ is seen as a random variable that could take, with a certain probability, the value of any of the fully defined instances of the schema. We view $P^f(\mathcal{C})$ loosely as modelling the agent's experience of seeing artefacts that are examples of the schema. For example, if the schema was that of a car, then the car is more likely to be a typical hatchback rather than a one with six wheels.

How do we apply this reasoning to representations that are only partially defined? Let $P(\mathcal{C}) : \mathcal{R}_{\mathcal{C}} \to \mathbb{R}$ assign a probability to any representation of $\mathcal{C}$:

$$P(I = \mathcal{C}) = \sum_{I'}^{\mathcal{R}^1_{\mathcal{C}}} \begin{cases} P^f(I' = \mathcal{C}) & \text{if } I' \preceq I \\ 0 & \text{otherwise} \end{cases} \tag{5.8}$$

where $\mathcal{R}_{\mathcal{C}}$ is the subcategory of a representational category $\mathcal{R}$ containing only instances of $\mathcal{C}$. In words, to find the marginal probability of any representation I, $P(\mathcal{C})$ sums together the probabilities of all fully defined representations for which I is a *strict* abstraction (or to which I is equal). We will define the probability of the wholly undefined representation as $P(* = \mathcal{C}) = 1$, meaning that it is an abstraction of all other representations of that particular schema.

Now that we can view the representational subcategory $\mathcal{R}_{\mathcal{C}}$ as a random variable that can take on the values of wholly or partially defined representations we can define the information content and entropy of individual representations. Firstly, let $J(\mathcal{C}) : \mathcal{R}_{\mathcal{C}} \to \mathbb{R}$ be the information content of a representation $I \in \mathcal{R}_{\mathcal{C}}$ in bits, where $J(I = \mathcal{C}) = -log_2(P(I = \mathcal{C}))$. It is easy to verify that the information content of the wholly undefined representation $J(* = \mathcal{C}) = 0$ as $P(* = \mathcal{C}) = 1$. The information content of a completely defined representation is always greater than or equal to the information content of any of its non-strict abstractions. Generally, $I \leq I' \Rightarrow J(I = \mathcal{C}) \geq J(I' = \mathcal{C})$. In other words, the information content of a representation increases monotonically with every detail that becomes defined.

With the help of J the entropy functor $H(\mathcal{C})$ can be redefined as follows.

$$H(I = \mathcal{C}) = \sum_{I'}^{\mathcal{R}^1_{\mathcal{C}}} \begin{cases} P(I' = \mathcal{C}) \times J(I' = \mathcal{C}) & \text{if } I' \prec I \\ 0 & \text{otherwise} \end{cases} \tag{5.9}$$

The entropy functor $H(\mathcal{C})$ defines that there are zero bits entropy in any representation that is fully defined. Furthermore, the entropy inherent in the schema $H(\mathcal{C})$ is simply the entropy of the fully undefined representation $H(* = \mathcal{C})$. Whereas the information content increases monotonically with a decrease in non-strict abstraction, entropy only decreases monotonically with a decrease in *strict* abstraction. That is, $I \preceq I' \;\Rightarrow\; E(I = \mathcal{C}) \leq E(I' = \mathcal{C})$.

With this definition of information content and entropy it becomes clear how an imagination functor might increase and decrease entropy as it moves down the non-strict abstraction lattice. By defining values for relations the entropy is reduced and information content is increased. By instantiating new inhabitants with undefined values for its relations the entropy is increased and the information content stays the same. Imagination functors can increase the complexity

of an instance by instantiating new inhabitants and defining their properties.

5.3.2.1 Imaginative power

With the quantification of the amount of information present in a representation, as well as the amount of entropy still to be reduced, it is possible to introduce the concept of imaginative power. That is, an agent is limited in the amount of information they can imagine and keep in working memory (Baars and Gage 2010, p.34).

For example, let $s_2 \in \mathcal{S}_2$ be a content level tool state and let $V_2(s_2)$ be the agent's internal representation of this state. By traversing the representational category $\mathcal{R}_2$ with T_2 the agent can imagine representations derived from $V_2(s_2)$ that contain more information. A limit l in imaginative power would constrain the representations that could be reached in that the difference between the information contents of both could not go above some threshold. That is, imagination is constrained in such a way that $J(T_2(V_2(s_2))) - J(V_2(s_2)) \leq l$.

A limit in an agent's imaginative power provides an explanation for the need to externalise representations and progress from the viewed externalised state. With the adoption functors introduced in Section 5.4.2 it is possible to make the concept of imaginative power precise for cases where an imagined representation $T(V(s)) \not\leq V(s)$.

5.3.3 Moving from Content, to Construct, to Action

At the very start of creative practice, when the agent is met with a blank canvas, they have a representation in mind at the content level $r_2 \in \mathcal{R}_2$. This representation could either be fully defined according to the schema $B(r_2)$, or it could be somewhat vague and leave certain details to be decided through creative practice. In either case the agent will need to come to some decision about the first action to take; the first of many in creating a construct $r_3 \in \mathcal{R}_3$ that will yield the imagined content r_2.

The question at hand is the following. Which types of functors do we need in our framework to model how an agent might move from the content level $\mathcal{R}_2$, to the construct level $\mathcal{R}_3$, and then arrive at the action level $\mathcal{R}_a$? In other words, if an agent has some idea of what they want the artefact to look or sound like, what construct do they think they need, and how do they build it?

5.3.3.1 From Content to Construct

To move from content to construct the agent will need to form a hypothesis. That is, given a certain goal representation $r_2 \in \mathcal{R}_2$, the agent hypothesises that if it were to produce an artefact it would view at the construct level as $r_3 \in \mathcal{R}_3$, then they would view that artefact as r_2 at the content level. In other words, the agent has to find an r_3 that it thinks looks like r_2. To be precise, the hypothesis is of the following form. The agent holds a goal content state $r_2 \in \mathcal{R}_2$, as well as the hypothesised construct state $r_3 \in \mathcal{R}_3$, whereby the agent believes that r_3 relates to r_2 in such a way that for any tool construct state $s_3 \in \mathcal{S}_3$, with corresponding tool content state $s_2 \in \mathcal{S}_2$, whereby $V_3(s_3) \leq r_3$, it must be the case that $V_2(s_2) \leq r_2$. Note that it must also necessarily be the case that $H(V_3(s_3)) \leq H(r_3)$ and $H(V_2(s_2)) \leq H(r_2)$.

We might also consider weaker hypotheses whereby the agent, given a goal state $r_2 \in \mathcal{R}_2$, adopts a goal state $r_3 \in \mathcal{R}_3$ for which it believes that there exists some $r'_3 \leq r_3$ that corresponds to an $r'_2 \leq r_2$. This represents the belief that from r_3 the agent can get to $r'_3 \in \mathcal{R}_3$, but not just yet. An agent could adopt such a weaker hypothesis because it might be limited in the

amount of detail in can conjure up purely mentally and without the use of externalisation of the intermediate representation (see Section 5.3.2.1).

We use the word *hypothesis* because agents do not perfectly predict the workings of their tools. Their reasoning about what construct a certain action will result in, or what a certain construct will look like, is imperfect. Therefore, it is unlikely that they can plan out the construction of an artefact from start to finish without error. The use of the tool to externalise a goal state is not just the implementation of an idea, it is part of the formation and elaboration of the idea.

The framework includes two functors to describe how the agent finds the hypothesis. Let $G_2 : \mathcal{R}_2 \to \mathcal{R}_3$ be a functor that generates the seed $o_3 \in \mathcal{R}_3$ from which the agent will search for the hypothesis. Let $C_2 : \mathcal{R}_3 \to \mathcal{R}_2$ be the functor that predicts what a construct-in-mind looks like at the content level. Given a goal content representation $r_2 \in \mathcal{R}_2$ the agent can map this to a starting point for the search $G_2(r_2)$. The agent can then employ its imagination to find an $r_3 = T_3(G_2(r_2))$ for which it hypothesises that it will yield r_2. That is, the following inequality should hold.

$$(C_2 \circ T_3 \circ G_2)(r_2) \leq r_2 \tag{5.10}$$

It is important to keep in mind that a particular wholly defined $r_2 \in \mathcal{R}_2$ can often be achieved by more than one $r_3 \in \mathcal{R}_3$. That means that when G_2 picks a starting point for the search in $\mathcal{R}_3$ it could do so in a way that leaves some possible solutions unreachable. The most optimal seed for the search could be defined as the most maximally informative representation in $\mathcal{R}_3$ from which all solutions could still be reached. Formally, $G_2(r_2)$ is the join $\bigsqcup\{r_3 \in \mathcal{R}_3 \mid C_2(r_3) \leq r_2\}$ (see box 5.1 for the definition of a join). In contrast, the least informative representation from which all solutions can still be reached is $*$. This constraint is also captured by stating that $r_2 \leq (C_2 \circ G_2)(r_2)$, for any $r_2 \in \mathcal{R}_2$. In fact, C_2 and G_2 are adjoint functors and form a Galois connection. This means that it must hold that $C_2(r_3) \leq r_2$ if and only if $r_3 \leq G(r_2)$.

It is clear then that G_2 is best defined in terms of C_2, and that is just as well, because C_2 has a clear connection to the real world. It is the agent's understanding of the render functionality $R : \mathcal{S}_3 \to \mathcal{S}_2$ of the software tool. The quality of the agent's hypothesis in moving from *Content* to *Construct* is therefore mostly dependent on how accurately they can predict the workings of the tool.

There is one important constraint that prediction functors should adhere to. A prediction functor should maintain consistency of abstraction: if $r_3 \leq r_3'$, with $r_3, r_3' \in \mathcal{R}_3$, then $C_2(r_3) \leq C_2(r_3')$. This condition is in fact already inherent in the definition of a functor, but it is worth pointing out.

5.3.3.2 From Construct to Action

When an agent has formed a hypothetical goal construct state $r_3 \in \mathcal{R}_3$, they must still form a hypothetical action plan $r_a \in \mathcal{R}_a$. This is modelled in much the same way as moving from *content* to *construct*, albeit with one important difference.

When moving from *construct* to *action* the agent takes into account the current observed construct state $V_3(s_3)$, where $s_3 \in \mathcal{S}_3$ represents the current tool construct state. We therefore have our generator functor $G_3 : \mathcal{R}_3 \times \mathcal{R}_3 \to \mathcal{R}_a$ and the predictor functor $C_3 : \mathcal{R}_a \times \mathcal{R}_3 \to \mathcal{R}_3$[1].

[1]Note that for cosmetic reasons the functors C_3 and G_3 are shown slightly different in the diagram infig:ct-full-model

Box 5.1: Meets and joins

Meets and joins are two important constructs in partial order categories. They are notions of the greatest lower bound and least upper bound of two given objects in the partial order. Let, a, b be objects from a partial order. The object $m = a \sqcup b$ is the the meet of a and b iff $m \leq a$ and $m \leq b$ and there is no other object x such that $x \leq a$, $x \leq b$, and $m \leq x$. A join $j = a \sqcap b$ of two objects a and b is the object for which it holds that $a \leq j$, $b \leq j$, and there is no other object x such that $a \leq x$, $b \leq x$, and $x \leq j$.
In our representational categories the join is defined for any pair of representations, but the meet might not necessarily exist. The join and meet of three or more objects is simply the recursive application of the binary join or meet.
Meets and joins in a partial order category are examples of the more general concepts of products and co-products respectively. Other examples of products and co-products are the cartesian product and disjoint union in Set, and the union and intersection in a power set category.

The imagination functor at the *action* level is $T_a : \mathcal{R}_a \times \mathcal{R}_3 \rightarrow \mathcal{R}_a \times \mathcal{R}_3$.

Like C_2 represents the agent's model of the tool's render functionality, C_3 models the state transforming features of the tool $E : \mathcal{S}_a \times \mathcal{S}_3 \rightarrow \mathcal{S}_3$. The chain of hypothetical reasoning is entirely modelled in terms of the agent's understanding of the tool. Our model therefore expresses very clearly how the tool is incorporated into the thinking of the agent.

An imagined action can be physically performed by interaction with the keyboard and mouse. The functor $A : \mathcal{R}_a \rightarrow \mathcal{S}_a$ models an agent's ability to perform actions.

5.3.3.3 Composite imagination

In Section 5.3.1 the imagination functors T_2, T_3, and T_a were presented as a way of traversing the representational categories as partial orders. Section 5.3.3.1 highlighted that the representations found by the traversal functor should also be considered in light of higher level goals (see Equation 5.10). This shows that there is another way of imagining at a particular level, and that is by borrowing from the imagination functor at lower levels. For example, in addition to $T_2 : \mathcal{R}_2 \rightarrow \mathcal{R}_2$ we can construct $T_2^{'} = C_2 \circ T_3 \circ G_2$ and $T_2^{''} = C_2 \circ C_3 \circ T_a \circ G_3 \circ G_2$. This construction accounts for sparks of imagination that can come from mindless doodling or conceptual ideas that spring from thinking about techniques.

5.4 Closing the loop

As soon as an agent has formed a concrete enough hypothesis they are ready to act and advance the artefact. The action-in-mind $r_a \in \mathcal{R}_a$ is performed and results in a command $s_a = A(r_a)$ as interpreted by the tool. The tool subsequently executes the command resulting in the changed construct state $s_3 = E(s_a, s_3')$ and the corresponding content state $s_2 = R(s_3)$. So far so good, but how does the change in tool state feed back into the agent's reasoning? Firstly, the outcome of the action is a test of the hypothesis chain and it can confirm the prediction as either correct or incorrect. Secondly, the agent incorporates the information present in the tool state into its own goal representations.

5.4.1 Testing the hypothesis chain

After acting comes viewing, and so the three levels of tool state, $s_a \in \mathcal{S}_a$, $s_3 \in \mathcal{S}_3$, and $s_2 \in \mathcal{S}_2$, can be mapped, by the view functors, to corresponding representations $r_a \in \mathcal{R}_a$, $r_3 \in \mathcal{R}_3$, and $r_2 \in \mathcal{R}_2$ respectively. The agent can now compare the goal states to the perceived states and can judge whether its predictions were correct.

The first thing to test is whether the performed action was the intended one. That is, is it the case that $V_a(s_a) \leq r_a$? If it is, then we can check the hypotheses. If it is not, then either something went wrong in the performance of the action (e.g. pressed the wrong key) or the agent does not know how to operate the tool's interface correctly.

The first hypothesis to check is whether $V_3(s_3) \leq r_3$. That is, did the imagined action yield a tool state that fulfils the goal construct state r_3? If so, then we can test the second hypothesis. If not, then there are two possible options. The agent might still achieve the construct goal by acting further if $r_3 \leq V_3(s_3)$. If neither $r_3 \leq V_3(s_3)$ or $V_3(s_3) \leq r_3$, then the agent has simply made a wrong prediction. Checking the second hypothesis, whether $V_2(s_2) \leq r_2$ or vice versa, follows the same reasoning.

This order of testing the hypothesis chain, action first and content last, is necessary to determine whether the predictions themselves were right. For example, if the action was performed badly then we could not say much about the hypotheses in the first place. However, it is not necessary to test in this order to have a rational agent. Instead, an agent could simply check whether it is happy with the perceived content state $V_2(s_2)$ in relation to r_2. If it fits the goal, then the agent might not care about the fact that it came about through faulty predictions. An agent can be lucky and rational at the same time.

5.4.2 Adoption

Because the representational spaces form partial orders the agent is capable of determining whether a viewed tool state fits with its goals, as explained in the previous section. The question is then how an agent incorporates the information present in the tool state into its goal representations.

A key feature of the model is that an agent can have underspecified goals. Goals are underspecified because either the agent does not care about certain details, because they simply have not thought about them, or because they were unable to imagine all details because of limited imaginative power. It is, however, possible that by advancing the artefact certain details become defined that were not cared about or not predicted. Details that were undefined in goal representations can take on the values from the viewed artefact. By the goal representation taking on those values the agent is effectively adopting information from the environment and from the tool.

To model this adoption of information from the external artefact let us introduce a set of new functors. Let $M_2 : \mathcal{R}_2 \times \mathcal{R}_2 \to \mathcal{R}_2$, and $M_3 : \mathcal{R}_3 \times \mathcal{R}_3 \to \mathcal{R}_3$ be the adoption functors that merge a goal representation with a perceived tool state into a new goal representation by adopting information from the perceived tool state.

There is a relatively simple way in which we can rationally define the general adoption functor $M : \mathcal{R} \times \mathcal{R} \to \mathcal{R}$ for a representational category $\mathcal{R}$. We consider four separate cases and see what a rational strategy for merging two representations might be in each of them. Let $r \in \mathcal{R}$ be the goal representation and let $V(s) \in \mathcal{R}$ be the perceived tool state. The first case is when $V(s) \leq r$, meaning $V(s)$ has the same information as r and more. It would be a good strategy to adopt the information from $V(s)$ and let it be the new goal state from which

to imagine further details. An equally simple case is when $r < V(s)$, meaning that $V(s)$ is not yet as defined as r, but it at least has some of the same information as r. In this case there is no information to adopt. The third and fourth cases are when neither $V(s) \leq r$ or $r \leq V(s)$. The distinction between these last two cases lies in whether r is *compatible* with $V(s)$.

Two representations are compatible if there exists some third $r' \in \mathcal{R}$, such that $r' \leq r$ and $r' \leq V(s)$. Effectively this means that none of the information in r and $V(s)$ contradicts, even though there might be some overlap in the information that they contain. In these case the two representations can be *merged* by taking the meet $r' = r \sqcap V(s)$ (see box 5.1 for the definition of a meet), producing a new representation that contains the information from both. It is worth noting that this is the opposite of the join that is used to define the generator functors, whereby information is retained only if it is present in both. The definition of the adoption functors as taking the meet works for all cases set out above, except when two representations are not compatible. In this fourth case it is not quite as clear what to do. The goal and tool state representations are incompatible when an agent has made a prediction error and the resulting tool state is not as expected. An agent could not adopt any information and try again, it could abandon the goal, or it could adopt only non-contradicting details, to name a few options.

The defined adoption strategy above is quite strict in that it adheres strictly to the goals an agent has set for itself. It could however be reasonable to define non-strict adoption strategies. For example, an agent could be "lazy" in that it would rather forsake its goal and continue with the current artefact state as is. The defined adoption strategy is also "greedy" in that it adopts all information from the viewed artefact. It is possible for an agent to be more cautious in adopting information from the artefact into its goal representation, perhaps because it would like to mull over the details more attentively.

5.5 A higher meaning

There is one more category, and related functors, to introduce before we present the diagrammatic version of the complete model. The already introduced representational categories $\mathcal{R}_2$, $\mathcal{R}_3$, and $\mathcal{R}_a$ each correspond to a type of tool state. As the model stands, the agent will work from a content goal representation $r_2 \in \mathcal{R}_2$, abduce an $r_3 \in \mathcal{R}_3$ and $r_a \in \mathcal{R}_a$ to work towards achieve a content state $s_2 \in \mathcal{S}_2$ that satisfies the content goal. To reiterate, the content goal describes what the artefact is to look or sound like and at times a creator will start from such a goal. However, at other times a creator's goal will lie on a more abstract plane that does not directly refer to how things look or sound.

For example, a cartoonist might have the *conceptual* goal of satirising a politician's hypocritical stance on the issue du jour. This does not describe anything visual and, as a representation, lives in the world of human interpretation. Just as a construct goal serves the content goal, the content goal serves the *concept* goal.

The hypothetical chain described in Section 5.3.3 should thus be extended by one link. We introduce the category $\mathcal{R}_1$ to model the existence of conceptual goals. With $\mathcal{R}_1$ come the generator functor $G_1 : \mathcal{R}_1 \to \mathcal{R}_2$, the prediction functor $C_1 : \mathcal{R}_2 \to \mathcal{R}_1$, $T_1 : \mathcal{R}_1 \to \mathcal{R}_1$, and the basis functor $B_1 : \mathcal{R}_1 \to \mathbf{Cat}$. The representational category $\mathcal{R}_1$ is used as a catch-all for any meaning that is abstracted away from the content level. This meaning lives solely in human interpretation and so the model does not include any functor $V_1 : \mathcal{S}_2 \to \mathcal{R}_1$ that maps directly from artefact to the conceptual level.

All the categories and functors have now been introduced. The full diagrammatic version

of the model is presented in Figure 5.4.

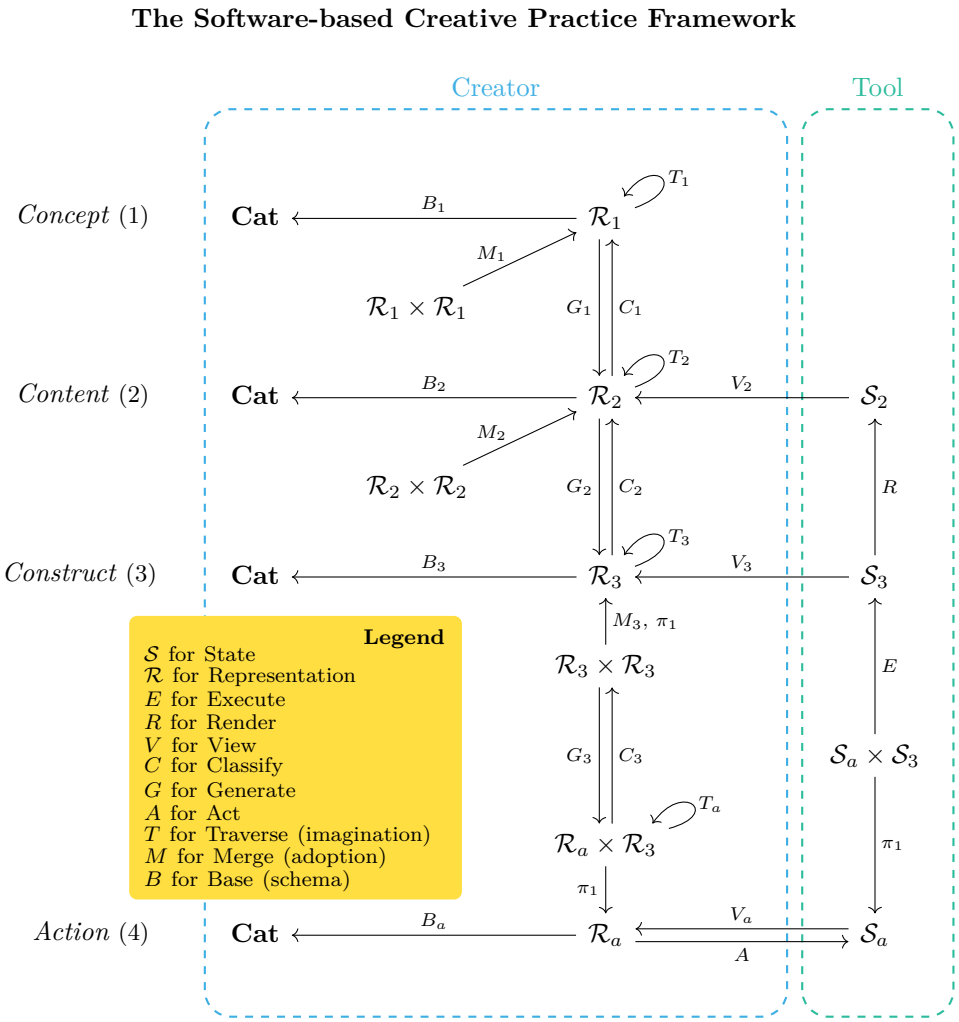


Figure 5.4: The complete Software-based Creative Practice Framework model in diagrammatic form.

5.6 Agent definition

The diagram in Figure 5.4 depicts all the categories and functors defined until now. It is, however, not yet a definition of a creative agent. It is tempting to interpret the diagram as some sort of flowchart, but it really only defines the components necessary to formally define an agent, or a software tool for that matter. This section will give formal definitions for both.

5.6.1 Formal definition of a creative software tool

The formal definition of a creative software tool is not far off what can be seen in Figure 5.4. Let **Tool** be the category of all creative software tools. A particular tool $t \in \mathbf{Tool}$ is a tuple $(\mathcal{S}_a, \mathcal{S}_3, \mathcal{S}_2, E, R)$ of a category of possible actions $\mathcal{S}_a$, a category of possible constructs $\mathcal{S}_3$, a category of possible renders or exports $\mathcal{S}_2$, and two functors $E : \mathcal{S}_a \times \mathcal{S}_3 \to \mathcal{S}_3$ and $R : \mathcal{S}_3 \to \mathcal{S}_2$ that define the software application's algorithms. **Tool** does not have any morphisms.

It will be useful to define a given tool's state space as a category. Given a particular tool $t \in \mathbf{Tool}$, where $t = (\mathcal{S}_a, \mathcal{S}_3, \mathcal{S}_2, E, R)$, let $\mathbf{State}_t$ be the category where the objects are the possible tool states. Specifically,

$$\mathrm{Ob}(\mathbf{State}_t) = \{(s_a, s_3, s_2) \in \mathrm{Ob}(\mathcal{S}_a \times \mathcal{S}_3 \times \mathcal{S}_2) \mid R(s_3) = s_2\}\,. \tag{5.11}$$

This can be understood as a combination of a construct, its corresponding content, and a chosen but not yet executed command. The morphisms in $\mathbf{State}_t$ are defined by the tools transformations E. That is, for any two states $s, s' \in \mathbf{State}_t$, where $s = (s_a, s_3, s_2)$ and $s' = (s'_a, s'_3, s'_2)$, there is exactly one morphism in $\mathrm{Hom}_{\mathbf{State}_t}(s, s')$ if and only if $s'_3 = E(s'_a, s_3)$. The observable part of creative practice can be modelled by a path through $\mathbf{State}_t$.

5.6.2 Formal definition of a creative agent

The formal definition of a creative agent is somewhat more involved than that of a creative software tool. Let $\mathbf{Agent}_t$ be the category of all possible agents working with the tool t. An agent $a \in \mathbf{Agent}_t$ is a combination of view, imagination, adoption, generator, prediction, and perform functors. For each of the concept, content, construct, and action levels the agent has a goal representation. Finally, given a tool $t \in \mathbf{Tool}$ the agent has a behaviour policy $L_t : \mathbf{Agent}_t \times \mathbf{State}_t \to \mathbf{Agent}_t$ that models the agent's reasoning in the cycle of viewing and acting. That is, in determining the next concrete action an agent still has to decide when to view, when to imagine, when to predict, act, and so on. Formally,

$$\begin{aligned}
&\mathrm{Ob}(\mathbf{Agent}_t) = \mathrm{Ob}(\\
&\quad \mathcal{R}_1 \times \mathcal{R}_2 \times \mathcal{R}_3 \times \mathcal{R}_a && \textit{goals} \\
&\times \mathrm{Fun}(\mathcal{S}_2,\ \mathcal{R}_2) \times \mathrm{Fun}(\mathcal{S}_3,\ \mathcal{R}_3) \times \mathrm{Fun}(\mathcal{S}_a,\ \mathcal{R}_a) && \textit{views } V_x \\
&\times \mathrm{Fun}(\mathcal{R}_1,\ \mathcal{R}_2) \times \mathrm{Fun}(\mathcal{R}_2,\ \mathcal{R}_3) \times \mathrm{Fun}(\mathcal{R}_3 \times \mathcal{R}_3,\ \mathcal{R}_a \times \mathcal{R}_3) && \textit{generators } G_x \\
&\times \mathrm{Fun}(\mathcal{R}_2,\ \mathcal{R}_1) \times \mathrm{Fun}(\mathcal{R}_3,\ \mathcal{R}_2) \times \mathrm{Fun}(\mathcal{R}_a \times \mathcal{R}_3,\ \mathcal{R}_3 \times \mathcal{R}_3) && \textit{predictors } C_x \\
&\times \mathrm{Fun}(\mathcal{R}_1,\ \mathcal{R}_1) \times \mathrm{Fun}(\mathcal{R}_2,\ \mathcal{R}_2) \times \mathrm{Fun}(\mathcal{R}_3,\ \mathcal{R}_3) && \textit{imagination } T_x \\
&\times \mathrm{Fun}(\mathcal{R}_a \times \mathcal{R}_3,\ \mathcal{R}_a \times \mathcal{R}_3) \\
&\times \mathrm{Fun}(\mathcal{R}_1 \times \mathcal{R}_1,\ \mathcal{R}_1) \times \mathrm{Fun}(\mathcal{R}_2 \times \mathcal{R}_2,\ \mathcal{R}_2) && \textit{adopters } M_x \\
&\times \mathrm{Fun}(\mathcal{R}_3 \times \mathcal{R}_3,\ \mathcal{R}_3) \\
&\times \mathrm{Fun}(\mathcal{R}_a,\ \mathcal{S}_a) && \textit{perform } A \\
&\times \mathrm{Fun}(\mathbf{Agent}_t \times \mathbf{State}_t,\ \mathbf{Agent}_t)) && \textit{policy } L_t
\end{aligned} \tag{5.12}$$

where $\mathrm{Fun}(\mathcal{X}, \mathcal{Y})$ is the category of functors from $\mathcal{X}$ to $\mathcal{Y}$. An agent $a \in \mathbf{Agent}_t$ is thus of the following form:

$$\begin{aligned}
a = (&r_1, r_2, r_3, r_a,\ \ V_2, V_3, V_a,\ \ G_1, G_2, G_3,\ \ C_1, C_2, C_3, \\
&M_1, M_2, M_3,\ \ T_1, T_2, T_3, T_a,\ \ A,\ \ L)
\end{aligned} \tag{5.13}$$

The morphisms in $\mathbf{Agent}_t$ model the decision-making as well as learning an agent undergoes. There is a morphism in $\mathrm{Hom}_{\mathbf{Agent}_t}(a, a')$ for every state $s \in \mathbf{State}_t$ where $L_t(a, s) = a'$ and L_t is the policy of a.

With this definition different sorts of agent policies can be discussed. For example, stubborn agents that do not "learn" and only reason about goals and actions are those that only change

the goal factors and non of their functors. Agents that learn to predict better with experience are those that "change" their predictor functors C_x. At a meta-level, learning new behaviour can be seen as a policy adapting itself. We do not give a concrete definition for L_t or even contend that there is a single right one. However, Chapter 6 investigates a simple algorithm and how it behaves in relation to behaviour observed in the tutorial videos.

5.7 Discussion

This chapter presented a formalisation in Category Theory of the grounded theory from Chapter 4. The formal theory abstracts away from the focus on Blender. Also, it has a stronger focus on the cognition of creative practice. Some concepts in the formal model were already clearly framed in the last chapter, an example of which is the distinction between *content* and *construct*. Other parts, such as the generation and prediction functors, are further removed from the grounded theory and therefore more speculative. This final section will discuss what is valuable in the model, and what is lacking.

A creative software tool is modelled very simply in the SbCPF. It mainly expresses that in software-based creative practice a creator works on a document format that is different from the format of the end result. This distinction is not so clearly present in, for example, painting or sculpture. An interesting extension of the model as presented would be to explicitly deal with undo and redo operations. These operators are different from any others in that they rely on the history of states rather than just the current state. If such operators are included then the category $\mathcal{S}_a$ has to be defined differently. This would then also require a more clearly defined concept of an agent's memory of historical artefact states.

Tool state is viewed by the agent, resulting in internal representations of said state. The two main qualities of the way the representational categories are constructed is that, firstly, any representation is based on a schema, and, secondly, abstraction relationships are clearly defined. Exactly what can and cannot be represented is left open and currently any "thought" can be thought. Because representations are instances of schemas it is possible to define information content and entropy present in a representation. One aspect that has not yet been clearly outlined is the possibility of an agent suddenly using a different view and thereby having a different representation of the same tool state. How this could be included in the abstraction network is an interesting question.

There are certain concepts the representation categories do not deal with generically. For example, there is no explicit method for representing the conjunction or disjunction of two representations, or negation. That is, given a schema, there is no generic way of combining two thoughts or excluding artefacts that have a certain property. The current model does not preclude such semantics from being defined by individual schemas, but an algebra on the representational category itself would provide a generic way of modelling such logic. This certainly lies within the scope of Category Theory (Abramsky and Tzevelekos 2011).

Another concept that is not explicitly catered for is that there are different types of abstraction. The current model allows for abstraction by virtue of underspecification. However, it does not cater generically for semantic abstraction. That is, the goal of making a "vehicle" and the goal of making a "tricycle" cannot be in an abstraction relation without the schema defining that they should. This is another area where the representational categories can be made much more precise, for example by integrating other theories such as *chunking* (Gobet et al. 2001). This could also lead to a more generic way of dealing with hierarchical representations

and divide-and-conquer strategies of focusing attention on only a part of a representation at a time.

The definition of measures of information content and entropy build on the existence of a basis schema and a corresponding probability distribution. Although declaring there is a probability distribution is easy, it is much harder to define one for a particular schema, especially if the schema is complicated. The predictive processing theory (see Section 2.3.6) posits that the mind has some way of modelling the probabilities of expected impulses, and the role the probability distributions in the SbCPF is not dissimilar.

In the SbCPF, an agent's imagination is bounded by giving it limited imaginative power. It is well known that there are cognitive limits, such as limits related to working memory. The integration, into the current model, of work pertaining to cognitive limits, as they relate the to imagination, is likely fruitful ground for future work.

The current model does not explicitly explain how an agent decides that one imagined representation is to be preferred over another. It only posits that agents attempt to reduce entropy. It stands to reason that the representation that reduces the most entropy is to be preferred. The information content can be of use as well. Faced with a choice between two alternatives choices for a single property of the schema the creator can pick the probable option or the improbable one. Either choice would reduce the entropy by the same amount, but the uncommon choice would have a higher information content. Wiggins (2012b)' theory of spontaneous creativity is a much more elaborated version of this idea.

The way the agent moves between representations is modelled through the imagination, generation, and prediction functors. The model does not place many restrictions on what constitutes a valid imagination and so is compatible with other theoretical frameworks (Boden 2006; Wiggins 2012b). What is significant however is that the model shows how imagination at a different level of representation can be borrowed to imagine at higher levels. This is made possible by the generation and prediction functors. However, the generation functors are defined themselves in terms of prediction, and prediction itself is the model the agent has of the world. The bottom line is that it is through the agent's understanding of the world that they are able to make high-level goals a reality. This is in line with recent theories that frame cognition as prediction (Clark 2013; Wiggins 2012b).

One marked difference from other models of creative reasoning is that the SbCPF has a strong focus on the iterative cycle of acting and viewing, similar to the engagement-reflection framework (Sharples 1999; Pérez y Pérez 1999). The SbCPF also formalises the way in which information can be adopted from the external artefact into the internal goal representations. The model describes a straightforward definition for the adoption of information, but leaves space for other styles to be defined.

We have described the practical creative process as working from an abstract idea to a real artefact. With each step the agent reduces the vagueness, or entropy, until the artefact is finished. The SbCPF defines all the components it posits are necessary for this process. However, one glaring omission is that the model does not put forward an algorithm that describes the way an agent puts these components to use. For example, when does an agent imagine? When does an agent change focus to a different level of representation, or do they all work in parallel? These are difficult questions and the model does not include an answer. In Chapter 6 the simplest of policies is presented together with the results of implementing it.

Chapter 6

Illustration and evaluation of the SbCPF through computational modelling

6.1 Introduction

The Software-based Creative Practice Framework (SbCPF) introduced in Chapter 5 presents a descriptive theory of creative practice. It claims a practically creative agent can be structured in a particular way and describes the necessary components that make up that structure. Because the SbCPF is formally described in category theory terms it can be taken as a blueprint for an artificial agent. In this chapter we take the SbCPF as a software architecture, and investigate whether we can successfully implement an artificial agent. The implementation serves partly as an illustration of the SbCPF, and partly as an evaluation.

Whether the attempt is successful depends on four factors. Firstly, if we can implement a functioning agent that follows the SbCPF then that is evidence that the SbCPF is intrinsically consistent. Secondly, if the resulting agent follows the SbCPF architecture and does not need to deviate or add significant components then that is evidence that the SbCPF provides the right abstraction for guiding the construction of artificial agents. Thirdly, an actual artificial agent will need a concrete policy (as described in Section 5.6) and so an implementation will provide evidence that a rational policy can indeed be defined and how. Finally, and most importantly, if an artificial agent shows behaviour similar to that of a human agent then that is evidence that the SbCPF is a valid descriptive theory of human creative practice.

This chapter is structured as follows. Section 6.2 sets out the method for our investigation and Section 6.3 presents the results. Section 6.4 discusses the results and evaluates whether the attempt at constructing an artificial agent has succeeded as per the four factors set out above.

6.2 Method

The method consists of three major parts: a) build a simple creative software tool, b) build and train an agent to use the tool, and c) evaluate the agent by running simulations. We will set out each part separately here.

6.2.1 Creative software tool

6.2.1.1 Requirements

A creative software tool can theoretically be any piece of software capable of producing an artefact. However, we should impose some requirements to make sure that our tool design is relevant as well as that the complexity is within bounds.

The artefacts the tool produces must be easily perceivable by an artificial agent. We opt for the visual domain and say the tool must produce images of a certain fixed size. Images are easily processed with machine vision algorithms and because of that they make a good choice for the content category $\mathcal{S}_2$.

A second requirement is that the tool construct and action categories ($\mathcal{S}_3$ and $\mathcal{S}_a$ respectively) are of low complexity. The tool must allow its user to construct hierarchical and compositional constructs. This would represent considerable complexity, and would make it very difficult for the artificial agent to perceive constructs. Therefore, we consider that this complexity lies outside of the scope of this work. We will come back to this decision in the discussion in Section 6.4.

Another requirement is that the render functionality $R : \mathcal{S}_3 \to \mathcal{S}_2$ introduces information into the artefact at the content level. If it did not then all actions would effectively be operating directly on the content state $\mathcal{S}_2$. This would be more akin to working on a real painting rather than with a software tool.

The final requirement is that the artefacts that the tool produces must not have overtly human semantics. For example, if the agent were to attempt architectural designs it would require an understanding of human use of space. By avoiding semantics grounded in human experience we avoid having to teach the agent about the human world and associated complexity.

6.2.1.2 Design

There are many ways to design a tool and fulfil the requirements set out above. For this study we have settled on a simple graphical pattern design tool. It presents the agent with a board of 3x3 empty cells, each of which can hold a tile. A tile is an image of 24x24 pixels and included in the tool are a total of 256 tiles to choose from. A particular tile may appear in more than one cell. There are therefore 256^9 possible configurations of completely filled boards.

The tiles were chosen from a large collection of just over 550,000 potential 24x24 images to cover the whole range of what the agent can see as perceivably distinct images. We will come back to this in Section 6.2.2 in the discussion of the content views.

A completed board of 3x3 cells, each filled with a tile, corresponds to an image of 72 pixels wide and 72 pixels high. A construct state $s_2 \in \mathcal{S}_2$ is a 3x3 matrix of tile identifiers, where an identifier is a number 1 to 256. A cell in the matrix can also be set to the special value $*$, which indicates that cell has not been filled. The render functor assembles the images corresponding to the specified tile ids into one image and leaves the pixels for unfilled cells transparent.

The tool has only two commands. The first allows the user to add a tile by specifying a row, column, and tile id. If a tile is added to a cell that is filled already, then the existing tile id is overwritten. The second command serves to remove a tile and is accomplished by specifying a row and column.

This simple design fulfils all the requirements. The resulting artefact is an image and the constructs are non-compositional and non-hierarchical. The render functionality introduces a lot of information by using the tool's tile images. Also, the agent can learn about abstract tile

patterns without further knowledge of human affairs.

6.2.2 Artificial agent

The SbCPF defines an agent as a collection of functors and a decision-making policy that utilises those functors (see Section 5.6). We will describe the design choices for each of the functors here, but we will leave details such as training parameters for the results section.

6.2.2.1 View functors

The SbCPF dictates we design and implement three view functors, namely V_2, V_3, and V_a (see Section 5.2.1).

Action At the action level $V_a : \mathcal{S}_a \to \mathcal{R}_a$ maps to instances of the schema $\mathcal{C}_a$, which is depicted in Figure 6.2. A schema's objects and morphisms are encoded by defining Python[1] classes. For example, Listing B.2 shows the definition of the schema $\mathcal{C}_a$ in Python code. The functor $V_a : \mathcal{S}_a \to \mathcal{R}_a$ is implemented manually and is a straightforward translation of the tool's action state to an agent schema instance. All schema instances are encoded by the agent as described in box 6.1.

Box 6.1: Encoding of schema instances

Instances of any schema are encoded by specifying inhabitants and values for their relations in a way similar to how the Resource Description Framework (RDF)[a] encodes information. For example, Listing 6.1 shows how an *Add* instance is encoded. First the schema is created by defining the parameters such as the number of rows, columns, and tiles. An inhabitant of the *add_tile* schema type is created and assigned to the variable *root*. The root inhabitant does not have a clear identity of its own and it is given an anonymous name with *Var()*. The relation values for *root* are then defined with the (subject, relation, object) triples. The name of the relation must match the name defined in the schema. The value of the relation is a tuple of a schema type and the name of a specific inhabitant of that type. The *tile* relation is not included here and is therefore interpreted as being undefined. The actual instance is then created by passing the triples and the root inhabitant to the SchemaInstance constructor.
The triples of a schema instance can be seen as the edges of a directed graph, much like in RDF. A graph encoded as triples can be easily processed by certain algorithms, like our imagination functors and entropy calculation, but not by others. For example, the triple encoding is not a great fit for neural networks that prefer vectors or matrices of numbers. The view, prediction, and generator functors all have to work with schema instances, and so we have implemented a translation from the triple encoding to vector encoding and vice versa (please see Section B.1). The principle behind the translation is to view the graph as a fixed form tree and packing it into a vector form by "squashing" it. This is a relatively simple procedure because none of the schemas have loops and because schema instances have root inhabitants. That makes that an instance is effectively a tree.
Boolean features are encoded as a single element of the vector. Other features that can take on a value from a set of two or more possible values are one-hot encoded: every possible value in the set is assigned an element in the vector. If a feature is assigned a particular value then the corresponding element in the vector is set to 1. All the other vector elements for the feature are set to 0. Each feature is preceded by a single element of value 0 or 1 indicating whether that feature is defined or not.

[a] `https://www.w3.org/TR/rdf11-concepts/`

[1] `https://www.python.org/`

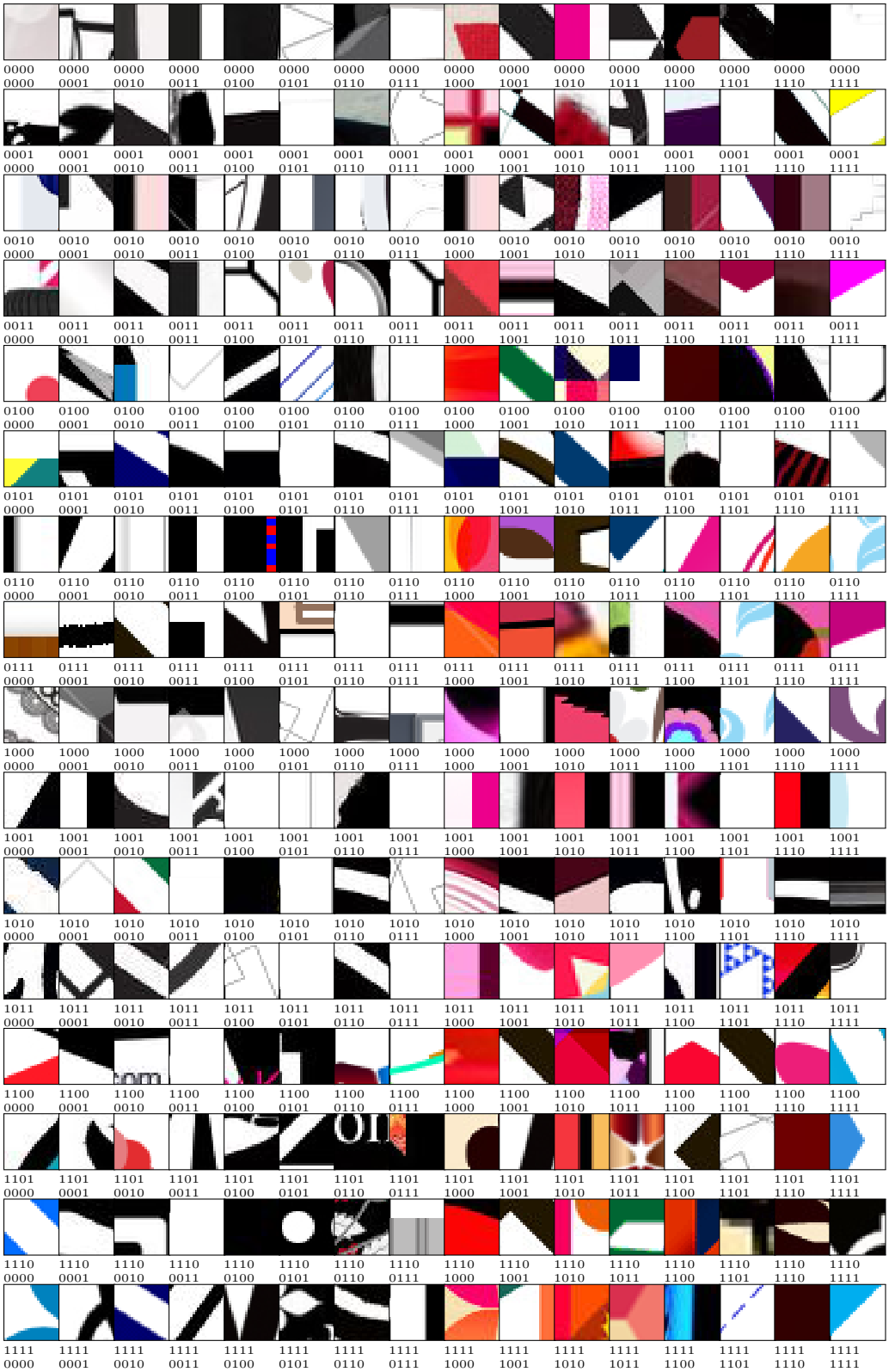


Figure 6.1: These are all the tiles available in the tool. The eight numbers below each tile are the values, true or false, for each of the eight features the agent perceives for the tile.

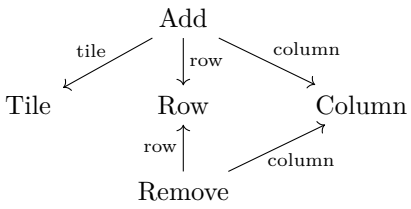


Figure 6.2: Diagram representing the schema for representations of command invocations.

Listing 6.1: An example schema instance

```
schema = SchemaAction(rows=3, columns=3, num_tiles=256)
root = (schema.add_tile, Var())
triples = [(root, 'row', (schema.row, 1)),
           (root, 'column', (schema.column, 2))]
instance = SchemaInstance(triples, root)
```

Construct At the construct level $V_3 : \mathcal{S}_3 \to \mathcal{R}_3$ maps to instances of the schema $\mathcal{C}_3$, which is depicted in Figure 6.3. An inhabitant of *Board* has 9 relations, each defining the tile for a particular cell. Listing B.3 shows the Python definition of the schema.

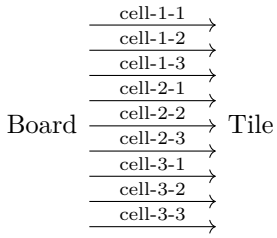


Figure 6.3: Diagram representing the schema for representations of constructs.

$V_3 : \mathcal{S}_3 \to \mathcal{R}_3$ is implemented manually and translates $\mathcal{S}_3$ to $\mathcal{R}_3$ almost verbatim. Empty cells in $\mathcal{S}_3$ are marked as undefined in $\mathcal{R}_3$.

Content The view $V_2 : \mathcal{S}_2 \to \mathcal{R}_2$ that maps the tools content to an internal representation is more interesting than the views at the lower levels. The content it views is in the form of a 72×72 pixel image, and we would like the agent's representation of such an image to be much more compressed. $V_2 : \mathcal{S}_2 \to \mathcal{R}_2$ is implemented as a neural network. This allows the view to be trained in an unsupervised manner on a large data set, learning a compression appropriate for the data.

The neural network is a combination of convolutional layers, variational autoencoder layers (Kingma and Welling 2013), and deconvolutional layers, in that order. It takes a single cell's 24×24 pixel image, compresses it to an 8-dimensional real-valued vector at the encoding layer, and then attempts to reconstitute the original image from the 8D vector. The network is trained to a) minimise the difference between the input and output images, and b) maximise the similarity of the encoding to a Gaussian distribution. This has the effect of maximising the information in the encoding while centring the distribution of each dimension around zero.

The network has two convolutional layers, each with 8 kernels of width and height of 4 and a stride of 2. This feeds into the encoding layer, which feeds into 4 deconvolutional layers that double the resolution until a 24×24 pixel image is produced that resembles the input, provided the autoencoder works well.

The 8-dimensional encoding of the image is used to determine the agent's internal representation of the image. Each dimension is seen as a feature of the image that can be either true or false. If the value for a dimension is less than or equal to zero the value for the feature is false, and if the value is greater than zero the value for the feature is true. In other words, the agent has an 8 bit description of the $24 \times 24 \times 4 \times 8 = 18432$ bit image. The agent can thus perceive 2^8 distinct tiles.

The neural network encodes a single tile at a time and so $V_2 : \mathcal{S}_2 \to \mathcal{R}_2$ uses it 9 times, once for each cell of the board, resulting in a $9 \times 8 = 72$ bit description of the board. The content schema that $V_2 : \mathcal{S}_2 \to \mathcal{R}_2$ maps to is shown in Figure 6.4 and its definition can be seen in Listing B.3.

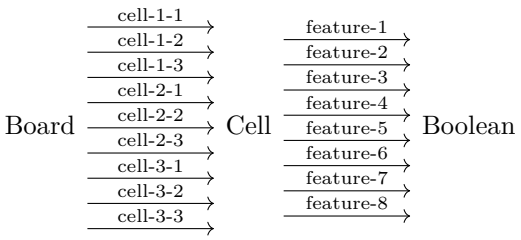


Figure 6.4: Diagram representing the schema for representations of content.

$V_2 : \mathcal{S}_2 \to \mathcal{R}_2$ is trained in unsupervised fashion on a dataset of just over 550,000 images of 24 by 24 pixels. The dataset was created by searching the Internet for images by keywords related to abstract graphical patterns (e.g. "geometric+pattern", "sharp+pattern", "line+pattern") and downloading the found images. From the 14,287 downloaded images random tile-sized sections were taken to produce a training set of 550,000 images.

The tool has 256 tiles because the agent can only perceive that many distinct tiles. The tiles in the tool are selected from the 550,000 images so that the agent can view them all as distinct. The image for a given tile id is selected by first representing the id as a base-2 number. The values (0 or 1) for each of the resulting 8 digits are then taken to be the values of the 8 boolean features. Given a particular tile id, all images from the data set that have feature values corresponding to the tile id are gathered and from them the image with the largest real-valued feature vector magnitude is chosen as the image for that tile id. This method tends to select images for which the feature values are seen as strongly false or strongly true.

By training $V_2 : \mathcal{S}_2 \to \mathcal{R}_2$ on numerous images we simulate the agent having "experienced" a great number of different patterns. It could not, however, differentiate between many of the tiles in the dataset and the tiles in the tool when it is reasoning. That is, the view compresses and hides information from the creative reasoning.

Concept In the SbCPF the highest representational category $\mathcal{R}_1$ exists to capture interpreted meaning (e.g. cultural and social meaning) that is not solely a function of the artefact at the content level. For our agent to have such a conceptual level it would need experience of things outside of just viewing images. It lies outside of the scope of this work to give

it such experience as our focus is on the structure of practice. However, we will simulate a conceptual experience by giving the agent a view $V_1 : \mathcal{S}_2 \to \mathcal{R}_1$ that maps the content image to the conceptual representational category. Where V_2 views the image per cell, V_1 looks at the image in its entirety and therefore has a conception of what the image looks like as a whole.

We define $V_1 = V_1' \circ V_2$. That is, V_2 is part of V_1. It takes the output of the content level view and uses that as input to V_1', which is a neural network whose architecture is shown in Figure 6.5. The input to V_1' is 9 cells of 8 real-valued features for each cell. The first layer is a convolutional layer of four kernels of size 1×1 with a stride of 1. It re-describes each cell from the input individually. Its output goes into the second convolutional layer which has 6 kernels with a size of 2×2 and a stride of 1. Each cell at the second layer describes a square of 4 cells from layer 1. Its output goes into the final convolutional layer which has 16 kernels with a size of 2×2 and is applied only once to the second layer output. The single cell at layer 3 describes a square of 4 cells from layer 2. Layer 2 describes overlapping corners of the original input, while layer 3 describes the entire input. Each cell at each of the three layers feeds into its own, equally sized, fully connected layer at the encoding level. That means there are 9 layers of size 4, 4 layers of size 6, and 1 layer of size 16. These layers are concatenated and rounded to form the boolean-valued output encoding. This way a conceptual representation can describe features of individual tiles, features of combinations of squares of 4 tiles, as well as features of the image as a whole. With this architecture V_1 maps to instances of the schema $\mathcal{C}_1$ depicted in Figure 6.6.

The encoding layer of V_1' is a variational autoencoder and the network is trained similarly to how V_2 is trained. The same image dataset of abstract patterns is used, but it is preprocessed before training begins. Several random sections of 72×72 pixels are taken from each image. These sections are divided into 9 tile-sized subsections, each of which is described with the already trained V_2. The resulting $3 \times 3 \times 8$ boolean matrix is saved to disk to be used as a training example for V_1'. In total there are 575,000 examples in the V_1' training set.

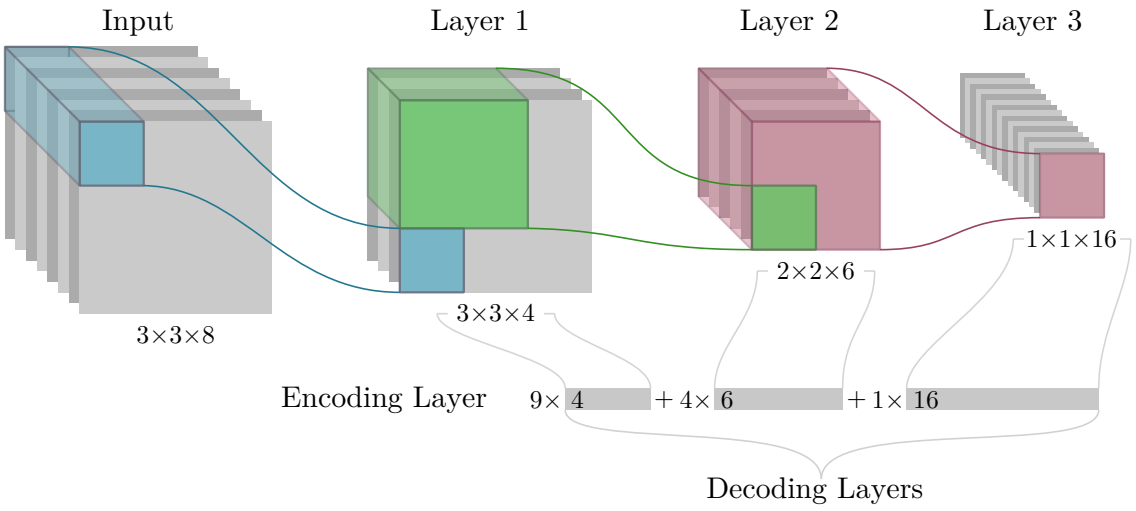


Figure 6.5: The neural network architecture of V_1'.

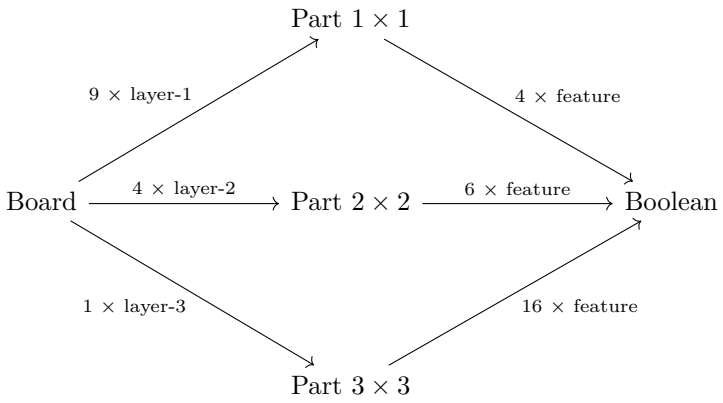


Figure 6.6: Diagram representing the schema for representations of concept.

6.2.3 Prediction functors

The prediction, generator, imagination, and policy functors form the core of this study. The view functors are, to a degree, arbitrary. That is, if they have been sensibly designed and produce representations that describe the tool states well, then the success of the study comes down to whether the set of C, G, T, and L functors can together form a well-behaving ensemble.

The prediction functors are to mimic the operation of the tool. They are in essence predictors of the external environment's behaviour. C_3 predicts how actions transform construct state and C_2 predicts what construct state looks like at the content level. C_1 maps content into the sphere of higher meaning.

We implement our prediction functor as simple fully connected neural networks with only a few layers. They take vector forms of the schema instances (see box 6.1) at one level as input and produce instances at another level as output. Each predictor functor is trained on a specifically generated dataset.

It is important to highlight that the prediction functors map between representational categories that allow for abstraction. They should therefore be capable of taking an incomplete representation and correctly map it. Specifically, it should maintain consistency of abstraction (see Section 5.3.3.1). The difficulty with this condition is that the agent cannot ever completely observe underspecified tool states and therefore cannot completely learn the map between representational categories from observation directly. For that reason we generate underspecified examples and include them in the data sets.

6.2.3.1 C_3, action to construct

$C_3 : \mathcal{R}_a \times \mathcal{R}_3 \to \mathcal{R}_3$ maps from a tuple of current construct state and an action to a new construct state. It is trained on a data set where each example is a tuple (x, y) of input x and output y. The input x is itself a tuple of a construct representation and an action representation, and y is the resulting construct representation of executing the action. The example in the data set are generated from sessions of random tool use. Each session starts with an empty board and 30 randomly generated actions performed in sequence. After each action the construct states before and after the action are viewed with V_3 and the action is viewed with V_a. The resulting $\mathcal{R}_3$ and $\mathcal{R}_a$ representations are subsequently serialised into vector form and written to disk as a single example. Each session thus produces 30 training examples. Empty cells in

$\mathcal{S}_3$ are viewed as undefined cells in $\mathcal{R}_3$. For that reason we do not have to explicitly generate underspecified representations for this data set. The dataset consists of one million training examples in total.

6.2.3.2 C_2, construct to content

The training examples for the C_2 data set are generated as follows. A random construct tool state $s_3 \in \mathcal{S}_3$ is generated by filling its cells with random tiles. A cell has a 15% chance of being left unfilled, resulting in the distribution in Figure 6.7. The corresponding content state $s_2 \in \mathcal{S}_2$ is computed with $R : \mathcal{S}_3 \to \mathcal{S}_2$. The states s_3 and s_2 are then viewed with V_3 and V_2 respectively, resulting in $r_3 = V_3(s_3)$ and $r_2 = V_2(s_2)$ whereby the unfilled cells are taken to be undefined. The representations are subsequently serialised and written two disk as a single example. The data set contains one million examples in total.

Number cells filled	%
0	0.0
1	0.0
2	0.0
3	0.1
4	0.5
5	2.8
6	10.7
7	26.0
8	36.8
9	23.2

Figure 6.7: The percentage of the data set made up by examples with n cells filled.

6.2.3.3 C_1, content to concept

The data set to train $C_1 : \mathcal{R}_2 \to \mathcal{R}_1$ is generated differently again. Two random construct tool states $s_3, s_3' \in \mathcal{S}_3$ are generated by filling their cells with random tiles in the same way as for the C_2 dataset. From s_3 and s_3' the corresponding content states s_2 and s_2' are computed with $R : \mathcal{S}_3 \to \mathcal{S}_2$. The construct states are discarded and the content states are viewed with the content and concept view functors V_2 and V_1 to produce representations $r_2 = V_2(s_2)$, $r_2' = V_2(s_2')$, $r_1 = V_1(s_2)$, $r_1' = V_1(s_2')$. Underspecified representations can then be generated by taking the join of r_2 and r_2' and the join of r_1 and r_1'. The tuple $(r_2 \sqcup r_2', r_1 \sqcup r_1')$ is then taken as an example, serialised into vector form and written to disk. The join of two representations from $\mathcal{R}_1$ represents the information that can be uniquely determined by the overlapping information in the two representations from $\mathcal{R}_2$. The data set contains one million examples in total.

6.2.4 Generator functors

Like the predictor functors the generator functors are also implemented as fully connected neural networks. G_3 and G_2 are trained on the same data sets as C_3 and C_2 respectively. The data set for C_1 is adapted slightly before being used to train G_1.

Before the C_1 data set can be used to train G_1 the examples have to be filtered. Because representations in $\mathcal{R}_1$ are compressed versions of $\mathcal{R}_2$ it is the case that different representations from $\mathcal{R}_2$ correspond to the same representation from $\mathcal{R}_1$. Although this is not a problem when training C_1, it will be a problem when training G_1, because a proper mathematical function cannot map a single $\mathcal{R}_1$ to multiple $\mathcal{R}_2$. It can, however, map to the join of those

$\mathcal{R}_2$ representations. To filter the data set it is first indexed by the $\mathcal{R}_1$ representations. Those that have multiple $\mathcal{R}_2$ representations linked to them are set to link to the join of those $\mathcal{R}_2$ representations instead. Once the filtering is completed all the examples are serialised to vector form and written to disk.

From the one million examples in the C_1 data set there were only 580 concept representations that had an average of 2.12 content representations linked to them. The low number of overlapping representations is to be expected given that the total size of $\mathcal{R}_2$ is 3^{72} (9 cells, each with 8 features, each of which can be true, false, or undefined) with only 2^{72} fully defined representations. We are therefore only covering a tiny proportion of the entire space with the data set. We will explore this further in the discussion.

6.2.5 Imagination functors

The imagination functors serve to take a generated seed representation and fill in details that are defined in the representation's schema. We have opted to use genetic algorithms (GAs) to implement the imagination functors. GAs belong to the class of evolution-inspired search algorithms (Goldberg 2006). GAs rely on a) random mutations of and mating between potential solutions to produce new solutions, and b) a fitness function that scores those potential solutions to guide the direction of the search. GAs are a good choice here because they are simple to implement, and they lend themselves well to problems where the solutions are of symbolic form, as are our representations.

6.2.5.1 Genetic algorithm

The GA operates on schema instances in triple form as described in box 6.1. It is the same for all levels of representations and as far as GAs go our implementation follows a fairly standard approach. It takes a seed representation, as provided by a generator functor, and instantiates a population of exact clones of the seed. It goes through several generations of mutation, mating, and (tournament) selection by fitness score. The best scoring individual from any of the generations is returned as the newly imagined representation. The pseudo code for the GA algorithm is shown in Algorithm 1.

The mutation and mating algorithms are implemented as recursive functions. The full Python source code can be found by referring to Section B.1.

6.2.5.2 Mutation

The MUTATE function takes a single schema instance I and a probability p between 0 and 1. It starts mutating at the specified root inhabitant. For each relation that the schema defines for the root inhabitant the mutation function assigns, with probability p, a new random value and thereby possibly overwrites the relation if it was defined in I already. If the relation was not redefined in this step then the MUTATE function checks if the relation is defined in I and, if so, recurses on the value for the relation. The recursion stops once there are no more defined values for any of the relations of the root value it is currently operating on or if there are simply no relations defined in the schema for it. The pseudo code for MUTATE is shown in Algorithm 2.

Algorithm 1 The imagination genetic algorithm

```
procedure GA(seed, goal)
    g ← 0
    pop ← MAKE-VECTOR(popsize, seed)
    best ← seed
    while g < generations do
        mutations ← EMPTY-VECTOR
        append seed to mutations
        for repr in pop do
            m ← MUTATE(repr)
            append m to mutations
        end for
        newPop ← []
        while length of newPop < popsize do
            parent1 ← SELECT(mutations, 3)
            parent2 ← SELECT(mutations, 3)
            offspring ← MATE(parent1, parent2)
            append offspring to newPop
            if SCORE(seed, offspring, goal) ≥ SCORE(seed, best, goal) then
                best ← offspring
            end if
        end while
        pop ← newPop
        g ← g + 1
    end while
    return best
end procedure

procedure SELECT(mutations, n)
    sub ← pick n elements from mutations
    ▷ Sort by SCORE in decreasing order
    sub ← SORT(sub)
    return sub[0]
end procedure
```

Algorithm 2 The mutation algorithm.

```
procedure MUTATE(I, root, p)
    relations ← the set of relations that I.schema defines for root
    for rel in relations do
        if random(0, 1) < p then
            ▷ root.rel indicates the rel-value I defines for root
            ▷ RANDOM(rel) generates a valid random value for rel
            root.rel ← RANDOM(rel)
        else
            if root.rel ≠ undefined then
                MUTATE(I, root.rel, p)
            end if
        end if
    end for
end procedure
```

6.2.5.3 Producing offspring

The MATE function works similarly. It takes two instances I and I' and produces a new instance I''. The root inhabitants of the two instances must be of the same type. First MATE assigns I'' a new instance with an empty root inhabitant of the same type as that of I and I'. For each of the relations specified in the schema for the root inhabitant MATE picks the root inhabitant of I or I' randomly and copies the value for that relation to the root inhabitant of I'', even if that value is undefined. If both I and I' actually have defined values then MATE recurses with those values as the new root inhabitants. This simple strategy is viable because none of the schemas contain any looping paths. The algorithm can be visualised as placing two trees of the same shape side by side. For each node in the tree shape pick either the value from the node in the left tree or the one from the right tree. If you pick an undefined node then all leafs below that should be undefined as well. The pseudo code for MATE is shown in Algorithm 3.

Algorithm 3 The mating algorithm.

```
procedure MATE(I, I')
    I'' ← empty schema instance
    MATE_h(I, I.root, I', I'.root, I'', I''.root'')
    return I''
end procedure

procedure MATE_h(I, root, I', root', I'', root'')
    ▷ PICK(r1, r2) chooses and returns one of its arguments
    root'' ← PICK(root, root')
    relations ← the set of relations that I.schema defines for root
    for rel in relations do
        ▷ root.rel indicates the rel-value I defines for root
        if root.rel ≠ undefined ∧ root'.rel ≠ undefined then
            root''.rel ← MATE^h(I, root.rel, I', root'.rel, I'', root''.rel)
        else
            root''.rel ← PICK(root.rel, root'.rel)
        end if
    end for
end procedure
```

6.2.5.4 Fitness function

As can be seen in Algorithm 1 the GA is guided by the scoring function when it selects the parents for the next generation's offspring. In GA terms, the scoring function tests the fitness of the individuals in a population. Individuals that score higher are more likely to be selected as parents and when they mate their offspring propagates their feature values. Over the course of several generations the GA finds better scoring representations and the best one will be returned as the solution.

The SbCPF suggests that the representations that should score highly are those that a) reduce entropy the most, b) increase the information content the most, c) stay within the bounds of imaginative power, d) are judged to achieve, or work towards, a higher level goal. The fitness function therefore has to incorporate all these requirements and produce a single numeric score with which different representations can be compared. We go through each requirement in turn.

Entropy and Information Content A probability distribution is needed to calculate entropy or information content of a representation (see Section 5.3.2) and so we will need such a distribution before the fitness function can use these measures. Luckily we can define a simple distribution because schema instances are defined as having a root inhabitant.

First we distinguish between complex and primitive types in a schema. Complex types are those that are the source of one or more relations. All other types are considered primitive. We define a probability distribution $P_{\mathcal{C}}(C)$ for every primitive type C in the schema $\mathcal{C}$

$$P_{\mathcal{C}}(o = C) = \begin{cases} \mid O_{\mathcal{C}}(C) \mid^{-1} & \text{if } o \text{ is a valid option} \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

where $O_{\mathcal{C}}(C)$ is the set of options the agent considers valid for the inhabitants of type C, and $\mid O_{\mathcal{C}}(C) \mid$ denotes the size of the set. This is simply a uniform distribution. For example, for the action schema $\mathcal{C}_a$ the Row type has the valid options $O_{\mathcal{C}_a}(\text{Row}) = \{1, 2, 3\}$ and $P_{\mathcal{C}_a}(\text{Row})$ assigns the probability of a third to each option.

Although $P_{\mathcal{C}}$ is defined for a particular type in the schema it is really used as the probability of a relation taking on a value. For example, consider the relation row : Remove $\rightarrow$ Row from schema $\mathcal{C}_a$. Given an inhabitant $rem \in$ Remove (for which we do not know the value for the row relation), the probability of the row value $P(\text{row}(rem)) = P_{\mathcal{C}_a}(Row)$.

With this we can define the probability distribution for complex types. In fact, it is defined in the same way as for primitive types, but the set of options for a complex types is defined differently. The set of options for a complex type is a function of the relations defined for it. That is,

$$O_{\mathcal{C}}(C) = \prod_{D}^{\mathcal{C}} \prod^{\text{Hom}_{\mathcal{C}}(C,D)} O_{\mathcal{C}}(D). \tag{6.2}$$

The identity of an inhabitant of a complex type is therefore completely defined by the values of its relations. This makes that we could define the probability distribution for complex types as

$$P_{\mathcal{C}}(o = C) = \prod_{D}^{\mathcal{C}} \prod_{r}^{\text{Hom}_{\mathcal{C}}(C,D)} P_{\mathcal{C}}(r(o) = D). \tag{6.3}$$

The entropy and information content of a schema instance are calculated as per the equations given in Section 5.3.2. As the probability distribution to be used we take the probability distribution for the type of the instance's root inhabitant as defined in this section.

Note that these recursive definitions work because all schemas are acyclic, meaning that no recursive loops occur in calculating the probability of a root inhabitant. Note also that by these definitions all distributions are uniform and all relations are independent. This makes that the increase in information content is in fact the same as the reduction in entropy and it is therefore that we only look at entropy reduction.

To calculate the entropy reduction $\Delta(I, I')$ from the seed I to the imagined representation I' we simply take the difference between their entropies

$$\Delta(I, I') = H(I) - H(I'). \tag{6.4}$$

The bounds of imaginative power If the GA was only guided by maximum entropy reduction it would, given enough generations, fill in all details in a single run. However, this would not take into account the concept of limited imaginative power. The fitness test should not just minimise the entropy of the representations it imagines, but it should do so while staying close to the limit l we set for it.

Instead of using the entropy reduction score $\Delta(I, I')$ directly we will use the limited entropy reduction score

$$\Delta_l(I, I') = l - \mid l - \Delta(I, I') \mid . \tag{6.5}$$

This allows the entropy reduction to be counted as normal as long as it is below the limit l. The entropy reduction score is reduced gradually if the entropy reduction grows above the limit.

Towards the goal All imagined representations are part of the hypothesis chain as described in Section 5.3.3. Any imagined representation should therefore be seen within the context of higher level goals, aside from those at the concept level. It does the agent no good to imagine ideas that might have high entropy reduction, but are not relevant to what they are trying to achieve. Relevance of imagined representations breaks down into two factors: compatibility and pertinence.

Compatibility refers to whether the agent believes the imagined representation yields or might lead to achieving the goal representation. Consider for example that the agent has a goal representation $r_2 \in \mathcal{R}_2$ and is scoring a newly imagined representation $r_3 \in \mathcal{R}_3$. If the agent thinks r_3 will directly yield r_2 or something more specific (i.e. $C_2(r_3) \leq r_2$), then r_3 should score very high. If $r_2 < C_2(r_3)$ then the agent thinks it can still reach an $r'_3 \in \mathcal{R}_3$ for which $r'_3 < r_3, C_2(r'_3) \leq r_2$. The imagined r_3 is thus promising, and we should still score it highly. If $C_2(r_3)$ and r_2 are compatible as defined in Section 5.4.2 (i.e. they have no contradictory information) then r_3 might also still lead to another highly scoring representation, albeit with non-pertinent details. In this case r_3 should be assigned some score. If none of the above is the case then r_3 should receive no score at all, because the agent does not believe it is relevant in achieving the higher level goal r_2. This can be encoded in a compatibility ranking function $\rho : \mathcal{R}_n \times \mathcal{R}_{n+1} \to \mathbb{N}$ defined as

$$\rho(r_n, r_{n+1}) = \begin{cases} 3 & \text{if } C_n(r_{n+1}) \leq r_n \\ 2 & \text{if } r_n < C_n(r_{n+1}) \\ 1 & \text{if } \exists r' \in \mathcal{R}_{n+1} \ . \ r' < r_{n+1} \ \wedge \ C_n(r') \leq r_n \\ 0 & \text{otherwise} \end{cases} \tag{6.6}$$

The second factor became clear from experiments with an early version of the agent implementation. The system had a preference for keeping busy imagining values for details that were not helpful to the higher level goals. For example, if a goal representation at the content level called for particular feature values for the tile in the second row and second column, then the artificial agent would imagine all kinds of constructs, but none of them with a tile in the second row and second column. That is, it was keeping busy, but only with finding solutions to problems it did not have. In other words, it was procrastinating. All the solutions it found were of rank 1 as determined by ρ. The imagined constructs were compatible with the goal representation, but only because it was avoiding anything relevant to the higher level goals.

To allow the agent to focus on the task at hand the fitness function includes a pertinence score component. The function $\pi : \mathcal{R}_{n-1} \times \mathcal{R}_n \to \mathbb{R}$ takes the imagined representation and the

higher level goal representation and returns a real-valued number indicating to what degree the two representations deal with the same issues. By including π in the fitness function the GA more quickly finds pertinent solutions.

In this case, the pertinence score is high when two representations are concerned with the same cells, and low when they are concerned with different cells entirely.

Fitness function Having introduced the different components of the fitness function its full definition can be given. Algorithm 4 shows how the individual score components are combined into a single fitness score.

The biggest component of the final score is the multiplication of the limited entropy reduction score, the ranking score, and the pertinence score. This is scaled by 10^r to exagerrate the importance of the ranking. If the rank is 0, then the whole component is zero.

The pertinence score is added to the score separately as well. The effect of this is that if the rank is zero those solutions that at least address the right area of the board are preferred. The next population generation then has the chance to produce solutions with higher ranks.

Algorithm 4 The scoring algorithm.

```
procedure SCORE(seed, imagined, goal)
    p ← π(imagined, seed)
    e ← Δ_l(seed, imagined)
    r ← ρ(goal, imagined)
    score ← p + r * 10^r * e * p
    return score
end procedure
```

6.2.5.5 Act functor

The agent's representation of action are translated directly to a command invocation. It is a simple translation and is therefore implemented manually. The tool uses default values if the agent fails to specify a value for any of a command invocation's properties.

6.2.6 Policy functor

An agent's policy functor $L_t : \mathbf{Agent}_t \times \mathbf{State}_t \rightarrow \mathbf{Agent}_t$ determines what action the agent takes when faced with a certain tool state. No actual definition of the policy functor was given in Chapter 5. A particular implementation of a policy should define when and how an agent employs its views, generators, predictors, and other functors to come to a decision about an action to perform and possibly how to update its functors. In this section we will define the policy used for this study.

Although the SbCPF does not give a definition of a policy, it does suggests a certain structure by defining the functors it can make use of and by stating an agent works iteratively from an abstract goal to a concrete artefact, driven by entropy reduction. The policy for this study was manually designed to adhere strictly to what the SbCPF suggests while keeping the complexity as low as possible.

As stated the purpose of a policy is to take as input the current state of the agent (i.e. its currently held goals and functors) and tool state and decide what action to take in the tool. Our policy determines actions, but it does not update the agent's functors along the way and so the agent does not learn. It breaks down into the two main recursive functions ACT

and IMAGINE, shown in Algorithm 5. With these algorithms the definition of our agent is complete and simulations can be run.

The simulation world consists of only the agent and the tool. The agent views the current tool states with the view functors. The resulting internal representations are accessible to the agent and are referred to in Algorithm 5 as $state_\ell$, where ℓ refers to the concept, content, construct, or action level. The agent also has internal goal representations, referred to as $goal_\ell$. The agent starts a session of tool use with some concept goal $start_1 \in \mathcal{R}_1$. The simulation then asks the agent to determine the next action to take by invoking ACT($step, \ell, start_1$), where *step* starts at 1 and is increased by 1 for every time ACT is invoked, and ℓ is "concept".

The policy works in top-down fashion in that it will first checks its progress with regard to its concept goal. Only in light of that progress will it decide what to do at the content level. Similarly, it will only consider what needs to be done at the construct level in light of its progress with regard to the content goal, and so on and so forth. The bulk of the logic behind this behaviour is defined in lines 7 to 23 in Algorithm 5.

At line 7 the agent check whether it has achieved the concept goal as ℓ is initially set to "concept". If so it will adopt information from the viewed tool state into the concept goal as well as remember the current step as a checkpoint for the current level: a moment in time when the tool state matched the concept goal. It will then invoke the IMAGINE function (we will come back to this shortly).

If however the concept goal has not yet been achieved, and the agent thinks it might achieve it yet, then ACT will simply recurse to make the decision at the content level (line 13). The agent simply suspends judging the concept state of the artefact until it either definitely fits with the content goal or definitely does not. The agent will need to take further action before it can make this judgement and so it recurses and attempts to make the decision at the content level. If it has to suspend judgement there also then it will recurse to the construct level and so on.

If the current state cannot lead to the goal then the agent has taken a wrong turn and it will need to backtrack (line 14). There is however a limit to how often the agent can attempt to retry a goal at a certain level. If the number of *retries* at the current level has not yet exceded the limit then the agent will re-imagine from the current level down. That means backtracking to the last known checkpoint for the level at hand and invoking IMAGINE from the next level down. If the retry-limit was exceeded then the agent has to forsake the current level's goal and retry one level up, meaning that it will replace the current level's goal with another thereby effectively adopting a different hypothesis. Of course the agent cannot retry at a level above the concept level. If it finds it needs too many tries to achieve a concept goal it will need to either imagine a new concept goal or give up entirely.

This particular policy can be described as "greedy". That is, it is eager to adopt all information from the current state, even if it was the result of a wrong prediction. The agent is also "perfectionist" as it does not allow any margin for error in judging state against the goals it holds.

The second part of the algorithm is defined by the IMAGINE function (line 28). Its job is to employ the genetic algorithm to imagine new details for its goals, in service of higher level goals the agent might have. IMAGINE starts imagining at the level as requested by ACT. It has to act slightly differently in case the agent is retrying, but the main logic is the same. Firstly, IMAGINE determines what it should use as the seed for the GA. It then takes the GA's output and adopts that as the new goal for the current level. Then, if it has not yet reached the action level, it generates the goal state for the level below with the G functor and recurses.

Once the policy has produced a newly imagined goal at the action level the simulation will take that as the action the agent has decided upon and will use the agents perform functor to produce a tool command invocation. The simulation then executes the tool action, updates the tool states, and continues with the next simulation step.

6.2.7 Evaluation

There are two separate evaluations that need to be performed. Firstly, the agent implemented as described above is to be evaluated itself. Secondly, the agent serves as an evaluation of the SbCPF as described in Chapter 5.

6.2.7.1 Evaluating the agent

In order to evaluate the agent we will look at how it functions. It is not a given that a system that follows the SbCPF architecture will lead to a valid software program. Therefore we will need to see if the system in fact functions at all, and if it is capable of using the tool to produce artefacts.

The agent should also be evaluated in terms of how well it functions. That is, given a seed goal at the conceptual level, what is the probability that it achieves that goal? Also, how often does it make a hypothesis error requiring it to retry? At what level do most hypothesis errors occur?

6.2.7.2 Evaluating the SbCPF

The purpose of the agent implementation was to create an instance of the SbCPF and thereby evaluate it. We should first evaluate whether the agent is in fact a valid instance. That is, does the agent's architecture follow the SbCPF closely? If it deviates from the SbCPF architecture, what is the reason and what does that entail?

Finally, even though the agent works with a software tool of very low complexity, can we see agent behaviour that reminds us of the tutors' practice as described by the grounded theory in Chapter 4? If we see contradictory behaviour then that is evidence that either the implementation or the SbCPF has missed the mark. If the behaviour is similar then that is some evidence of the validity of the SbCPF. It will however be difficult to compare behaviour between these two very different domains and levels of complexity.

6.3 Results

This section first presents the parameters and training results for each of the agent's component functors. It then presents the results pertaining to the functioning of the system as a whole. Because our focus is strongly on the latter, we have not attempted to maximise the performance of each of the component functors. Instead, the aim was to find a good compromise between accuracy and simplicity.

6.3.1 Views

The hyperparameters of the convolutional neural networks $V_1 : \mathcal{S}_2 \to \mathcal{R}_1$ and $V_2 : \mathcal{S}_2 \to \mathcal{R}_2$ were chosen so as to provide the lowest possible complexity while maintaining acceptable performance.

Algorithm 5 The algorithm that determines which action to perform at each step.

```
1: procedure ACT(step, ℓ, start_1)
2:     if step = 1 then                                   ▷ If this is the first step...
3:         goal_1 ← start_1                               ▷ then start from the given seed concept.
4:         IMAGINE(ℓ, false)
5:     else if ℓ = action then                            ▷ If we have recursed to the action level...
6:         IMAGINE(action, true)                          ▷ then map down imagine a new action.
7:     else if state_ℓ ≤ goal_ℓ then                      ▷ If we have achieved the (partial) goal...
8:         goal_ℓ ← M(goal_ℓ, state_ℓ)                    ▷ adopt information from viewed tool state,
9:         SAVE-CHECKPOINT(ℓ, step)                       ▷ remember the current state and goals,
10:        RESET-RETRY(ℓ)                                 ▷ reset retry counts from this level down,
11:        IMAGINE(ℓ, false)                              ▷ and continue imagining new details.
12:    else if compatible(goal_ℓ, state_ℓ) then           ▷ If we could still reach the goal...
13:        ACT(step, ℓ + 1, start_1)                      ▷ recurse to take the decision one level down.
14:    else                                               ▷ If the current state cannot lead to the goal...
15:        if retries_ℓ > retry-limit then                ▷ If exceeded retries limit...
16:            if ℓ = concept then                        ▷ If stuck at the highest level...
17:                start over                             ▷ give up and start fresh.
18:            else                                       ▷ If stuck at one of the other levels...
19:                RESET-RETRY(ℓ)                         ▷ reset retry counts from this level down,
20:                REIMAGINE(ℓ − 1)                       ▷ give up and retry one level above.
21:            end if
22:        else                                           ▷ If allowed to retry...
23:            REIMAGINE(ℓ)                               ▷ try this level again.
24:        end if
25:    end if
26: end procedure
27:
28: procedure IMAGINE(ℓ, retry)
29:    if retry ∧ ℓ > concept then                        ▷ If retrying...
30:        seed ← state_ℓ                                 ▷ start from restored state.
31:    else                                               ▷ If not retrying...
32:        seed ← goal_ℓ                                  ▷ continue from current goal.
33:    end if
34:    goal_ℓ ← GA(seed, goal_{ℓ−1})                      ▷ Imagine new details for the goal.
35:    if ℓ < action then
36:        goal_{ℓ+1} ← G_ℓ(goal_ℓ)                       ▷ initialise lower level goal.
37:        IMAGINE(ℓ + 1, false)
38:    end if
39: end procedure
40:
41: procedure REIMAGINE(ℓ)
42:    LOAD-CHECKPOINT(ℓ)                                 ▷ Restore last valid state remembered for level.
43:    retries_ℓ ← retries_ℓ + 1                          ▷ Increase retry count for level.
44:    IMAGINE(ℓ + 1, true)                               ▷ Re-imagine from one level down.
45: end procedure
```

6.3.1.1 Content view, V_2

The V_2 view reached an accuracy score of 0.91 after 240,000 batches of 100 examples per batch. The accuracy was calculated by taking a test set x of 10,000 examples, computing the autoencoder output y, and quantifying the autoencoding error as

$$\text{accuracy}(x, y) = 1 - \text{avg}(\mid x - y \mid)$$

where avg takes the geometric mean over all examples and all dimensions. The convolutional encoding layers had 8 kernels each, as did the decoding layers. Further training, up to 100 epochs, did not result in a higher accuracy. Using 16 kernels for encoding and decoding did not improve accuracy, although using 4 kernels significantly reduced it.

To check that the resulting encodings are informative we use V_2 to calculate the sigmoid of the real-valued encodings for all examples in the training set. The histograms for each feature in the encoding are plotted in Figure 6.8. They show that for every feature there is a sufficiently even distribution of examples around the value 0.5. This is important because it shows that the features remain informative when rounding the real-valued feature values to boolean values.

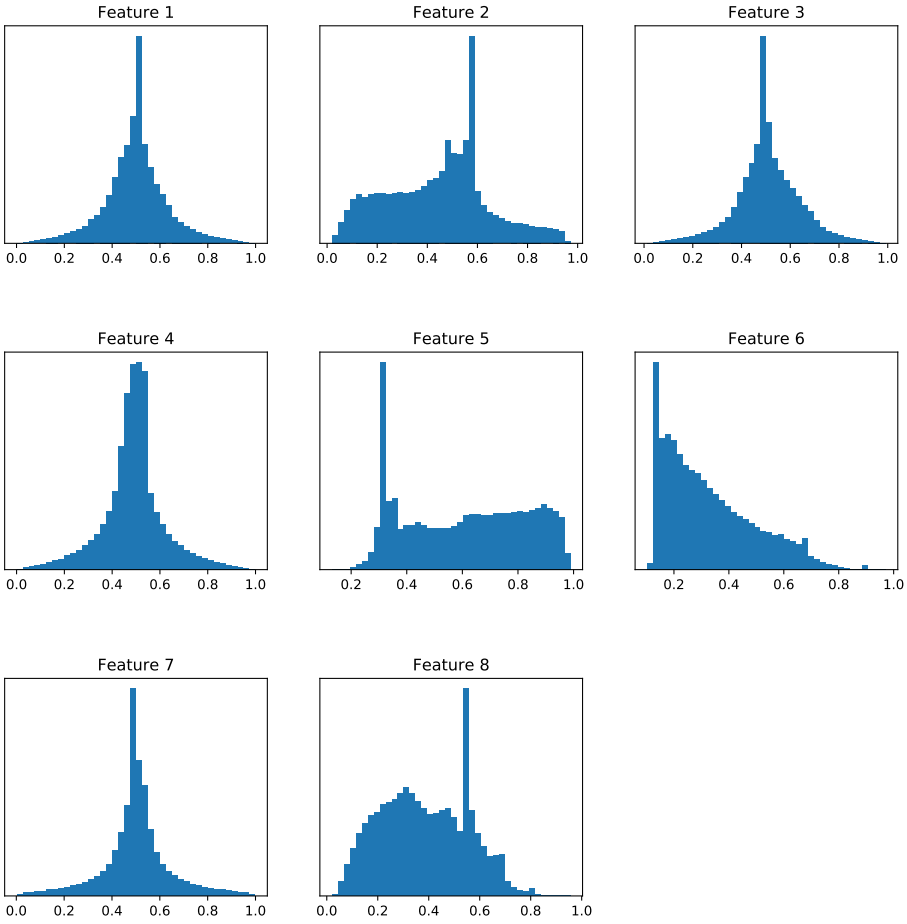


Figure 6.8: The distribution of values for each feature as calculated by V_2 for all examples in the data set. The horizontal axes indicate the sigmoid of the feature value and the vertical bars indicate the number of examples per feature value.

The pairwise correlations between two features can give an indication of the amount of overlap in information encoded by those features. Figure 6.9 shows the Pearson correlation matrix from which it can be seen that there are generally no strong correlations between features. This indicates that the variational autoencoder was trained successfully.

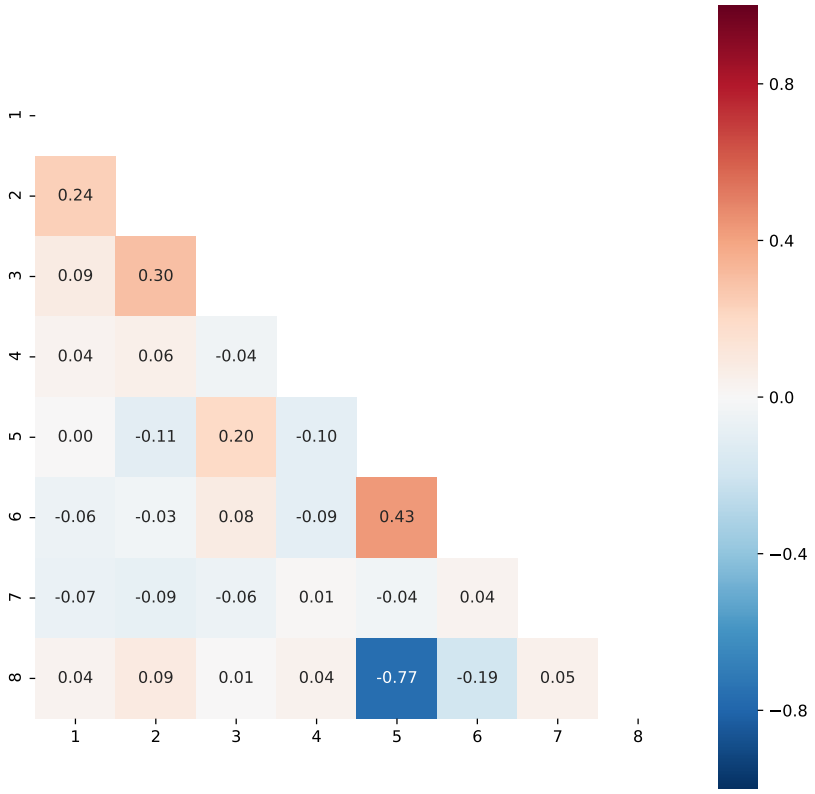


Figure 6.9: The pairwise correlations for all features as calculated by V_2.

The V_2 decoding layers can be used to give an intuition for how much information is retained after a tile has been encoded. Figure 6.10 shows the first 128 tool tiles as well as their encoded-then-decoded versions. These are generated by first encoding each tile as the 8-dimensional real-valued vector and immediately decoding. As can be seen the generated versions are very different, but this is to be expected when compressing by a factor of $\frac{24\times24\times4}{8} = 288$ over a dataset of varied abstract patterns. Also it is worth pointing out that the point of this exercise is not to construct a highly performant autoencoder, but to give the agent a view that produces a chosen number of feature values and encodes as best it can.

6.3.1.2 Concept view, V_1

The V_1 view reached an accuracy score of 0.99 after 160,000 batches of 100 examples per batch. The three convolutional encoding layers had 4, 6, and 16 kernels respectively, resulting in an encoding of size $3 \times 3 \times 4 + 2 \times 2 \times 6 + 16 = 76$. The decoding layers are two fully connected layers of size 76 and 72. The encoding size of V_1 is larger than that of V_2 and it might appear therefore that there is no compression. However, it should be noted that the bottleneck lies at the first convolutional layer of V_1 of size $3 \times 3 \times 4 = 36$ and so the compression is significant. Further training, up to 100 epochs, did not result in a higher accuracy.

The sigmoid of the feature value distributions are shown in Figure 6.11. Most features are shown to be nicely centered around 0.5, although there are a few pathological cases.

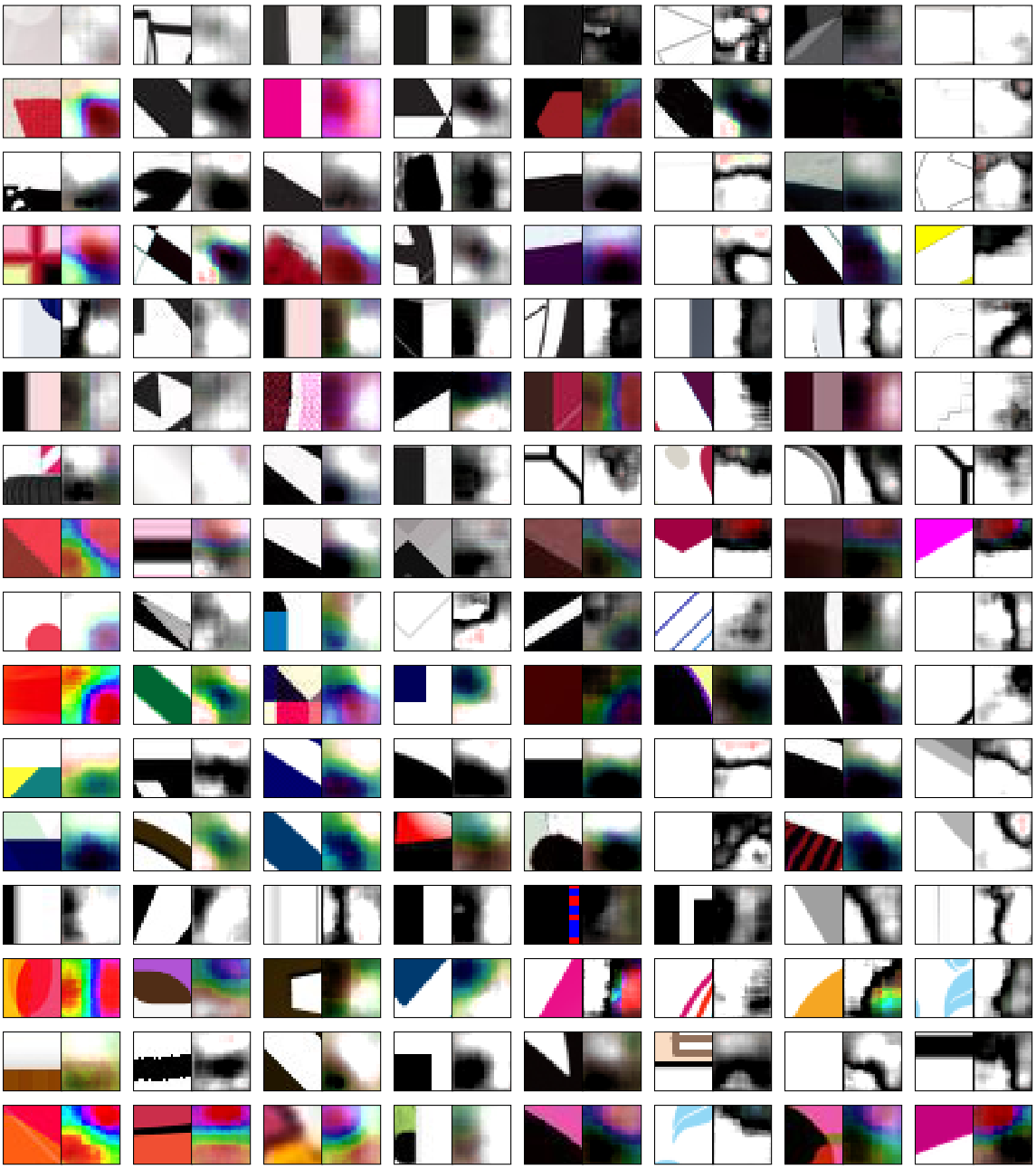


Figure 6.10: Shown are the first 128 of the 256 tiles available in the tool, alongside their encoded-then-decoded versions as produced by V_2.

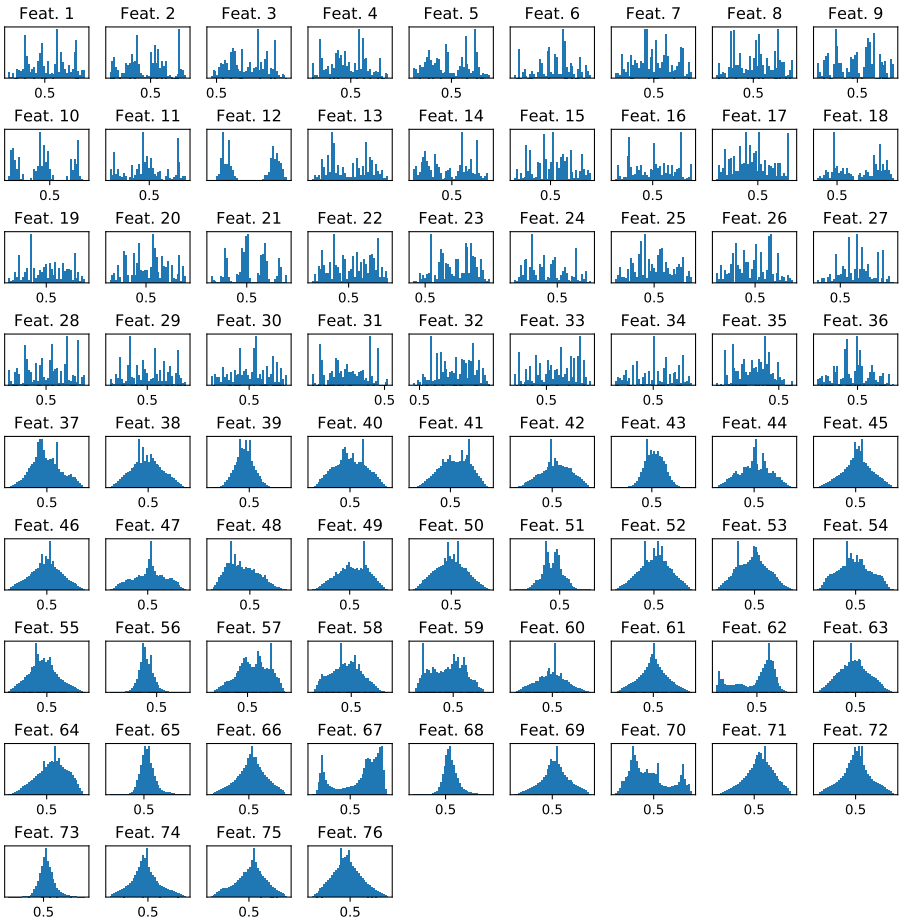


Figure 6.11: The distribution of values for each feature as calculated by V_1 for all examples in the data set. The horizontal axes indicate the feature value and the vertical bars indicate the number of examples per feature value.

The correlations for V_1's features are plotted in Figure 6.12. As was the case for V_2 the correlations are weak for V_1 also, indicating successful training.

Autoencoders can be used to generate images by taking random values as an encoding and then decoding those values into an image. This can give some insight into the functioning of the autoencoder. In our case we can generate a random concept representation, generate a content representation from it with V_1's decoder, and then apply V_2's decoder 9 times to produce a 72×72 pixel image. In that way we obtain an indirect view of what the agent is "thinking". This only works for fully specified representations however, and it does not serve any other purpose for this study than to verify that the two autoencoders are not obviously broken. Figure 6.13 shows 16 images generated from random concept representations. It is worth noting that the nine individual cells can clearly be seen. This is because the V_2 decoder runs separately for each cell. However, the entire board has a certain consistency across all cells due to the concept features that describe combinations of tiles at the second and third convolutional layers of V_1.

6.3.2 Imagination

The main parameters that have to be set for each of the imagination functors are the parameters for the genetic algorithm and the imaginative power.

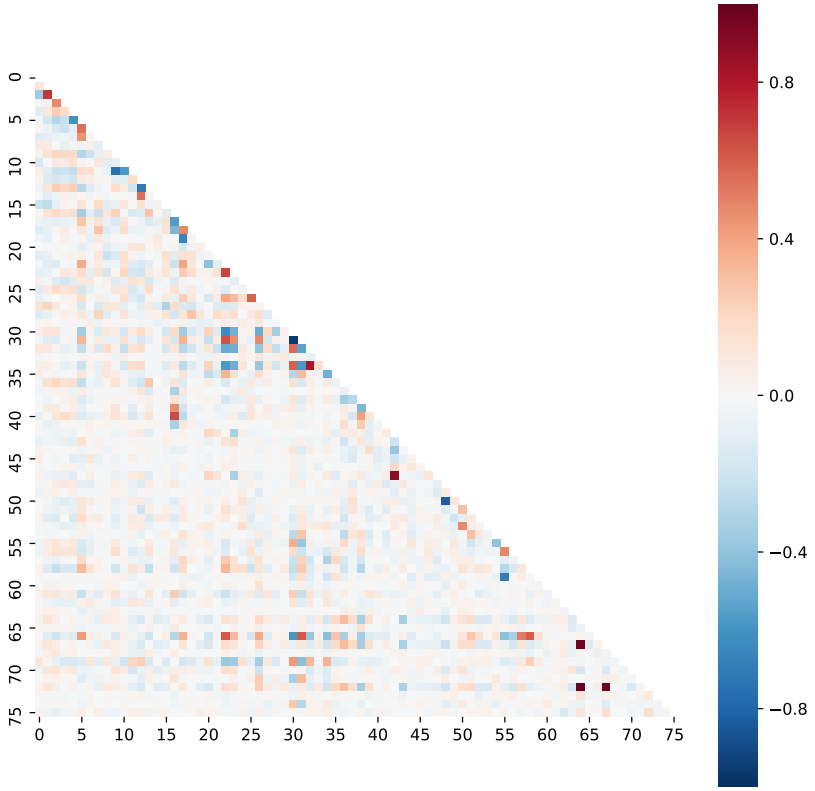


Figure 6.12: The pairwise correlations for all features as calculated by V_1.

The imaginative power is set differently at each level of imagination. At the concept and content level the imaginative power is 2 bits. The agent therefore tends to define two extra boolean feature values per imagination step. The imaginative power at the construct level is set to 8 bits. At the construct level a single choice of tile corresponds to 8 bits entropy reduction, simply because there are 2^8 tiles to choose from. If the imaginative power was lower, then that would mean the agent would not be able to make any decisions at the construct level. The imaginative power at the action level is set so that the agent can imagine a fully defined action, meaning a choice of row, column, and tile id.

Early experiments indicated that if the imaginative power at the concept and content levels were much higher that the agent had trouble making progress. That is, high level goals with too much detail at once required the agent to make too many correct hypotheses in a row to achieve those goals. By keeping the imaginative power low the agent can make progress. This does however indicate a limit in predictive accuracy, which we will discuss shortly.

The parameters for the genetic algorithms were chosen to give the agent just enough time to imagine high scoring representations without wasting time generating too many. At the concept level the GA runs for 15 generations with a population size of 250 and a mutation probability of 25%. At the content level the GA requires 25 generations with a population of 500 and the same mutation probabity. The content level GA needs more computational cycles, because the representations it generates are scored against the concept goal, while at the concept level there is no such constraint and imagination is free. The construct level GA runs for 20 generations with a population size of 1000 and the same mutation probability. The construct level GA has a larger population size because that gives it the ability to cover a wider range of tile ids. The action level GA runs for 15 generations with a population of 50 and a mutation probability of

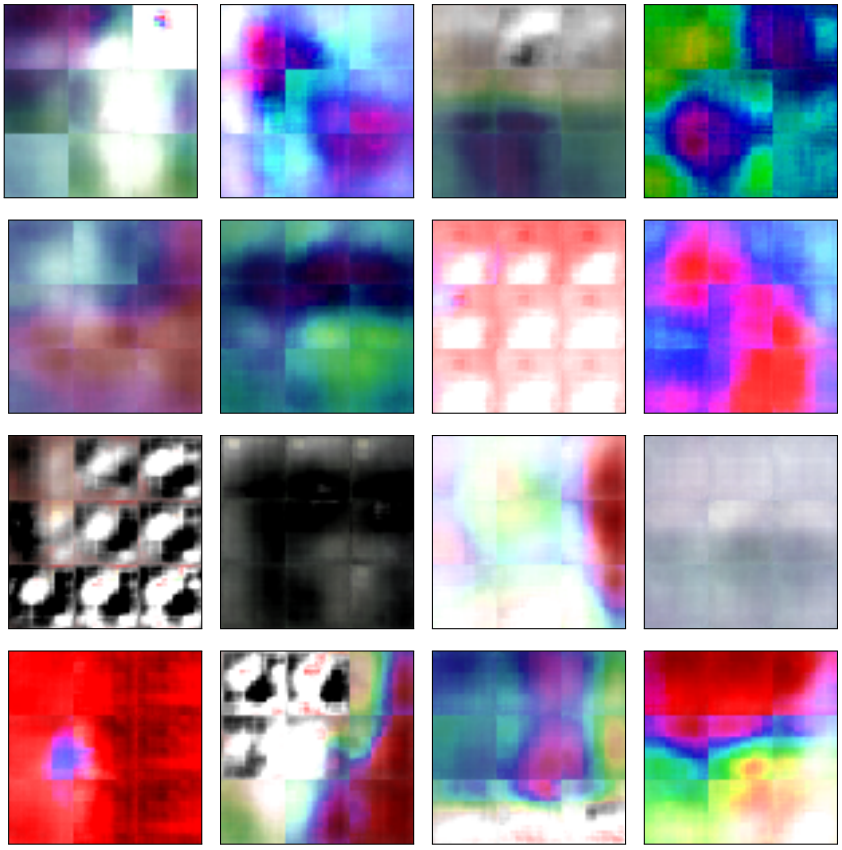


Figure 6.13: Images generated from random concept representations.

0.5. The G_3 functor typically finds the right choice of tile and cell and so the action GA only comes into play when G_3 has failed to come up with the right choice. In that case the action GA simply needs to try different tile ids and check them with C_3.

6.3.3 Mapping up and down

The GAs follow a fairly standard design, and have been intentionally kept simple. They randomly generate options and rely mostly on the prediction functors to guide them in the right direction. The prediction and generator functors are in a real sense the core of the system and it is their accuracy that determines the performance of the agent.

There are in total three generator functors and three predictor functors. They are all implemented as fully connected neural networks with two hidden layers, albeit with different sizes for the input, hidden, and output layers. The only exception is C_3 which has only one hidden layer. The hyperparameters for the neural networks were chosen to minimise the sizes of the hidden layers while maintaining good accuracy scores. All networks were trained with a batch size of 200 and were optimised with the Adam algorithm (Kingma and Ba 2014) and a learning rate of 0.001.

Table 6.1 show the parameters and accuracy scores for each of the six functors. The accuracies are determined on separate test sets of examples that are not seen in training. They are created in the same way as the training sets and contain 100,000 examples each. The accuracy score is calculated by considering each feature of the schema individually rather than at the individual neurons. For example, a tile id feature is encoded as a one-hot vector of length 256. If the network predicts a wrong tile id then we count that as a single mistake

Functor	Input	Hidden 1	Hidden 2	Output	Accuracy
G_1	224	228	228	173	82.8%
C_1	173	228	228	224	96.0%
G_2	173	144	144	2571	99.8%
C_2	2571	144	144	173	100.0%
G_3	5142	270	270	543	97.4%
C_3	3114	1000	n.a.	2571	98.2%

Table 6.1: The parameters and accuracy scores for the predictor and generator functors. The parameters are the input, hidden, and output layer sizes of the neural network.

rather than two or more wrongly predicted individual neuron values. To calculate the accuracy score we simply take the number of correctly predicted features and divide by the total number of features defined by the schema. Wrongly predicting whether a feature is defined or not also counts as a mistake. The accuracy score can be interpreted as the average percentage of features accurately predicted. As can be seen in the table the notable outlier is G_1, which has a significantly lower accuracy score than any of the other functors.

It should be noted that the large input and output layer sizes for C_2, G_3, and C_3 are due to the one-hot encodings of the tile identifiers.

6.3.4 Simulations

An agent starts every simulation run with a blank slate, and, therefore, with a completely undefined concept goal. From there, the agent imagines the first few bits of concept, and works towards a fully defined artefact. At each step in the simulation the agent has to determine an action. A simulation run is finished when the agent finishes an entire artefact or after 100 steps have been taken, whichever occurs first. We have run a total of 182 simulation runs consisting of a total of 16,220 actions taken. In 30 sessions the agent completes the board within 100 steps.

For each step of the simulation the agent will have formulated a hypothesis that ultimately resulted in the adoption of an action to take for that step. Figure 6.14 shows how often and at which levels the agent succeeds or fails to make the right hypothesis. The agent's policy is a greedy one and makes judgements with regard to its hypotheses from the top down. That means that it considers at the content level only those decisions that were deferred from the concept level, and so on and so forth. Only a small part of the decisions are made at the concept level. That means a concept goal typically requires multiple actions to achieve and so the agent will continue at the lower levels until a judgement can be made at the higher levels. The action level is not included here because the agent does not judge actions.

Table 6.2 show the numbers that are depicted in Figure 6.14. Hypotheses that are judged at the concept level are mostly rejected. As we move to the lower level the hypotheses are more frequently accepted and at the construct level the vast majority of hypotheses turn out to be correct.

The numbers in Table 6.2 include every retry as a failed hypothesis. If instead we consider how many goals are achieved, regardless of the number of attempts the agent needs, then we can see in Table 6.3 that the numbers improve. That means the randomness in the imagination leads to the agent trying a different approach. Retrying is therefore a valid way of improving

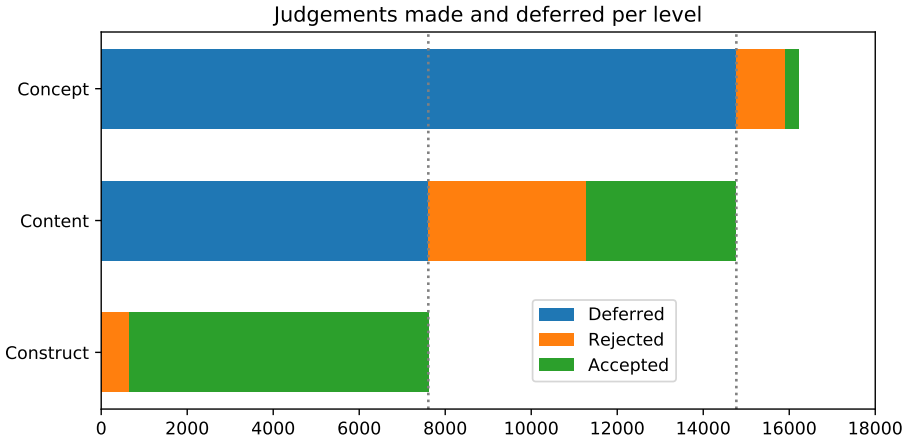


Figure 6.14: The bars indicate the number of judgements that are made at each level or deferred to the level below.

Level	Deferred (%)	Rejected (%)	Accepted (%)	Total
Concept	91.1	7.0	1.9	16220
Content	51.5	24.9	23.6	14770
Construct	n.a.	91.6	8.4	7607

Table 6.2: The exact number of decisions and the percentage of deferred, rejected, and accepted hypotheses per level.

performance. The trend is the same however; the agent finds higher level goals harder to achieve.

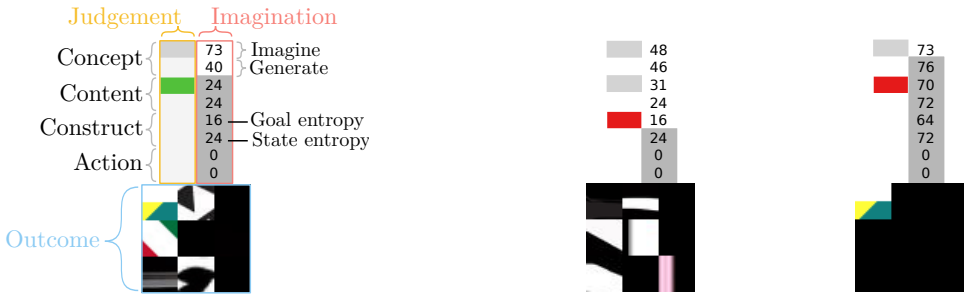
Level	Achieved (%)	Given up (%)	Total
Concept	51.2	48.8	602
Content	81.7	18.3	4267
Construct	97.9	2.1	7119

Table 6.3: The number of goals attempted at each level and the percentage that was achieved or not.

The behaviour of the agent was inspected and debugged with the help of a visualisation tool that loads a session log and displays each step of the agent's decision process and progress towards its goal.

Figure 6.15a breaks down the components of a step visualisation. The left vertical bar indicates how the outcome of the previous step is judged. A grey block means that the decision is deferred to the lower levels. Green or red blocks mean that the state was accepted or rejected respectively. In Figure 6.15a the decision was deferred to the content level, where the content state was accepted as matching the goal. In Figure 6.15b the judgement was deferred down to the construct level, where the construct state was rejected, and in Figure 6.15c the decision was made at the content level, where the content state was rejected.

Once a judgement has been made, the agent employs its imagination to think of new goals. The vertical bar labelled "Imagination" in Figure 6.15a visualises at what level the imagination starts. Each level is divided into two blocks, one for the invocation of the imagination functor and one for the *generate* functor. If the background of a block is grey then the corresponding functor was invoked, whereas white indicates it was not. For example, in Figure 6.15a, after accepting the current content state, the agent imagines new details at the content level and



(a) (b) (c)

Figure 6.15: Each subfigure shows the visualisation of a reasoning step.

maps down the newly imagined goal to the construct level with the *generate* functor. This is repeated for the construct and action levels. However, in Figure 6.15b the agent has rejected the outcome at the construct level. Instead of imagining a new construct goal the agent retries and generates an action goal seed as the first step. Figure 6.15c is an example of when the agent has exceeded the retry-limit. The current content state is rejected, and the agent retries from the concept level instead of generating a new construct goal seed and retrying from the content level. That is, it takes the current concept goal and generates a new seed content representation for content-level imagination.

Overlaid on the vertical imagination bar is the entropy of the goal representation and the current state at each of the levels. These are calculated after the imagination step and before executing the imagined action.

The state of the artefact, as rendered by the tool, is shown below each step. This is the state of the board after the imagined action has been executed and is the state that is judged in the next step.

Figure 6.16 depicts the first 78 steps of one of the simulation sessions. Each step shows how the agent judges the outcome of the previous hypothesis, the levels involved in imagining of the new hypothesis given the judgement, and the outcome of the new hypothesis. Figure 6.16 also shows, as line graphs, the development of the entropy of the imagined, adopted, and actual states.

The line graphs in Figure 6.16 give an overview of how the artefact progressed over the course of the session. The “state” line indicates the expectation of how much information is yet to be defined before reaching a fully defined artefact. The closer that line is to zero, the more the board is filled. The “imagined” line shows the measure of how much information is left undefined in the goal representations. The “adopted” line shows the same, but for the last checkpoint. The moments of change in the “adopted” graph correspond to adopting or resetting to a checkpoint state. Figure 6.16 shows that the agent filled the board completely three times, but that the board was rejected at the concept level each time. The three blue troughs in the concept level graph correspond to those three attempts where the agent had to defer judgement until the board was completely filled.

Although hard to read at first, the visualisation tool proved invaluable in inspecting the agent’s policy behaviour. It brought to our attention one particular pathological behaviour. The agent can get into an infinite loop when it imagines a new goal construct that is the same as the current construct state and then subsequently imagines it should remove a tile from an empty cell. This then results in an unchanged construct state. The agent then repeats this

process and is thus stuck in a loop. The deletion action is a reasonable action, as it results in exactly the construct state that the agent has assumed as its construct goal. The problem must therefore be that the agent cannot imagine any moves that it thinks will yield the content goal. Emptying an already empty cell allows it to effectively do nothing and stay where it is. This behaviour also occurs in Figure 6.16. It starts in the last 8 visible steps and continues until the simulation reaches the step limit.

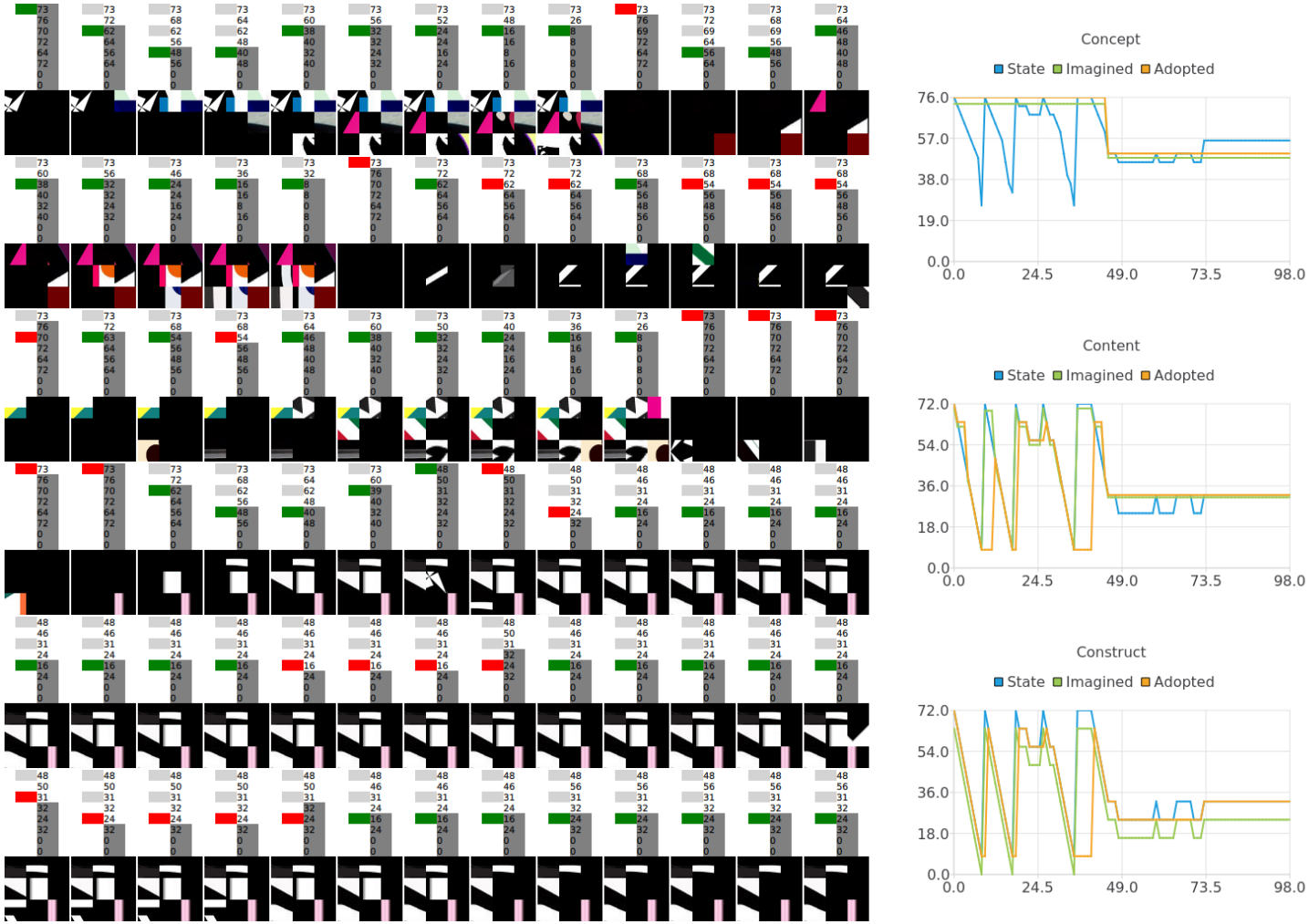


Figure 6.16: This visualisation of a session summarises the decisions made by the agent and how it progressed or failed with regard to its goals. The steps are ordered left to right, top to bottom.

6.4 Discussion

In this chapter, an implementation of the SbCPF was introduced. This model has a simple implementation for every component of the framework, and is therefore a model of an agent that engages in software-based creative practice in a manner similar to that of a human practitioner. The agent interacts with a software tool and through that interaction changes the software's document state, thereby creating a digital artefact. The agent can view the tool's document directly or view it in its rendered form. As a result of viewing the tool state, the agent produces internal representations. It is able to compare these against its goal representations and derive the next course of action by employing its imagination. The agent is able to work from partial concepts to concrete artefacts, adopting information from the environment along the way.

More specifically, a simple agent was implemented that fills a 3×3 board with tiles. The agent's architecture follows the SbCPF to the letter. It has an implementation for each of the SbCPF's functors. Furthermore, it has a relatively simple policy functor that orchestrates the use of the other functors in order to decide which actions to take. The agent was used in simulation runs and its behaviour quantified.

Firstly, the agent is capable of making decisions and its architecture follows the SbCPF strictly. This shows that the SbCPF is intrinsically consistent. It also shows that the SbCPF can be taken as a software architecture for a practically creative agent without needing to specify large additional components or adapting the model significantly. It is worth pointing out also that in the agent's implementation only the view and perform functors have knowledge of the tool. All other functors are agnostic to the tool used in this study and could presumably be reused with another tool. This indicates that the SbCPF is indeed domain agnostic.

The agent's policy was kept as simple as possible and it did not require much additional logic to be implemented. This shows that the other functors do the heavy lifting and that the policy only needs to decide when those other functors are invoked.

The performance of the agent in the simulations highlights a few important issues. Table 6.2 and Table 6.3 clearly indicate that the agent judges hypotheses as incorrect much more frequently at the higher levels than at the lower levels. This can be explained by the fact that prediction and generation errors are compounded. That is, for a concept level goal to be achieved the whole hypothesis chain must be correct, and possibly over multiple actions. On the other hand, only one prediction has to be correct to achieve a construct goal. Another factor is that the agent can hold goals at the concept level that are internally inconsistent. Because the agent is free in imagining its concept goals, it can imagine any combination of feature values. However, the 2×2 and 3×3 feature values are entirely determined by the 1×1 feature values. As such, it could imagine values for its 1×1 features that are inconsistent with the 2×2 values it imagines. Future studies or implementations should take this into account. If deep neural nets are used then their layers should either explicitly be part of the hypothesis chain or, alternatively, only their output layers should be.

Also, the accuracies in Table 6.1 can be slightly misleading and it is worth taking note that they do not indicate the percentage of times the predictions or seed generations are correct. They are a measure of the average error in a single prediction. In that respect these levels of accuracies can easily lead to incorrect predictions and therefore incorrect hypotheses.

The outlier in the accuracy scores in Table 6.1 is the low accuracy of G_1. This can be explained by two factors. Firstly, the heavy compression of information that occurs between the concept and content levels: V_1 compresses V_2's output by 50% in the first convolutional layer. It is obviously difficult to reconstruct an $\mathcal{R}_2$ representations from a $\mathcal{R}_1$ representation,

especially taking into account the rounding that occurs to turn the V_1's outputs into boolean features.

The second factor is to do with the sheer size of the space of underspecified representations, relative to the space of fully defined representations. The data set on which G_1 is trained represents $5.5 \times 10^{-29}\%$ of the entire space. G_1's purpose is to accurately map abstract representations so as to provide a correct seed representation for the imagination functor. That means that G_1's training can be seen as taking two inputs and two outputs and learning which features of the output are uniquely determined by the overlap of information in the two inputs. Given the size of the actual data it is very difficult to learn these relations.

All this points to the importance of the imagination functors and especially the use of the prediction functor within the fitness function. If the generators are conservative and map to more undefined representations (or even the fully undefined one), then the imagination functors can still find good solutions if the prediction functors are accurate. In fact, the imagination functor and prediction functors could be good enough to make the generator functors redundant. Furhtermore, the agent has access to the current tool state, which could be used as the seed for the imagination functors. That would mean the agent is completely reliant on the external representations to make any imaginative step.

The schemas use boolean features instead of real-valued feature in order to simplify the entropy and information content calculations. This choice also simplified the logic of adopting information by merging representations. It would be interesting to see how real-valued features could be incorporated, perhaps with the use of ranges or by using functions indicating the agent's preference for certain values.

One finding that could signify an addition to the SbCPF is that the agent needed a way of calculating pertinence. That is, we could add a functor to the SbCPF that models the agent's ability to measure the relevance of an imagined representation at a level $\mathcal{R}_n$ with regard to a goal at level $\mathcal{R}_{n-1}$.

It was not possible to train the agent's neural network functors in an online fashion. The system as a whole is currently too slow to take a reinforcement learning approach to training. Future work could look at the possibility of implementing all functors as one integrated neural network that does allow for online training.

The tool and schemas were designed to keep the complexity of the representations low. Specifically, the artefacts the agent can produce are of a fixed form, simplifying the implementation of many of the components. One could liken this to the form of a classic haiku. However, most creative domains are unlike this and allow for compositional, hierarchical, and unbounded forms. For example, a Blender document can have one cube in it or millions of complex shapes. It was not within the scope of this study to look at how an artificial agent can view, represent, and imagine such complicated artefacts effectively. However, it is worth reminding ourselves of the complexity of the constructs and corresponding content that human creators deal with in a creative process. Although this study has presented an instantiation of the SbCPF in a toy domain, it remains to be seen if it can lead to instantiations that are effective in more realistic domains.

The behaviour of the agent can be likened, at least in some ways, to the behaviour of tutors as described in Chapter 4. Like the tutors, the agent has goals at the content and construct levels, and it judges state at those levels separately. The agent exhibits a sort of framing in that lower level goals and stretches of activity have to be seen within the context of higher level goals that have been explicitly set. It is perhaps no surprise that the agent behaves in this way, because that is how it was designed. However, it was not a given that an instantiation of the

SbCPF would allow for the implementation of a simple policy functor that would result in such behaviour. Be that as it may, the gap between the behaviour of tutors and the behaviour of our agent is very large. Apart from the superficial similarities we cannot claim evidence that SbCPF describes human practice, although this study certainly does not contradict it either.

In sum, the agent implementation is a successful illustration and evaluation of the SbCPF.

Chapter 7

Discussion

In the preceding chapters we have worked towards a formal theory of creative practice. Chapter 3 introduced a novel method to study video tutorials, Chapter 4 and Chapter 5 presented the resulting grounded and formal theories of applying that method to Blender video tutorials, and Chapter 6 described an evaluation of the formal theory through computational modelling. In this final chapter we will summarise the contributions, discuss the more speculative elements, and relate our findings to the related work from Chapter 2. We will conclude with a discussion of future work.

7.1 Novel method

Our original intent was to develop techniques for analysing the session histories of software-based creative practice. The motivation was that analysing such session histories, in terms of the underlying cognition on the part of the creator, could prove helpful in reflecting on creative practice and in communicating of practice to others. Many authors take this line of argument (see Section 2.5). The review of literature showed that there were no clear theories of the cognition underlying creative practice that we could use to guide the development of analysis techniques. Our focus thus shifted to gaining the required insight into software-based creative practice. Chapter 3 represents the first salvo in that effort: a novel method to analyse the creative practice displayed in tutorial videos and extract from them a theory that describes the structure of creative practice.

The method combines several well-established methods and theories. It incorporates Grounded Theory, Rhetorical Structure Theory, Category Theory, and Gesture Theory. Rhetorical Structure Theory teases out the structure of a tutor's speech. Gesture Theory helps us argue that we can distinguish between action that is part of practice and gestures. Category Theory is the language and mechanism of formalisation. Grounded Theory is the glue that binds all these elements together. The final piece to making the method work is the novel Action-Speech Taxonomy (see Section 3.4.1.3) that sensitises the analyst to the relation between the tutor's actions and the speech describing them. Of course the proof of the pudding is in the eating and so the value of the method lies in the results it's able to produce.

7.2 A grounded theory

The grounded theory of creative practice in Blender video tutorials, and a tutor's communication of that practice, contains many interesting findings.

One major finding is the clear distinction tutors make between the content and construct levels of description. This mirrors the clear distinction that is present in complex creative software between the document format and the artefact's final form. Adobe Photoshop, with its layers and non-destructive effects, and Logic Pro, with its virtual instrument tracks and VSTs, are another two examples of software where this distinction is clearly present. Tutors reason about the content and construct levels explicitly. For example, construct states are sometimes judged in terms of their future potential, reminiscent of the Empowerment model of intrinsic motivation (see Section 2.4.3).

Content and construct states are both judged against some form of internal representation of the goal artefact. This can be clearly seen in the tutor's use of referring to nascent 3D structures by the name of the thing they are ultimately to represent. That is, a single vertex is the tip of a finger in the mind of the creator, even though it could be anything in the mind of an onlooker. Construct goals are described by tutors as being abduced from content goals and so it seems reasonable to suggest that a creator would not start a creative practice from a construct level goal. An exception is perhaps the case of a creator experimenting with new types of constructs and exploring what kind of contents they would result in.

The way tutors present and reason about their goals strongly suggest that creators have non-trivial internal representations and that goal representations precede any external representations. Goal representations are perhaps coupled, in the sense of embodied cognition, to the external artefact in some way. However, it is also clear that they are separate entities as is evidenced by how tutors speak about judging the state of the artefact. Goal representations are abstract to a degree, and formed just-in-time, but even so they could still not be convincingly described as high level and loosely defined as Baber does (see Section 2.3.5).

Tutors at times reason about Blender's workings explicitly, showing they have some representation of the workings of software features and that they can predict or imagine, to some degree of accuracy, the outcome of hypothetical actions. This must be the case in behaviour driven by some artefact goal, for it would be difficult otherwise for a creator to find their way through the over 1,500 available commands and their many parameter options. This is in line with the extended thesis of cognition. Not only is the Blender software part of the cognitive circuitry, but so is the creator's predictive model of Blender's features.

A minor finding, but one that aptly demonstrates the value of the method, regards the difference between *acting-to-see* and *approaching* (see Section 4.2.2). Both are actions of changing the 3D view (e.g. panning and zooming) without changing the artefact. However, one is used in inspecting and evaluating the artefact, while the other is part of positioning oneself prior to acting and should therefore be seen as part of modifying the artefact. This subtle, but important, idea would not have easily occurred to the authors of the rule-based segmentation of session history methods (see Section 2.5).

Tutors are sometimes seen to act with an uncertainty of what the consequence of their actions might be. They explicitly signal the uncertainty to the viewer, and so they are aware of it. This is compatible with the Predictive Processing view of cognition (see Section 2.3.6) in that tutors are predicting the outcome of their actions and that they do so with a certain precision. However, it should be noted that they do so in parallel to a construct goal that might itself be precise. For example, if a creator is unable to find an action that yields a goal they

can then drop that goal. A Predictive Processing based cognitive architecture would have to be able to deal with such artefact goal driven behaviour.

An interesting finding is how the method highlights points of analytical tension. These can be seen as pivot points in the creative practice where a tutor forms the construct goals for the next stretch of activity in context of the reflection of the activity just completed. Pivot points, and the way in which tutors frame activity, are clear segmenting markers. This is Schön *seeing-moving-seeing* in action (see Section 2.2.2).

The general schema of a chunk of tutorial activity is a sequence consisting of framing, narration, tweaking, and reflection. Narration often starts with an approaching action, tweaking involves repeated changes with minor effects, and reflecting coincides with changing the viewpoint. Techniques for the analysis of session histories could attempt to detect these observable action patterns and so segment the practice in chunks.

Findings with regard to the types of action that can accompany speech have made it possible to develop the Action-Speech Taxonomy. Although the different types (i.e. pointing, showing, and demoing) appeared easy to recognise, much like is the case with face to face gesturing, it would be interesting to see how strongly different analysts would agree in their use of the taxonomy.

By putting the above findings together a picture emerges of creative practice as a goal-driven process whereby the content and construct goals are to some degree abstract. The construct goal details are defined in flight. The moments of reflection and framing between chunks of activity serve to fully define partial construct details within the context of the current state of the artefact and the abstract construct and content goals. This notion of abstraction in goals and just-in-time detail definition is worked out in the formalisation of the grounded theory.

7.3 A formal theory

The final part of the method is to use the language of Category Theory as a way to develop and express the grounded theory formally. This has proven to be a very fruitful approach. The simultaneous flexibility and rigour of Category Theory allows exploring new ideas freely, while being driven to exactitude. The diagrammatic way of expressing theoretical constructs plays a large role in this. Ideas can be easily sketched, while the precise and deep semantics of those sketched diagrams immediately throw up questions and put inconsistencies in the spotlight. The experience of developing theory in this way can be likened to that of using sketching in design (see Section 2.2.2). However, learning Category Theory is not an easy task and could easily require a large investment of time. In our case it has proven to be worth the effort as it has helped to analytically work through the grounded theory and put it in formal attire.

The formal theory, which we have dubbed the Software-based Creative Practice Framework (SbCPF), presents a minimal model of the components of the cognitive system comprised of creator and software tool. The SbCPF is an embodied and extended model of cognition. The creator's understanding of the software tool is very clearly outlined and mirrors the structure of the tool itself.

The SbCPF posits that there are four distinct conceptual spaces: concept, content, construct, and action. These should not be seen as abstractions of one another whereby, for example, the content space contains higher-level descriptions of the construct concepts. Instead, these spaces describe entirely different things and so the maps between them are not relations of abstraction, but of causality. Abstraction relations do exist, but only between concepts within

a conceptual space. Maps between spaces should respect these abstraction relationships, or at least an agent should tend to respect them.

The *concept* conceptual space is perhaps somewhat of a clutch, for it is included as a catch-all mechanism to model any higher level conceptual meaning a creator might experience. That is, the cultural, social, or personal value of the content that is being created, as perceived by the creator, is pushed into this space. The SbCPF does not intend to make many statements about the conceptual meaning of content in the context of a concept goal, apart from that content goals serve to achieve concept goals.

Imagination is modelled as traversal of the conceptual space. The SbCPF leaves the manner of traversal completely open, and so models of creative imagination like Wiggins' CSF (see Section 2.4.2) can fill the gap left by the SbCPF. It is worth noting however that in the SbCPF the imagination functors are highly dependent on the prediction and classification maps between conceptual spaces. They play a role within the traversal process of the conceptual spaces and without them imagined concepts could not be mapped to action.

The SbCPF models conceptual spaces as spaces of instances of schemas. The schemas define types and their properties. It is through the schema definition that the notion of abstraction is made precise. We should admit that this is a very opinionated definition of the conceptual spaces and that there are other ways of defining the conceptual spaces. For example, Wiggins' CSF uses a language of thought and the Predictive Processing (PP) thesis uses generative models.

Although the SbCPF is unmistakably a model of cognition in which prediction plays a large role, that role is not the same as the one that prediction plays in PP. Prediction in the SbCPF is vital in the formation of the hypothesis concept-content-construct-action chain. The prediction maps are in fact the creator's understanding of the software tool they are using. In PP the role of prediction is in the process of perception and at the lower levels of motor control. In fact, SbCPF and PP are not quite playing in the same league. SbCPF is a descriptive model of creative practice cognition as it describes the behaviour of the cognitive system. PP, on the other hand, is more of an explanatory model and could very well underlie the behaviour described by the SbCPF.

The motivation of an agent, as per the SbCPF, is simply defined as a drive to reduce the entropy present in an imagined artefact. The entropy is the quantification of the amount of detail that is undefined still and is determined by the agent's notion of the probability of its observations. This could be seen as an intrinsic motivation that is similar to that described by Schmidhuber. However, Schmidhuber's notion of motivation would sit best at the content and concept levels, for it considers the final form of the artefact. It is also best suited for the selection of interesting conceptual ideas for artefacts in the first place. The SbCPF notion of entropy-reduction is better suited for the levels below the concept space as it motivates the imagination of the lower level details in the moment-to-moment interaction with the software tool.

The entropy-reduction mechanism also plays a role in the imaginative power limit, which posits that an agent is limited in the amount of details that they can imagine at once. It is for this reason that an agent is required to approach the creation of artefacts in an embodied and interactive manner, like is the case with sketching in design (see Section 2.2.2). The concept of precision from PP could be helpful here as well. If an agent could assess the certainty of its own hypotheses chains then it could decide to take smaller imagination steps and validate its hypotheses more frequently. In other words, if the agent is uncertain about its strategy then they would imagine fewer details at once.

From an AI perspective the SbCPF is a reinforcement learning model. It generates a hypothesis for action and improves its prediction with each and every observation of corresponding action, construct, content, and concept. With use, the agent learns to understand the tool better and the quality of the hypotheses improve.

Although the SbCPF describes what components a practically creative agent would need, it does not explain how those components combine to produce actual behaviour. This issue is explored in the computational evaluation of the theory.

7.4 Computational model

Chapter 6 presents an implementation of an artificial agent that is capable of using a very simple tool. With the tool the agent fills the nine cells of a square board with image tiles. The agent implementation is a computational model of the SbCPF. It serves partly as an evaluation, partly as illustration, and partly as further exploration. As evaluation the computational model shows that the formal theory is internally consistent and that it can serve successfully as a software architecture. It follows the architecture very closely and the only deviation is the addition of the concept view functor that serves to simulate the conceptual level. The exploratory part of the computational model is the development of a behaviour policy that combines the agent's components to produce coherent behaviour.

The policy functor is a simple recursive definition. Its main job is to judge the outcome of actions against currently held goals. The policy's effectiveness is improved significantly by being able to retry new hypotheses after a previous hypothesis was proven false, especially given that the agent is very strict in judging artefact against the imagined goal artefact. Retrying is thus a valid strategy to overcome prediction errors, as long as the agent can come up with different hypotheses. Online learning should prevent the agent trying to use the same hypothesis over and over again in the first place.

A particularly interesting finding in implementing the policy was the realisation that the generator functor was superfluous in the simple board-filling context. In fact, the perceived current artefact state proved sufficient as the seed for the imagination functor. That is just as well as the generator maps are difficult to learn due to the fact that a particular content idea could potentially be realised by more than one construct, and a construct by more than one action. That is, generator maps have to learn one to many relations, while classification maps learn simpler one-to-one relations. This means that although observations of construct and corresponding content can be used directly to learn the classification map, they have to be processed differently to handle the one-to-many relation. In online learning of the generator map the agent thus has to reconcile new observations with old in a significantly more complex way. This is not only a problem in the maps between conceptual spaces, but is a problem for maps that learn abstraction relations as well. That is, any layer that re-describes abstractly another layer of description throws away information, and thus runs the risk of having to solve the one-to-many problem in prediction. PP deals with this problem by positing that predictions take the form of probability distributions. All this raises the prospect that imagination *must* happen at every layer of representation when one hopes to work from a high-level idea to concrete action. Otherwise, full detail needs to emerge from within the prediction from an abstract high-level idea. It is not clear what kind of mechanism could provide the required low-level detail rationally.

Also, this provides insight into why it is so beneficial to work with external representations.

Specifically, in the absence of precise or correct predictions from a higher level an agent is able to form a concrete and specific hypothesis by imagining details at the lower level. It could take the perceived current artefact state as the imagination seed and test any imagined new states against the higher level goal with the classification functor. In other words, imprecise prediction is helped by imagination and classification and can even be replaced by the combination of imagination and classification.

The implementation of the agent is largely domain agnostic. The view functors and actual schemas are all specific to the board-filling tool, but the policy, imagination, adoption, classification, and generator functors are not. This shows that the SbCPF is indeed a general theory and that the computational model is not, for the most part, an ad hoc implementation optimised for board-filling.

One criticism that could be levelled against the computational model is its simplicity. The complexity of the board-filling tool is nothing like the complexity of a tool like Blender, or any tool that could lead to really creative artefacts. An important factor in this is that the tool does not allow the construction of compositional constructs, and consequently that the range of possible contents is limited. We would agree that the complexity of the tool is unsatisfactory. However, the agent is of non-trivial complexity, which shows that an architecture that can deal with different levels of conceptual spaces and abstraction in representations is not an easy task (at least not for us). Also, the computational model has led to useful and interesting results, despite its simplicity. Another consideration is that a computational model that can create compositional constructs is significantly more complex, something we hope to work on in the future.

7.5 Future work

This brings us to the main line of inquiry that future work would focus on. The vast majority of creative software tools are compositional in nature. This compositionality makes that the space of possible constructs is infinite. Working on compositional constructs requires a breaking down of goals into sub-goals, even within conceptual levels, to be able to deal with the boundless complexity. This has repercussions for the way goals are represented, how artefacts are viewed, how the classification and generators map between goals at different levels, and how abstraction and underspecification is modelled. There is nothing inherent in the SbCPF that would prevent it to being extended to handle these issues, but it is currently unclear how to do so. Future work would seek answers to these questions.

One angle of attack would be to look at the theory of conceptual chunking (Gobet et al. 2001; Fernand Gobet 2005). We speculate that chunking plays an important part in how creators break down goals into sub-goals and therefore in the structuring of creative practice. By developing the theory further we would expect to be able to move incrementally from the current descriptive theory to an explanatory theory of creative practice and therefore to a theory that is more easily falsifiable.

Another aim in future work would be to bring the computational model more in line with the current state of the art in AI research. For example, the Deep Reinforcement Learning approach could unlock more complicated tool use. Perhaps an architecture where all SbCPF functors are components in a larger neural network would enable faster training and make it possible to run more complex experiments.

7.6 Conclusion

Our original aim was to automatically analyse sessions of creative software use. The idea was that we could look at the button presses and command invocations and understand, to some degree, the cognitive process underlying the observed creative practice. Upon studying the literature we found no satisfactory model of creative practice upon which such an analysis could be based. For that reason, we set out to develop such a model.

The first contribution towards that goal is a novel method to analyse the creative practice displayed in video tutorials. The method's foundations are well-established theories and by leveraging the new Action-Speech Taxonomy it produces both a grounded theory and a formal theory.

The second contribution is an in depth grounded theory of creative practice as displayed and communicated in Blender video tutorials. Major findings are the existence of the content and construct levels and the way tutors appear to formulate goals in flight. The grounded theory is especially useful as an example of human behaviour that theories of cognition should be able to explain and therefore where they might be lacking.

The third contribution is the formalisation of the grounded theory in the form of the Software-based Creative Practice Framework. It takes the elements of the grounded theory and makes them precise. Its main qualities are that it a) it considers both creator and tool part of the cognitive system, b) it shows how an agent's predictions of tool actions are incorporated into reasoning, c) it characterises creative practice as an iterative working from an abstract idea to a concrete artefact, and d) it does so by imagining the details of the artefact in flight and in a close coupling with the external artefact. The SbCPF is descriptive of the cognition of creative practice rather than explanatory. It therefore serves as a useful instrument to discuss and compare related theories such as the Creative Systems Framework and the views on spontaneous creativity by Wiggins and Clark.

The final contribution is the computational model as an evaluation and further exploration of the SbCPF. It gives insight into why predictive models of cognition might require that creative practice is interactive in nature. It does so by showing how limited prediction precision can be combined with imagination with the current external artefact state as the seed.

Finally, future work should focus on how creators deal with the compositionality of creative software tools and how compositionality shapes creative practice.

Appendices

Appendix A

Rhetorical Structure Theory relations

The definitions of the Rhetorical Structure Theory (RST) relations, in the tables below, have been copied from the RST website[1]. The relations definitions are to be interpreted as described on the website:

> Each element of the definition is implicitly embedded in a constraint formula as follows: "It is plausible to the analyst that it was plausible to the author that ... ."
>
> On terminology in definitions: N stands for nucleus, S for satellite, W for writer (author, speaker) and R for reader (hearer.) For some brevity: in many places, N and S stand for the situations presented by N and S; N and S never stand for the text of N or S. Situation is a broad cover term that ranges over propositions or beliefs, actions whether realized or not, desires to act and approval for another to act. Similarly, positive regard is a broad attitudinal term that ranges over belief, approval of ideas, desire to act, and approval for another to act, all identifiably positive. Positive regard and belief (with its cognates), and plausible above are all degree terms, not binary.
>
> Notice that the name of the relation does not enter into the use of the definition. Finding good names for all of the relations is difficult or perhaps impossible, so some of the definitions will seem inappropriate for the name; see, for example, Justify.
>
> The relations are classified into two main types: nucleus-satellite and multinuclear (see the Introduction). They can also be classified according to whether they are Presentational or Subject Matter. Presentational relations are those whose intended effect is to increase some inclination in the reader, such as the desire to act or the degree of positive regard for, belief in, or acceptance of the nucleus. Subject matter relations are those whose intended effect is that the reader recognizes the relation in question.

[1] `https://www.sfu.ca/rst/01intro/definitions.html`

Table A.1: Definitions of the RST presentational relations.

Presentational Relations			
Relation Name	**Constraints on either S or N individually**	**Constraints on N + S**	**Intention of W**
Antithesis	on N: W has positive regard for N	N and S are in contrast (see the Contrast relation); because of the incompatibility that arises from the contrast, one cannot have positive regard for both of those situations; comprehending S and the incompatibility between the situations increases R's positive regard for N	R's positive regard for N is increased
Background	on N: R won't comprehend N sufficiently before reading text of S	S increases the ability of R to comprehend an element in N	R's ability to comprehend N increases
Concession	on N: W has positive regard for N on S: W is not claiming that S does not hold;	W acknowledges a potential or apparent incompatibility between N and S; recognizing the compatibility between N and S increases R's positive regard for N	R's positive regard for N is increased
Enablement	on N: presents an action by R (including accepting an offer), unrealized with respect to the context of N	R comprehending S increases R's potential ability to perform the action in N	R's potential ability to perform the action in N increases
Evidence	on N: R might not believe N to a degree satisfactory to W on S: R believes S or will find it credible	R's comprehending S increases R's belief of N	R's belief of N is increased
Justify	none	R's comprehending S increases R's readiness to accept W's right to present N	R's readiness to accept W's right to present N is increased
Motivation	on N: N is an action in which R is the actor (including accepting an offer), unrealized with respect to the context of N	Comprehending S increases R's desire to perform action in N	R's desire to perform action in N is increased
Preparation	none	S precedes N in the text; S tends to make R more ready, interested or oriented for reading N	R is more ready, interested or oriented for reading N
Restatement	none	on N + S: S restates N, where S and N are of comparable bulk; N is more central to W's purposes than S is.	R recognizes S as a restatement of N
Summary	on N: N must be more than one unit	S presents a restatement of the content of N, that is shorter in bulk	R recognizes S as a shorter restatement of N

Table A.2: Definitions of the RST subject matter relations.

Subject Matter Relations			
Relation Name	**Constraints on either S or N individually**	**Constraints on N + S**	**Intention of W**
Circumstance	on S: S is not unrealized	S sets a framework in the subject matter within which R is intended to interpret N	R recognizes that S provides the framework for interpreting N
Condition	on S: S presents a hypothetical, future, or otherwise unrealized situation (relative to the situational context of S)	Realization of N depends on realization of S	R recognizes how the realization of N depends on the realization of S

Table A.2: Definitions of the RST subject matter relations.

Subject Matter Relations			
Relation Name	**Constraints on either S or N individually**	**Constraints on N + S**	**Intention of W**
Elaboration	none	S presents additional detail about the situation or some element of subject matter which is presented in N or inferentially accessible in N in one or more of the ways listed below. In the list, if N presents the first member of any pair, then S includes the second: set-member, abstraction-instance, whole-part process-step, object-attribute, generalization-specific	R recognizes S as providing additional detail for N. R identifies the element of subject matter for which detail is provided.
Evaluation	none	on N + S: S relates N to degree of W's positive regard toward N.	R recognizes that S assesses N and recognizes the value it assigns
Interpretation	none	on N + S: S relates N to a framework of ideas not involved in N itself and not concerned with W's positive regard	R recognizes that S relates N to a framework of ideas not involved in the knowledge presented in N itself
Means	on N: an activity	S presents a method or instrument which tends to make realization of N more likely	R recognizes that the method or instrument in S tends to make realization of N more likely
Non-volitional Cause	on N: N is not a volitional action	S, by means other than motivating a volitional action, caused N; without the presentation of S, R might not know the particular cause of the situation; a presentation of N is more central than S to W's purposes in putting forth the N-S combination.	R recognizes S as a cause of N
Non-volitional Result	on S: S is not a volitional action	N caused S; presentation of N is more central to W's purposes in putting forth the N-S combination than is the presentation of S.	R recognizes that N could have caused the situation in S
Otherwise	on N: N is an unrealized situation on S: S is an unrealized situation	realization of N prevents realization of S	R recognizes the dependency relation of prevention between the realization of N and the realization of S
Purpose	on N: N is an activity; on S: S is a situation that is unrealized	S is to be realized through the activity in N	R recognizes that the activity in N is initiated in order to realize S
Solutionhood	on S: S presents a problem	N is a solution to the problem presented in S;	R recognizes N as a solution to the problem presented in S
Unconditional	on S: S conceivably could affect the realization of N	N does not depend on S	R recognizes that N does not depend on S
Unless	none	S affects the realization of N; N is realized provided that S is not realized	R recognizes that N is realized provided that S is not realized
Volitional Cause	on N: N is a volitional action or else a situation that could have arisen from a volitional action	S could have caused the agent of the volitional action in N to perform that action; without the presentation of S, R might not regard the action as motivated or know the particular motivation; N is more central to W's purposes in putting forth the N-S combination than S is.	R recognizes S as a cause for the volitional action in N

Table A.2: Definitions of the RST subject matter relations.

Subject Matter Relations			
Relation Name	**Constraints on either S or N individually**	**Constraints on N + S**	**Intention of W**
Volitional Result	on S: S is a volitional action or a situation that could have arisen from a volitional action	N could have caused S; presentation of N is more central to W's purposes than is presentation of S;	R recognizes that N could be a cause for the action or situation in S

Table A.3: Definitions of the RST multinuclear relations.

Multinuclear Relations		
Relation Name	**Constraints on each pair of N**	**Intention of W**
Conjunction	The items are conjoined to form a unit in which each item plays a comparable role	R recognizes that the linked items are conjoined
Contrast	No more than two nuclei; the situations in these two nuclei are (a) comprehended as the same in many respects (b) comprehended as differing in a few respects and (c) compared with respect to one or more of these differences	R recognizes the comparability and the difference(s) yielded by the comparison is being made
Disjunction	An item presents a (not necessarily exclusive) alternative for the other(s)	R recognizes that the linked items are alternatives
Joint	None	none
List	An item comparable to others linked to it by the List relation	R recognizes the comparability of linked items
Multinuclear Restatement	An item is primarily a reexpression of one linked to it; the items are of comparable importance to the purposes of W	R recognizes the reexpression by the linked items
Sequence	There is a succession relationship between the situations in the nuclei	R recognizes the succession relationships among the nuclei.

Appendix B

Source code

B.1 Archive

The source code can be found on the DVD included with the printed copy of this thesis. The source code is organised as per the directory tree in table B.2. The "python" directory contains all of the agent, tool, and simulation code. The "visualiser" directory contains the Qt[1] source code for the tool to visualise simulation logs.

python	All agent, tool, and simulation code
notebooks	Jupyter notebooks for training views and plotting
src	
patagotchi	
agent	All code for the agent, tool, and simulations
creators	Contains the policy functor and agent definition
environment	Contains the simulation code
imagination	Imagination functor
maps	Prediction and generator functors
perform	Perform functor
schemas	Schemas for all levels, instance (de)serialisation
views	View functors
commands	Command line tools to generate data sets and training of prediction and generator functors
tool	All code related to the tool
tool/assets/google/24/coverage	Contains all image tile assets
visualiser	The Qt utility to visualise simulation logs

Table B.2: The directory tree of the source code and the descriptions of what can be found in each directory.

B.2 Partial order diagram

Listing B.1: This is the Python source code with which the diagram in fig. 5.2 was generated.

```
1  import collections
2  import itertools
3  import graphviz
4
5  ALL_WAITERS = set(['w1', 'w2'])
6  ALL_TABLES = set(['t1', 't2'])
7
8  class SchemaInstance(object):
9      def __init__(self, waiters, tables, assignments):
10         self.waiters = waiters
11         self.tables = tables
12         self.assignments = assignments
13
14     def __repr__(self):
```

[1] Qt is a framework to build GUI applications for desktop or mobile. https://www.qt.io

```
        result = ",".join(["%s%s" % (key, "*" if val == None else val) for key,
                           val in sorted(self.assignments.items())])
        idle_waiters = self.waiters.difference(set(self.assignments.values()))
        result += "(%s)" % ",".join(sorted(idle_waiters))
        return result

    def __le__(self, other):
        return (other.waiters <= self.waiters) \
            and (other.tables <= self.tables) \
            and (set(other.assignments.keys()) <= set(self.assignments.keys())) \
            and (all([other.assignments[table] == None or
                      other.assignments[table] == self.assignments[table] for table
                          in
                      other.assignments.keys()]))

    def __lt__(self, other):
        return self <= other and self != other

    def __eq__(self, other):
        return other.waiters == self.waiters \
            and other.tables == self.tables \
            and other.assignments == self.assignments

def all_combinations(options):
    return itertools.chain(*[itertools.combinations(options, i) for i in
                             range(0, 1+len(options))])

def all_instances(max_waiters, max_tables):
    results = []
    for inhabitants in itertools.product(all_combinations(max_waiters),
                                         all_combinations(max_tables)):
        for picks in itertools.product(inhabitants[0] + (None,),
                                       repeat=len(inhabitants[1])):
            si = SchemaInstance(set(inhabitants[0]), set(inhabitants[1]),
                                dict(itertools.izip(inhabitants[1], picks)))
            results.append(si)
    return results

def direct_link(inst1, inst2, instances):
    for inst3 in instances:
        if inst1 < inst3 and inst3 < inst2:
            return False
    return True

if __name__ == '__main__':
    dot = graphviz.Digraph()
    dot.attr(rankdir='LR')
    dot.node_attr['fontname'] = 'Latin␣Modern␣Roman,LM␣Roman␣12:style=12␣Regular,
        Regular'
    instances = all_instances(ALL_WAITERS, ALL_TABLES)
    for inst in instances:
        dot.node(str(inst))
    for (inst1, inst2) in itertools.product(instances, instances):
        if inst1 < inst2 and direct_link(inst1, inst2, instances):
            dot.edge(str(inst1), str(inst2))
    dot.render('ch5-example-lattice.gv')
```

B.3 Agent's action schema

Listing B.2: This is the Python source code for the agent's action schema.

```
from patagotchi.agent.schemas import Schema, SchemaObject, Integer
import numpy as np


class Row(Integer):
    pass
```

```
7
8
9  class Column(Integer):
10     pass
11
12
13 class Tile(Integer):
14     pass
15
16
17 class Command(SchemaObject):
18     pass
19
20
21 class AddTile(SchemaObject):
22
23     def __init__(self, command, row, column, tile):
24         self.isA = command
25         self.row = row
26         self.column = column
27         self.tile = tile
28
29
30 class DeleteTile(SchemaObject):
31
32     def __init__(self, command, row, column):
33         self.isA = command
34         self.row = row
35         self.column = column
36
37
38 class SchemaAction(Schema):
39
40     def __init__(self, rows, columns, num_tiles):
41         self.rows = rows
42         self.columns = columns
43         self.num_tiles = num_tiles
44         self.command = Command()
45         self.row = Row(0, rows)
46         self.column = Column(0, columns)
47         self.tile = Tile(0, num_tiles)
48         self.add_tile = AddTile(self.command, self.row, self.column, self.tile)
49         self.delete_tile = DeleteTile(self.command, self.row, self.column)
```

B.4 Agent's construct schema

Listing B.3: This is the Python source code for the agent's construct schema.

```
1  from patagotchi.agent.schemas import Schema, SchemaObject, Integer
2  import numpy as np
3
4  class TileId(Integer):
5      pass
6
7
8  class Board(SchemaObject):
9
10     def __init__(self, rows, columns, tile_id):
11         for row in range(rows):
12             for column in range(columns):
13                 setattr(self, "cell_{0}_{1}".format(row, column), tile_id)
14
15
16 class SchemaConstruct(Schema):
17
18     def __init__(self, rows, columns, num_tiles):
19         self.rows = rows
```

```
        self.columns = columns
        self.num_tiles = num_tiles
        self.tile_id = TileId(upper=num_tiles)
        self.board = Board(rows, columns, self.tile_id)
```

B.5 Agent's content schema

Listing B.4: This is the Python source code for the agent's content schema.

```
from patagotchi.agent.schemas import Schema, SchemaObject, Boolean
import numpy as np

class Cell(SchemaObject):

    def __init__(self, feature, num_features_per_cell):
        for i in range(num_features_per_cell):
            setattr(self, "feature_{0}".format(i), feature)


class Board(SchemaObject):

    def __init__(self, rows, columns, cell):
        for row in range(rows):
            for column in range(columns):
                setattr(self, "cell_{0}_{1}".format(row, column), cell)


class SchemaContent(Schema):

    def __init__(self, rows, columns, num_features_per_cell):
        self.rows = rows
        self.columns = columns
        self.num_features_per_cell = num_features_per_cell
        self.feature = Boolean()
        self.cell = Cell(self.feature, num_features_per_cell)
        self.board = Board(rows, columns, self.cell)
```

Appendix C

3Dami data set

The 3Dami organisation[1] is a non-profit that runs seven day long 3D animation workshops. They invite kids from 14 to 18 and teach them how to make a minute-long animated short. The aim is to inspire kids to take up 3D modelling and animation, partly as an introduction to computational thinking. 3Dami promotes the use of open-source software such as Blender[2].

3Dami kindly allowed us to run a study at their 2014 summer workshop. All workshop participants used a modified version of Blender so that we could record all interactions with Blender as well as screenshots. After some attempts at analysing the data set it became clear that we did not have the proper theoretical foundations and our work shifted to what is presented in this thesis. As such, the data set is not used for this work and it was never published.

Because this study was instrumental in shaping the work of this thesis we include here an unpublished paper describing the data set. Our aim is to use the data set in future work.

[1] http://www.3dami.org.uk/
[2] https://www.blender.org

The 3Dami Data Set: Observations of Practical Creative Processes in 3D Animation

Vincent Akkermans
Queen Mary University of London
v.akkermans@qmul.ac.uk

Tom S.F. Haines
University College London
t.haines@cs.ucl.ac.uk

Peter E.J. Kemp
University of Roehampton
peter.kemp@roehampton.ac.uk

Mark B. Sandler
Queen Mary University of London
mark.sandler@qmul.ac.uk

Geraint A. Wiggins
Queen Mary University of London
geraint.wiggins@qmul.ac.uk

ABSTRACT

We present an open data set of practical creative processes performed in the Blender 3D modelling and animation software. The data was collected during the 3Dami summer studio during which three teams of in total 26 participants, aged 13 to 18, made three short animations over the course of seven days. The data set contains all usage interaction, interface screenshots, project files, assets, and asset manager logs. We describe the collected data in detail as well as the method by which it was collected. We furthermore present some initial analysis results. Additionally, we provide a qualitative description of the activities and experiences of the participants during the summer studio to aid in the data set's interpretation.

Author Keywords

Creativity; data set; open data; animation; cognition; computational thinking; computational creativity.

ACM Classification Keywords

H.5.2 User Interfaces: Evaluation/methodology

INTRODUCTION

Creativity

Creativity is a defining feature of human cognition and of vital cultural, social, and economic importance. It is studied from many angles and what the term denotes exactly depends on the field of study [30]. Boden emphasises that creativity is part of human cognition and writes that "creativity is the ability to come up with ideas or artefacts that are new, surprising and valuable" [2]. This definition is often used, for example in the field of computational creativity, which attempts to verify models of creative cognition through the implementation of cognitive architectures [33]. Csikszentmihalyi rejects the notion that creativity happens solely in the mind and posits that it is something that occurs in the interaction between cognition and the sociocultural context [5]. Others emphasise the psychological aspects of creativity [24], or that creativity can be a group phenomenon such as when people make music together [21, 26]. We can thus see that creativity can refer to people, minds, groups, communities, or artefacts. However, we will here be focused on another aspect: creative practice.

This work is 'Open Access', published under a creative commons license which means that you are free to copy, distribute, display, and perform the work as long as you clearly attribute the work to the authors, that you do not use this work for any commercial gain in any form and that you in no way alter, transform or build on the work outside of its use in normal academic scholarship without express permission of the author and the publisher of this volume. Furthermore, for any reuse or distribution, you must make clear to others the license terms of this work. For more information see the details of the creative commons licence at this website: http://creativecommons.org/licenses/by/3.0/.

Practical creative processes

It would be hard to argue that novel *ideas* do not originate in the mind, even if they're inspired (i.e. caused) by the perception of some phenomenon external to the mind. However, no cultural *artefact* is created in the mind alone, nor is it there envisioned in full form and later simply realised [11]. We thus have the thought that the practical process of making is a vital part of the conception of the artefact, or in other words, that practice influences creative cognition as much as the cognition influences practice. But what then is the structure of a practical creative process?

The creative process is sometimes seen as consisting of phases such as: preparation, incubation, insight, evaluation, and elaboration [5, 29]. These models are said to be recursive, in the sense that within each phase one might observe a complete process again. This is however problematic in that, if one follows this recursion ad infinitum, these descriptive theories of the creative process cannot completely explain how practical processes actually form. In contrast, Schön's theory of reflection-in-action provides a bottom-up explanation [27, 28]. It emphasises the fact that in order to solve problems practitioners interact with those problems and rely on tacit knowledge that can only be employed practically. Many other authors have also laid bare the interactive structure underlying the process in which designers solve problems [6, 9, 14, 15, 17, 25]. In this light it makes sense to view practice as part of the "cognitive circuitry" [4], especially when talking about practical creative processes.

Supporting creativity

Of course many creative workflows are now entirely software-based. In the HCI community quite some effort has been undertaken to study how tools can be improved to better support creativity [3, 12, 20, 29, 31]. Closely related is the work that investigates how the creative use of software can be logged, analysed, and exploited. [1, 10, 16, 18, 19, 23]. These studies highlight the potential, mostly educational, benefits of having access to information about how artefacts came to be. For example, it can provide opportunities for self-monitoring and reflection, for learning techniques from others without the need for explicit demonstration, or it can be used as documentation for designers or their collaborators.

Observation, understanding, and narration

Where the interactive nature of practical creative processes makes that they are observable, the pervasive use of software tools means that their observation is now also feasible. Given the importance of creativity it seems to us that there are still many ways in which to capitalise on this opportunity. Our particular interest is to make it so that every creative artefact made with software carries with it a description of the practical creative process that birthed it. In other words, how do we instrument an artist's tools such that they not only enable the creation of a work, but are also able to describe the process of creating that work?

Because this aim is without doubt overly ambitious we start by framing three questions. Firstly, how do we instrument software so that we are able to observe, and corollary, how much of the creative process is actually observable? Secondly, given these observations how are we to understand (or in a more strict sense represent) the process? Finally, given a model of a particular process, how should we narrate it so that the user is best served in their purpose? This work focuses on the first of these questions.

We'll have to restrict our scope further, for without picking a domain of practice we cannot practically continue. We have chosen to study practical creative processes in Blender[1], an open source 3D modelling and animation software. Blender was chosen because it a) can be freely modified, b) has a large community of users, and c) is a non-trivial software allowing for a great range of creative uses.

The 3Dami data set

Of course we cannot properly study these questions without a data set, and as of yet no open data set of practical creative processes exists. It is therefore that we have produced a data set of practical creative processes performed in Blender. It includes usage data, screenshots, Blender project files and assets, and asset manager logs. All data was collected during the 3Dami Summer Studio 2014, a seven day long event during which participants made three short films. We are publishing this data set openly and freely under a CC BY-NC-SA license[2]. The 3Dami data set is available for download online[3].

[1] `http://www.blender.org/`

[2] `https://creativecommons.org/licenses/`

[3] `http://www.inspectorb.com/3Dami`

It is this data set that we present in this work, which is organised as follows. The *method* section introduces the educational design of the 3Dami summer studio and the data collection method, the *results* section describes what happened during the 3Dami event and also describes in detail the resulting data set. We conclude with a quick discussion of the results, conclusions, and future work.

METHOD

3Dami summer studio

The 3Dami Summer Studio 2014 event[4] took place in London, England from 24 July to 1 August. There were 26 participants, 6 female and 20 male, all aged between 13 and 18. They worked as three separate teams, each creating a short film over the course of seven work days. The aim of 3Dami events is to teach the participants the entire process of making an animated film, from story development to post-production.

In this section we describe in some detail the educational design and schedule of the summer studio, as this will help explain the collected data or, in some cases, the absence thereof.

Participants were selected by 3Dami. Prospective participants were required to prove their motivation and basic grasp of Blender by submitting a portfolio. Motivation demonstrates that the participants will both turn up and work hard enough to complete their films in time. Participants with a wide variety of skills were selected, so that each team had members with the different skills required for making a film (e.g. modelling, texturing, rigging, animation). No criteria for selection of participants were inspired by the aim of performing data collection. All participants, or their legal guardians, agreed to the data collection before the start of the summer studio.

Participation to 3Dami was without monetary charge and participants could apply for travel bursaries. Participants were also given a small budget to buy lunch. Food and drinks were provided throughout the day.

The educational philosophy of 3Dami is based on constructivism, and specifically constructionism [22]. It holds that students learn best through constructing public artefacts. One of the early examples of constructionism was to make teaching mathematics like teaching art classes: through free creative exploration. The 3Dami event is similarly engaged in that it attempts to teach computational thinking (i.e. decomposition, pattern recognition, abstraction, logical reasoning, algorithm design, etc.) through 3D animation.

The schedule of a typical 3Dami summer studio is as follows.

1st Thursday

On the first day of the summer studio the participants are assigned to one of the three teams: Red, Green, or Blue. Participants meet their team members and attend a lecture on film making in the morning. In the afternoon the teams develop ideas for scripts and make story boards.

1st Friday

On the second day participants attend a lecture on the communication and technical skills required for working in teams,

[4] `http://www.3dami.org.uk/`

after which each team votes in a director and a producer. The participants now start using Blender by making an animatic (i.e. a preliminary version of the film made from the storyboard drawings) and the required assets for their films. In the afternoon the participants attend a presentation by an industry professional. Participants then break for the weekend and, although it is not required of them, it is expected that some participants continue to work on their film.

Monday
The participants continue their work, creating characters, sets, and props.

Tuesday
The participants should now focus on finishing character models. A trip to a movie studio is planned for the afternoon.

Wednesday
Character animation becomes the key focus. Another industry professional comes to give a presentation in the morning.

2nd Thursday
By the end of the day all of the shots should be done, so they can be rendered overnight and be available the following day.

2nd Friday
The day is spent on compositing, sound design, polishing, and creating the final film. In the evening family and friends are invited to the films' premieres.

On the first three days all the participants go out to lunch together. On the last 4 days half the members of each team go out for lunch and bring back food for the others.

Data collection
As mentioned before we are observing practical creative processes in Blender. This requires us to a) decide what to observe, b) instrument Blender such that we can observe, and c) implement a data collection and storage mechanism for the observations.

Instrumenting Blender
We have conceptualised each action within Blender as an event that can be described at five distinct levels of description (see table 1). This conceptualisation is similar to that presented in [13]. At the first level of description we think of an action as a basic action [32], that is purely described in physical movements of the body. As we increase the scope of the description we include in it the consequences of the action. For example, pressing the 'x' key on the keyboard has the consequence of deleting the currently selected object. An alternative conceptualisation is to see the basic action (the pressing of 'x') as a separate event from the consequence it causes (the deletion). We have not opted for this latter view as it does not fit the way in which users tend to think of or plan action. With the levels described in tables 1 we thus have a framework for describing what and how we observe interaction with Blender.

What can be observed is constrained foremost by how Blender is implemented. We have instrumented Blender to record four types of observations. The first type of observation, at the device input level, consists of mouse movements, mouse clicks, and keyboard events. These are the closest descriptions of physical actions possible. In our implementation subsequent mouse movements are aggregated into a single sequence of mouse positions so as to attenuate the volume of observations.

The second type, at the User Interface (UI) interaction level, is the simple button press. We did not find a good general way of recording other events at this level, such as interactions with special widgets.

We now come to the more informative levels. At the level of prescriptive changes we observe assignments and operators. Assignments are typically simple property changes, such as setting the render size. Operators are the main mechanism of implementing application logic, such as moving about objects in 3D space. Operators are essentially reified functions and can therefore be introspected at runtime. They are uniquely identified by a label and are parameterised (an OWL[5]/RDF[6] description of all Blender operators is available online[7]).

At the final level, that of descriptive changes, we find the actual changes to the document and program state. We investigated whether it was possible to collect each revision of the document or, alternatively, build revisions into the Blender document format. However, this proved too difficult. Instead we extract a representation of the state after each operator invocation. This includes, for example, a summary description of all objects in the document. Also, upon an operator invocation a screenshot is taken and the operator and the screenshot are tagged with the same unique identifier (screenshots were rate limited to one per second).

Thus, each interaction is described at level 2 through 5 (level 1 being the basic action), whereby each level is an interpretation of the previous one, taking into account the context of the program state.

In addition we collect information when Blender launches and when it finishes. A unique session key is generated at launch. This key, together with information about the Blender version, operating system, and hardware, is recorded in a 'session start' observation. Any subsequent observations of the types described above are tagged with the session key. The 'session end' observation helps to indicate that we have observed a full session. We thus define a session as being comprised of all the interactions that occur in a single instance of the execution of the application. It is possible that a user has more than one session open at a time and that they switch between them.

Blender was instrumented by placing hooks at certain points of its event handling loop (source code is available for perusal[8]). Upon calling such a hook relevant information is gathered and a Thrift[9] object is created. The Thrift language and protocol allow for data structures to be described in a do-

[5] `http://www.w3.org/OWL/`
[6] `http://www.w3.org/RDF/`
[7] `http://rdf.inspectorb.com`
[8] `https://github.com/InspectorB/blender-toc`
[9] `https://thrift.apache.org/`

Layer	Description
Physical action *movements of the body*	At the first level we find the basic physical actions performed by the user such as moving the hands. These are typically referred to as basic actions.
Device input *mouse and keyboard events*	Events at the device input level are descriptions of the basic actions at level 1. Observations at this level are thus direct observations of the physical actions performed by the user such as moving the mouse or pressing a key.
UI interaction *button presses, 3D viewport manipulators, menu selection, frame selection, tab selection*	At the User Interface level we find descriptions of events at the device input level as they are interpreted by the Blender UI. For example, a mouse click is here interpreted as a button press, a menu selection, or other UI widget interactions.
Prescriptive changes *operators and assignments*	At this level the interaction with a widget has been interpreted as a specification of a change that is to occur to the document or program state. Actions described at this level most closely align with the intention of the user.
Descriptive changes *actual changes between undo states*	Whereas actions described at the previous level are prescriptive, events at this level are descriptive. The specification of the change that was to occur has now been interpreted and executed. A description of an action at this level thus describes what a user's action has meant concretely, whether intended or not.

Table 1. Conceptualisation of the event levels of Blender interaction.

main specific language and to be exchanged easily between application written in different programming languages. The Thrift object is then placed in a queue. A separate thread connects to the data collection web service, identifies with the user's API key, and sends the objects from the queue. If a client is not able to realise a connection to the server, or if it loses an existing connection, it will retry at intervals of one second. New observations will continue to be placed in the queue if the connection to the server is down.

It was expected that participants would change from computer to computer regularly. Therefore each team was given its own API key and all members of the team used that key. As a result the recorded sessions can be linked to the team, but not to individual team members.

Data collection web service

The data collection web service is of a simple design. Its operation consists of a) accepting incoming connections from the Blender clients, b) verifying the API key against the user database, c) verifying integrity of the incoming message, and d) writing the message to an hourly rotating file on disk, prepended by a simple header. The messages are written in their original Thrift formats and are of variable size. The header that precedes every message is a constant 48 bytes and conveys the size and type of the subsequent message, the user from which it originates, and the client and server time stamps. The header design makes it possible to only decode the messages of interest (please see the data set web site[10] for a precise specification of the header format and the Thrift objects).

Screenshots received by the web service are written to disk as PNG files with their unique identifiers as the filenames. The corresponding operators then refer to these identifiers.

The asset manager

In line with a real studio the participants used an asset manager, custom built for the event. Accessed via a web interface it provides a list of assets with meta information, such as type (character, prop, set etc.), priority, who is assigned to create the asset and its current state. This allows the participants to organise themselves and work effectively, as in a real studio. Profile pictures were included, to help participants learn each others names. It also provides some error avoidance capabilities, as the participants make mistakes under pressure. This is primarily in terms of setting files up, in accordance with the asset type they are to contain. The render farm, which converts their files into a complete film, is integrated into this system.

It wrote a log file entry for each action that changed the meta data, such that the complete meta data should be recoverable from the log files alone. Each log entry includes date and time, the user and the team that performed the action, a function name indicating the action that is occurring and then all parameters. Filenames are given relative to the root directory of the given teams project. Short identifiers are typically used, for example 'char' instead of 'character' for file type.

Architecture

The architecture of all the technical components involved in the data collection is depicted in figure 1. All client machines ran Scientific Linux.

Limitations

The data collection is carried out in quite a particular setting and there are some things to be mindful of. Firstly, although many of the participants are talented sketch artists or animators, they are surely not yet at the level of the professional who is artistically and technically more developed. Secondly, we believe it quite uncommon that, in the 3D animation domain, adolescents work collaboratively in such an intensive manner. Therefore care should be taken in generalising findings from this dataset. Nonetheless, the work process is without doubt a creative one and thus suitable for investigating the questions set out in the introduction.

Another objection that can be made when attempting to generalise results is that the 3D animation domain is very technical (this is of course partly due to the design of the applica-

[10] See footnote 3

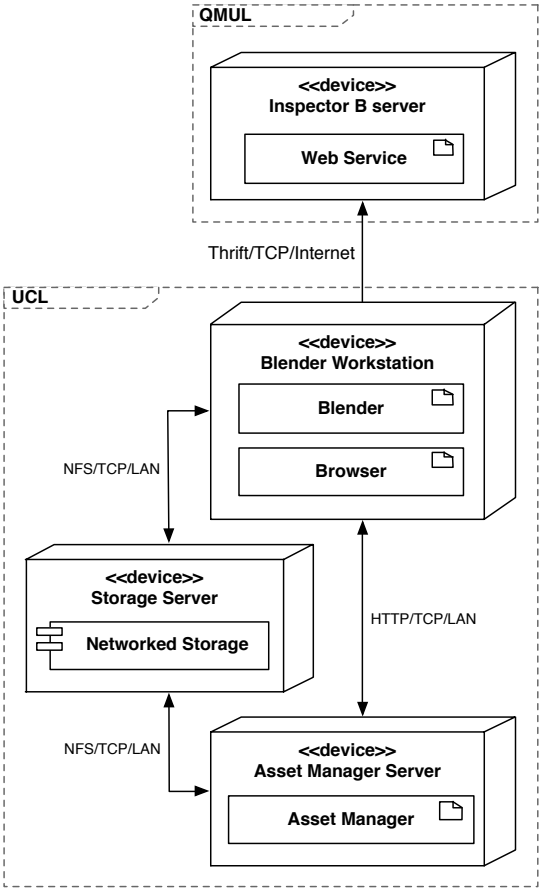


Figure 1. UML deployment diagram of the different technical components involved in the data collection.

tions). As such, we expect to see a higher rate of interaction whereby the tools are present-at-hand, as opposed to ready-to-hand, than might be the case in other domains [7].

Furthermore, considering the limited scope of the summer studio we cannot expect to collect many examples of all possible animation techniques. Neither will we see use of every possible Blender feature. However, the advantage of this scope is that it allows for the observation of practical creative processes from three complete, collaborative, and contained projects, which additionally we can observe with our own eyes and describe qualitatively.

The very nature of our investigation is to observe creative processes through instrumenting the tools that are employed in it. This restricts very much what we can know about the conscious considerations of the participants (something that is best studied by employing other methods, such as verbal reports [8]). Also, other tools that are employed by the participants, such as image editors or web browsers, are not instrumented. As such, although the aim is to observe completely the efforts of the participants in Blender, the data set should not be thought of as a complete observation of the entire creative process.

To remedy some of these limitations of the method we informally interview and observe participants throughout the summer studio. This allows us to contextualise the dataset qualitatively. In addition, at the end of event the participants complete a questionnaire. Findings from this are also presented in the *results* section.

RESULTS

In this section we first describe some of the qualitative observations made during the summer studio. The aim is to provide background information that should help in the interpretation of the collected usage data. We subsequently describe the data set in detail and present some analyses concerned with the completeness and reliability of the data.

Background information

As was scheduled the teams started story development in the afternoon of the first day, away from the computers. This process was facilitated by the tutors, who, for example, encouraged cycles of individual and group creativity. Participants communicated ideas verbally and gesturally as well as through the use of paper and pencil. Mobile phones were used to look up reference material, but only sparingly.

The Red and Green teams decided on a story and finished their storyboards with time to spare and started using Blender (this is reflected in figure 3). The Blue team spent more time developing their story and did not produce a storyboard until the afternoon of the second day.

On the morning of the second day it was decided that the Green team was to put to one side their original idea and pick up one of their earlier ideas. The original story idea involved a plane crash and was deemed inappropriate after three plane crashes had occurred in the week prior. As a consequence the Green team started working in Blender later than the other teams that day.

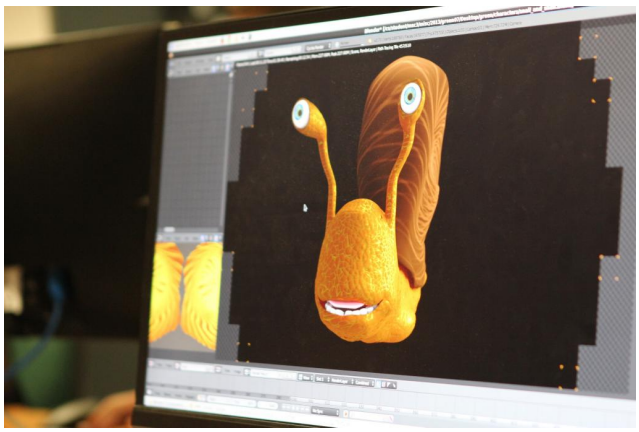


Figure 2. A member of the Green team renders one of the characters of their film 'Snail Fail'.

Each team was allocated a set of computers so that team members would be located as close to each other as possible. Although it was expected that participants would not necessarily use the same computer every day, they remained at the

computer they picked at the start of the event. However, participants regularly worked on other computers temporarily to help other team members. It is therefore possible that a session contains activity from more than one participant. There was very little collaboration between teams. Technical questions were more likely to be directed to the tutors than to the other teams.

The producers and directors that were voted in were typically the more confident participants with stronger film making skills. Producers were responsible for managing the assignments and progression of tasks, whereas directors were responsible for artistic direction. Because team members were packed together it was easy for members of a team to be aware of each other's activities. As such, discussions about an asset's aesthetic or technical properties or requests for help were frequent and short. To make sure the teams also engaged in more focused reflection and planning teams were asked to do 'daylies': short daily meetings with the entire team. The 3Dami tutors assisted the producers by making sure they, and other team members, were aware if the production was falling behind schedule.

Teams members were given, or took upon themselves, tasks that matched their experience and skills. As a consequence not all Blender features were used equally by all participants, or even by all teams. For example, not all participants within a team engaged in animation, and only one participant worked with Blender's Python scripting. Although it was expected that a division of labour would occur it is still worth pointing out that for some types of activities the data will be sparser than for others.

The assignments of tasks to team members was facilitated by the asset manager application. Excluding limited pre-event testing it was the first time the asset manager had been used. There were bugs and computational issues that were fixed during the event. The computational issues resulted in some periods of down time, during which the participants were unable to use it. In these periods they created files and managed their projects by hand. The Red team switched to paper at one point. In addition many of the participants did not use the asset manager, as they did not find it advantageous. Consequentially, it does not fully summarise the files of the event, so care must be taken when analysing the log files.

On the whole there were few technical problems. On occasion Blender would not load properly from the networked file system causing Blender to hang on launch. Participants would then forcefully quit the application and try again until it loaded properly. Also, Blender crashed on occasion, although it is difficult to know how often this happened.

Of the 26 participants 21 completed the post-event questionnaire. All participants indicated that they had enjoyed the summer studio, with 86% giving the highest score of 5 out of 5. When asked about what the best part of the event was the social elements of the event were often mentioned. This includes making new friends and having lunch together, but also collaborating with like-minded people. In this context the participants also often mentioned seeing the film come together and seeing it on the big screen at the premiere.

Participants however also said that the event had been stressful: 52% indicated the highest score for stress, while the rest gave a score of 3 or 4 out of 5. When asked about the least enjoyable part of the event many participants indicated that it was the stress, specifically of possibly not finishing in time. Also often mentioned were technical problems like issues with rendering. An interesting finding is that five participants mentioned that the story development had been their least enjoyable activity of the week. One participant disliked the arguing about the story development, but the others did not elaborate.

At the end of the summer studio all teams had finished their films in time for the premiere, albeit with opportunities for polishing[11]. The Red team's story 'Baby Rumble' centres on two babies who get embroiled in an epic fight as soon as their mother leaves the house. The Blue team's story is a cynical interpretation of a typical London tube journey. Finally, the Green team tells a tale of an unfortunate underestimation of a pesky snail.

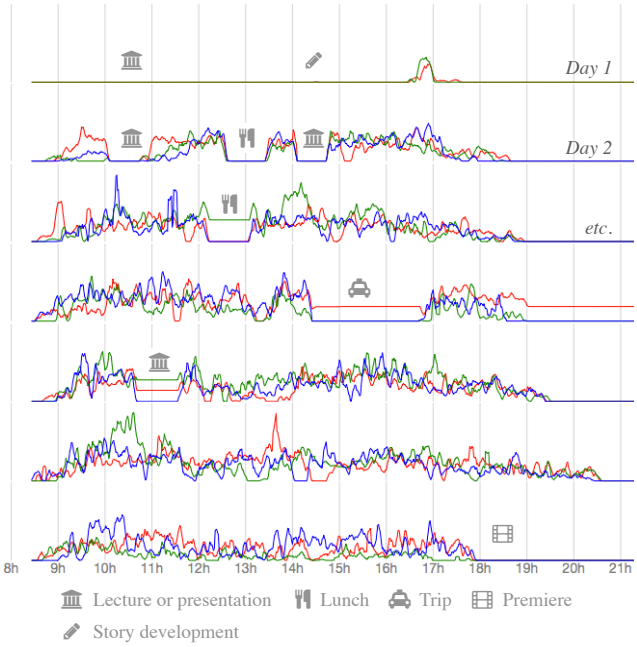


Figure 3. The general activity per team per minute over the entire 3Dami event measured by observation count. The line colours correspond to the team names.

The data set

Throughout the summer studio we recorded over 11 million observations (see table 2 for specifics), weighing in at approximately 19 gigabytes. In addition we recorded just over half a million screenshots, at 31 gigabytes. The screenshots were taken at a maximum resolution of 640 by 251 pixels, a sixteenth of the original screen area. The asset manager logs

[11]The films are available for watching online: `https://www.youtube.com/watch?v=5ZArdX9fLG0`

Observation type	Count
Session starts	1630
Session ends	1318
Button presses	149,665
Device input (mouse movements)	2,005,578
Device input (other events)	3,355,064
Assignments	57,808
Operators	5,546,703

Table 2. Observations counts.

comprise approximately 2900 events. The three teams combined produced 499 Blender project files, with all project files and assets coming in at 130 gigabytes.

Sessions

In total 1665 unique session keys were recorded (772 for the Red team, 434 for Green, 459 for Blue). The mean session duration is 52 minutes with a total of 1,439 hours for all sessions combined. It is worth stating again that a session is defined as being comprised of all the interactions that occur in a single instance of the execution of the application. As such, many sessions include substantial intervals of inactivity. If we do not include any period of inactivity of over 2 minutes the total duration comes down to 610 hours of use.

Many sessions consist of very few observations while a minority of sessions account for most of the data collected. In figure 4 we plot a histogram of the session observation count. The same figure shows a function of the bin number that represents what portion the observations are accounted for by the sessions of that bin, and previous bins. By definition a single short session does not contribute to the total number of observation as much as a longer session. However, a session is more likely to be short than long and therefore all short sessions combined could contribute significantly to the total observation count. However, in figure 4 we can clearly see that this is not the case.

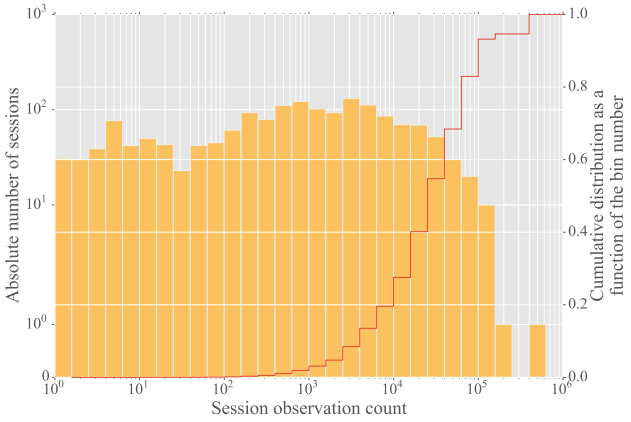


Figure 4. The cumulative distribution of observations plotted against the distribution of session size.

The reader might have noticed a curious inconsistency in the session start and end marker counts in table 2, whereby we count 1665 unique session keys, 1630 session starts, and 1318 session ends. In other words, 2.1% of the sessions are not marked with a start, 20.1% are not marked with an end, and 78.9% of sessions contain both a start and an end marker.

Of the 316 sessions that were marked *only* with a start 33% had a last observation with a time stamp near the end of the day when participants started to leave the venue. When participants left they typically did not shut down the computers, in some cases to leave them rendering shots. The computers were however forcefully shut down every day at midnight. In the case of a forceful shutdown the last observation would have been the last interaction before the participants left, but an end marker would not have been observed. We thus suspect that this shutdown procedure is the cause for a large part of the sessions without an end marker.

It is tempting to think that application crashes are the cause of the rest of these sessions. Although crashes were seen to occur occasionally we do not know what percentage of incomplete sessions can be attributed to them. For both causes (forceful shutdowns and crashes) we can be certain that we will have lost only a small amount of interaction data, if any.

Four sessions were marked with *only* an end, and 31 sessions did not contain any start or end markers. All of these 35 sessions are extremely short, consisting of only 1 to 3 observations. This suggests that these sessions correspond to the problems with starting the application at the start of each day.

Although the sessions where a marker is missing are problematic, we can be quite certain that the 1314 sessions with both a start and an end marker were observed completely.

Network stability

Because the Blender clients sent their observations to the data collection server over the internet there was the non-neglible possibility of connection loss. We investigated the difference between the observations' client and server timestamps as a manner of checking for network problems and therefore possible data loss. That is, if a disconnect had happened there would have been a sudden increase in the difference between the server and client timestamps, for an observation would remain in the queue locally and arrive at the server only after the connection was restored. We have not found any such sudden increases and do therefore conclude that no connection loss of any significant duration occurred.

Another potential network problem that could have caused data loss is network congestion. If there was congestion then a client's message queue would have filled up and data loss would have been significant in the event of a crash. In the case of congestion we would expect the gap between the client and the server timestamp to gradually increase. We have also not observed any such gradual increases, and we thus conclude that there were no bandwidth issues. In addition, no network issues were observed in the server monitoring system.

There were however sudden increases in differences between the client and server timestamp that happened in a very particular case: right after a period of inactivity in a session no shorter than 75 minutes. There are 55 sessions where the difference jumps to just over 15 minutes. It is our hypothesis

that this is caused by the computers automatically going to sleep after a certain period of inactivity and that this offsets the clocks. After wake-up the client timestamps on these machines gradually realign themselves with the server suggesting that the Network Time Protocol[12] service did its job. Any analysis of the data that relies on timing of events should take this phenomenon into account.

Figure 3 shows the activity of the three teams over the course of the event. Activity is measured in observation count per minute, smoothed by a 5 minute moving average (time is taken from server timestamps). Intervals where activity is absent or where there is a constant rate of activity (e.g. a participant has left an animation playing in a loop) all coincide with specific events such as lunch time or lectures.

Preparation for publication

The collected data in its original form was not fit for publication. It was prepared in the following manner.

Firstly, the hourly data files written to disk by the web service are split into session files. Each session file thus contains only observations from a single session. This aids visualisation of a single session of interest as well as the parallelisation of computational analysis.

Secondly, some of the screenshots were found to depict browser sessions or copyrighted reference material. The screen grabbing code in Blender was thought to only be able to record the OpenGL framebuffers written to by Blender. However, it happens that at times the captured screenshots show other applications such as the browser. Therefore it was necessary to remove the screenshots that depicted privacy sensitive information or other content not fit for publication. It might thus be that an 'operator' observation links to an absent screenshot. The redaction of screenshots was a manual process.

Thirdly, the asset manager logs contained the real names of the participants. In the published logs these names have been replaced with randomly chosen names. The new names still reflect the correct gender of the participants.

Types of operators and activity

To demonstrate a simple use of the dataset we have, in figure 5, plotted the distribution of operator invocations over the course of the summer studio. Each row shows all the operator invocations of a certain type and what proportion of them occurred on which day. We define an operator's type to be the Blender module to which it belongs, and types are thus determined by Blender's design. For example, all operators that operate on a *mesh* are defined in the *Mesh* module. In this way we can roughly group operators that are likely to be used in conjunction in a certain activity. The height of each row in figure 5 is proportionate to the prevalence, over all days, of that row's operator type compared to other types. Types are sorted from top to bottom by the importance of the last two days of the summer studio. That is, operator types that were observed more frequently in the last two days (these are marked in shades of blue) than in the first five (marked in shades of orange) are nearer the end.

Whereas figure 3 showed general levels of activity, from figure 5 we can get an impression of the types of activity per day and how this developed over the course of the summer studio. For example, the types of operators near the top (i.e. curve, mesh, sculpt, object, and UV) are related to modelling activities, such as creating scenes, props, and characters. Operator types near the middle (i.e. armature, anim, paint, image, and pose) are associated with activities such as texturing and animation. Near the bottom we find operator types (i.e. pose, graph, action, and sequencer) that were mostly used near the end of the week. These are associated with animation and sequencing separate shots into the final film. This matches the production process as was explained in the schedule and observed during the event. Thus, we could perhaps predict, with a reasonable degree of accuracy, to which stage of the film making process a session belongs from the types of operators observed during that session.

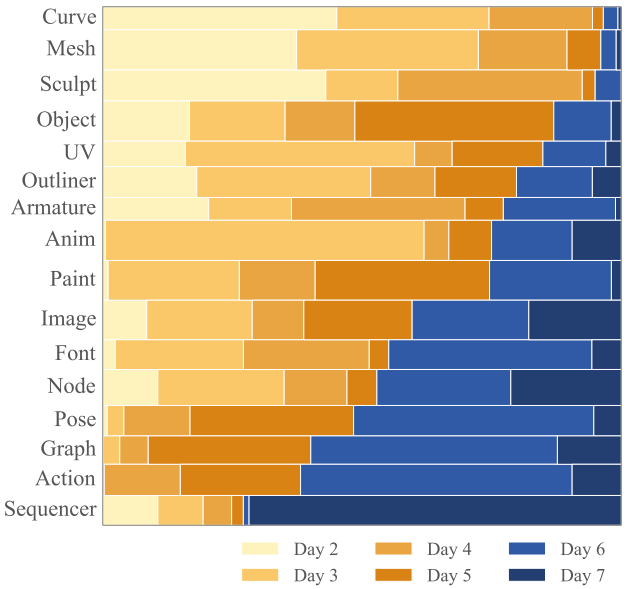


Figure 5. The distribution of operator observations over the course of the summer studio per operator type.

DISCUSSION

We have already outlined that the 3Dami setting is not a typical setting. However, the participants were skilled and motivated and able to conceive and complete a complex collaborative effort in a mature fashion. The process was guided by the tutors so as to resemble the typical animation production process. Furthermore, although the films they produced are not of a professional quality they are certainly creative artefacts. Their practical creative processes might not be representative of the more common processes of the individual amateur or the professional team, but there should be more similarities than differences.

The technical architecture proved to be reliable, but could be extended to provide better guarantees of completeness.

[12] `http://www.ntp.org/`

For example, the percentage of sessions without end markers was surprising. Some simple form of crash reporting could have provided a helping hand in determining the reason for the absence of these markers. Also, to prevent data loss in the event of a crash the observation queue could be run as a separate application. This would however have increased the complexity of the deployment, which would not be suitable for many managed classroom environments. Furthermore, because each participant largely used the same computer throughout the week it would have been possible to have every participant use a separate API key. That way we could have attributed each session to an individual participant more easily. Other more involved solutions to this problem would in hindsight have been worthwhile (e.g. the use of face recognition or Near Field Communication technology), albeit also difficult to deploy in certain environments.

Finally, the resulting dataset is of limited scope, but is sizeable enough for many uses. Due to the breadth of Blender and the large variety of film making techniques this means that for some activities or workflows the data will be sparse. However, the advantage of this limited scope is that the summer studio could be well documented and that we were able to provide a qualitative description of the context in which the data was collected. Furthermore, the project files and asset manager logs should provide further context and opportunities for analysis.

Although there is a possibility that we have not recorded all Blender interaction, we have shown with the analyses of timestamps, session markers, and activity over the course of the summer studio, that if data was lost it could not have accounted for too great a proportion. However, it is difficult to argue further about the possible absence of events we have not observed.

In sum, the data set was produced in a reliable manner, describes a substantive set of practical creative processes, and is well documented. It should thus enable us to investigate further the questions we have posed in the introduction.

CONCLUSION

We have presented in this work an open data set describing the practical creative processes of the 26 participants of the 3Dami summer studio 2014. The data set consists of a) usage interaction events of the participants' interaction with Blender, conceptualised within a multi-level framework, b) screenshots depicting the Blender state at each moment of interaction, c) asset manager logs that describe the overall production process of each team, and d) all the project files produced by the participants.

Furthermore, we have shown that the data set was produced in a reliable manner and have documented the events of the 3Dami summer studio in order to provide background information to aid in the interpretation of the collected data. Finally, the data set is published under an open license.

FUTURE WORK

The reason for collecting the 3Dami data set was so that we, and hopefully others, can begin to investigate how an artist's tools can be instrumented so that they not only enable the creation of an artefact, but are also able to describe how that artefact was created. This, in our view, comes down to inferring, from quantitative observations, a more human narrative, such as the qualitative observations in this work. Our future work will thus focus on the second and third part of the question set out in the introduction: how to understand and narrate the 3Dami data set.

Neither the questions we have posed nor the approach we have taken are specific to practical creative processes in Blender, or to the domain of animation for that matter. It would be interesting to see if the instrumentation of, for example, a music sequencer requires a different framing of the problem.

ACKNOWLEDGMENTS

Support from the Media and Arts Technology Programme: An RCUK Doctoral Training Centre in the Digital Economy. University College London provided funding, space and computation for the 3Dami summer studio; NESTA provided bursaries. Volunteers provided their time to run it. We express our thanks to the Blender Foundation and all Blender developers.

The first author planned and implemented the data collection study. The second and third authors organised and ran the 3Dami summer studio. The last two authors supervised the study.

REFERENCES

1. Akers, D. L., Simpson, M., Jeffries, R., and Winograd, T. Undo and erase events as indicators of usability problems. In *Proc. CHI 2009*, ACM (2009), 659–668.
2. Boden, M. A. *The creative mind: myths and mechanisms*. Routledge, London, England, 2004.
3. Candy, L., and Edmonds, E. Supporting the creative user: a criteria-based approach to interaction design. *Design Studies 18*, 2 (1997), 185–194.
4. Clark, A. *Supersizing the Mind: Embodiment, Action, and Cognitive Extension*. Oxford University Press, New York, 2008.
5. Csikzentmihalyi, M. *Creativity: Flow and the Psychology of Discovery and Invention*. HarperPerennial, New York, 1997.
6. Darke, J. The primary generator and the design process. *Design Studies 1*, 1 (1979), 36–44.
7. Dourish, P. *Where the action is: the foundations of embodied interaction*. The MIT Press, Cambridge, Massachusetts, 2004.
8. Ericsson, K. A., and Simon, H. A. *Protocol Analysis*, revised ed. The MIT Press, Cambridge, Massachusetts, 1993.
9. Goldschmidt, G. The backtalk of self-generated sketches. *Design Issues 19*, 1 (2003), 72–88.

10. Grossman, T., Matejka, J., and Fitzmaurice, G. Chronicle: capture, exploration, and playback of document workflow histories. In *Proc. UIST 2010*, ACM (2010).

11. Hagberg, G. L. *Art as Language: Wittgenstein, meaning, and aesthetic theory*. Cornell University Press, 1998.

12. Hewett, T. T. Informing the design of computer-based environments to support creativity. *International Journal of Human-Computer Studies 63*, 4 (2005), 383–409.

13. Hilbert, D. M., and Redmiles, D. F. Extracting usability information from user interface events. *ACM Computing Surveys (CSUR) 32*, 4 (2000), 384–421.

14. Hillier, B., Musgrove, J., and O'Sullivan, P. Knowledge and design. In *Environmental Design Research and Practice*, J. W. Mitchell, Ed. University of California Press, Los Angeles, 1972.

15. Kirsh, D., and Maglio, P. On distinguishing epistemic from pragmatic action. *Cognitive science 18*, 4 (1994), 513–549.

16. Lafreniere, B., Grossman, T., Matejka, J., and Fitzmaurice, G. Investigating the feasibility of extracting tool demonstrations from in-situ video content. In *Proc. CHI 2014*, ACM (2014), 4007–4016.

17. Lawson, B. *How designers think: the design process demystified*, 4th ed. Routledge, 2006.

18. Li, W., Matejka, J., Grossman, T., Konstan, J. A., and Fitzmaurice, G. Design and evaluation of a command recommendation system for software applications. *ACM Transactions on Computer-Human Interaction (TOCHI) 18*, 2 (2011).

19. Matejka, J., Li, W., Grossman, T., and Fitzmaurice, G. Communitycommands: command recommendations for software applications. In *Proc. UIST 2009*, ACM (2009), 193–202.

20. Mendels, P., Frens, J., and Overbeeke, K. Freed: a system for creating multiple views of a digital collection during the design process. In *Proc. CHI 2011*, ACM (2011), 1481–1490.

21. Nabavian, S., and Bryan-Kinns, N. Analysing group creativity: A distributed cognitive study of joint music composition. In *Proc. of Cognitive Science* (2006), 1856–1861.

22. Papert, S., and Harel, I. Situating constructionism. In *Constructionism*. Ablex Publishing Corporation, 1991.

23. Plaisant, C., Rose, A., Rubloff, G., Salter, R., and Shneiderman, B. The design of history mechanisms and their use in collaborative educational simulations. In *Proc. CSCL 1999*, International Society of the Learning Sciences (1999).

24. Plucker, J. A., and Renzulli, J. S. Psychometric approaches to the study of human creativity. In *Handbook of Creativity*. 1999.

25. Purcell, A. T., and Gero, J. S. Drawings and the design process: A review of protocol studies in design and other disciplines and related research in cognitive psychology. *Design Studies 19*, 4 (1998), 389–430.

26. Sawyer, R. K. *Group Creativity: Music, Theater, Collaboration*. Music, Theater, Collaboration. Lawrence Erlbaum Associates, 2003.

27. Schön, D. A. *The Reflective Practitioner: How Professionals Think In Action*. Ashgate Publishing Limited, London, England, 1991.

28. Schön, D. A. Designing as reflective conversation with the materials of a design situation. *Knowledge-based systems 5*, 1 (1992), 3–14.

29. Shneiderman, B. Creating creativity: user interfaces for supporting innovation. *ACM Transactions on Computer-Human Interaction (TOCHI) 7*, 1 (2000), 114–138.

30. Sternberg, R. J., Ed. *Handbook of Creativity*. Cambridge University Press, 1999.

31. Terry, M. A., and Mynatt, E. D. Recognizing creative needs in user interface design. In *Proc. Creativity & Cognition 2002*, ACM (2002), 38–44.

32. White, A. R., Ed. *The Philosophy of Action*. Oxford University Press, 1968.

33. Wiggins, G. A. The mind's chorus: Creativity before consciousness. *Cognitive Computation 4*, 3 (2012), 306–319.

Bibliography

Abramsky, Samson and Nikos Tzevelekos (2011). "Introduction to Categories and Categorical Logic". In: DOI: 10.1007/978-3-642-12821-9_1. eprint: arXiv:1102.1313.

Agres, Kat, Jamie Forth, and Geraint A Wiggins (2016). "Evaluation of musical creativity and musical metacreation systems". In: *Computers in Entertainment (CIE)* 14.3, p. 3.

Akers, David Light et al. (2009). "Undo and erase events as indicators of usability problems". In: *Proc. CHI 2009*. ACM, pp. 659–668.

Akkermans, Vincent and Than van Nispen Tot Pannerden (2008). "TWO NETWORK INSTALLATIONS : '1133' & 'COMPUTER VOICES'". In: *Proceedings of the 2008 International Computer Music Conference*.

Allen, James F (1983). "Maintaining knowledge about temporal intervals". In: *Communications of the ACM* 26.11, pp. 832–843.

— (1984). "Towards a general theory of action and time". In: *Artificial intelligence* 23.2, pp. 123–154.

Baars, Bernard J (1988). *A Cognitive Theory of Consciousness*. English. Cambridge University Press. ISBN: 9780521301336.

Baars, Bernard J and Nicole M Gage (2010). *Cognition, Brain, and Consciousness*. English. Introduction to Cognitive Neuroscience. Academic Press. ISBN: 9780123814401.

Baber, Chris (2015). "Thinking Through Tools: What Can Tool-Use Tell Us About Distributed Cognition?" In: *Studies in Logic, Grammar and Rhetoric* 41.1, pp. 25–40.

Baber, Chris, Tulin Gunduz Cengiz, and Manish Parekh (2014). "Tool use as distributed cognition: how tools help, hinder and define manual skill". In: *Frontiers in psychology* 5, p. 116.

Baber, Chris, Tony Chemero, and Jamie Hall (2017). "What the Jeweller's Hand Tells the Jeweller's Brain: Tool Use, Creativity and Embodied Cognition". In: *Philosophy & Technology*, pp. 1–20.

Beek, P van and D W Manchak (1996). "The Design and Experimental Analysis of Algorithms for Temporal Reasoning". In: *arXiv.org*. arXiv: cs/9601101v1 [cs.AI].

Bengio, Yoshua (2009). "Learning deep architectures for AI". In: *Foundations and trends® in Machine Learning* 2.1, pp. 1–127. DOI: 10.1561/2200000006.

Benwell, B (1999). *The organisation of knowledge in British university tutorial discourse: Issues, pedagogic discourse strategies and disciplinary identity*. Pragmatics.

Boden, Margaret A (2004). *The creative mind: myths and mechanisms*. London, England: Routledge. ISBN: 0-203-50852-1.

— (2006). *Mind As Machine*. English. A History of Cognitive Science. Oxford University Press. ISBN: 9780199292387.

Bono, Edward de (1990). *Lateral Thinking*. English. Creativity Step by Step. Harper & Row Publishers. ISBN: 0-06-090325-2.

Brown, John Seely, Allan Collins, and Paul Duguid (1989). "Situated cognition and the culture of learning". In: *Educational researcher* 18.1, pp. 32–42.

Candy, Linda and E A Edmonds (1994). *Artefacts and the designer's process: implications for computer support to design*. Journal of Design Sciences and Technology.

Candy, Linda and Ernest Edmonds (1997). "Supporting the creative user: a criteria-based approach to interaction design". In: *Design Studies* 18.2, pp. 185–194.

Charmaz, Kathy (2006). *Constructing Grounded Theory*. English. A Practical Guide Through Qualitative Analysis. SAGE. ISBN: 9780761973539.

Chemero, Anthony (2011). *Radical Embodied Cognitive Science*. English. MIT Press. ISBN: 9780262258081.

Chen, Hsiang-Ting, Li-Yi Wei, and Chun-Fa Chang (2011). "Nonlinear revision control for images". In: *ACM SIGGRAPH 2011 papers*. New York, New York, USA: ACM Press, p. 1. ISBN: 9781450309431. DOI: 10.1145/1964921.1965000.

Clark, Andy (1997). *Being there*. MIT Press Cambridge, MA.

— (2008). *Supersizing the Mind: Embodiment, Action, and Cognitive Extension*. New York: Oxford University Press. ISBN: 019971553X.

— (2013). "Whatever next? Predictive brains, situated agents, and the future of cognitive science". In: *Behavioral and Brain Sciences*. DOI: 10.1017/S0140525X12000477.

— (2015a). "Embodied Prediction". In: *Open MIND*. Ed. by Thomas K. Metzinger and Jennifer M. Windt. Frankfurt am Main: MIND Group. Chap. 7(T). ISBN: 9783958570115. DOI: 10.15502/9783958570115. URL: https://open-mind.net/papers/embodied-prediction.

— (2015b). "Radical predictive processing". In: *The Southern Journal of Philosophy* 53.S1, pp. 3–27.

— (2015c). *Surfing uncertainty: Prediction, action, and the embodied mind*. Oxford University Press.

— (2017a). "A nice surprise? Predictive processing and the active pursuit of novelty". In: *Phenomenology and the Cognitive Sciences*, pp. 1–14.

— (2017b). "Busting out: Predictive brains, embodied minds, and the puzzle of the evidentiary veil". In: *Noûs* 51.4, pp. 727–753.

Clark, Andy and David Chalmers (1998). "The extended mind". In: *analysis* 58.1, pp. 7–19.

Cohen, Harold (1995). "The further exploits of AARON, painter". In: *Stanford Humanities Review* 4.2, pp. 141–158.

Colton, S and Geraint A Wiggins (2012). "Computational Creativity: The Final Frontier?" In: DOI: 10.3233/978-1-61499-098-7-21.

Colton, Simon (2012). "The painting fool: Stories from building an automated painter". In: *Computers and creativity*. Springer, pp. 3–38.

Cross, Nigel (2011). *Design Thinking*. English. Understanding How Designers Think and Work. Berg. ISBN: 9781847888464.

Csikzentmihalyi, Mihaly (1997). *Creativity: Flow and the Psychology of Discovery and Invention*. New York: HarperPerennial. ISBN: 0-06-092820-4.

Darke, Jane (1979). "The primary generator and the design process". In: *Design Studies* 1.1, pp. 36–44.

Denning, Jonathan D, William B Kerr, and Fabio Pellacini (2011). "MeshFlow: interactive visualization of mesh construction sequences". In: *SIGGRAPH '11: SIGGRAPH 2011 papers*. DOI: 10.1145/1964921.1964961.

Duan, Yan et al. (2016). "RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning". In: *arXiv preprint arXiv:1611.02779*.

Fodor, Jerry A and Zenon W Pylyshyn (1988). "Connectionism and cognitive architecture: A critical analysis". In: *Cognition* 28.1-2, pp. 3–71.

Forth, Jamie, Geraint A Wiggins, and Alex McLean (2010). "Unifying Conceptual Spaces: Concept Formation in Musical Creative Systems". English. In: *Minds and Machines* 20.4, pp. 503–532. DOI: 10.1007/s11023-010-9207-x.

Furniss, Dominic, Ann Blandford, and Paul Curzon (2011). "Confessions from a grounded theory PhD: experiences and lessons learnt". In: *CHI '11: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. DOI: 10.1145/1978942.1978960.

Gärdenfors, Peter (2004). *Conceptual Spaces*. English. The Geometry of Thought. MIT Press. ISBN: 9780262572194.

Gardner, Howard (1993). *Creating Minds*. English. Basic Books. ISBN: 9780465027866.

Gedenryd, H (1998). "How designers work: Making sense of authentic cognitive activity". PhD thesis. Lund University Cognitive Studies.

Ghallab, M, D Nau, and P Traverso (2004). *Automated planning: theory & practice*.

Glaser, Barney G (2008). *Doing Quantitative Grounded Theory*. English.

Glaser, Barney G and Anselm L Strauss (1967). *The Discovery of Grounded Theory*. English. Strategies for Qualitative Research. Aldine Transaction. ISBN: 9780202363370.

Gobet, Fernand (2005). "Chunking models of expertise: Implications for education". In: *Applied Cognitive Psychology* 19.2, pp. 183–204. DOI: 10.1002/acp.1110.

Gobet, F et al. (2001). "Chunking mechanisms in human learning". In: *Trends in cognitive sciences* 5.6, pp. 236–243.

Goldberg, D.E. (2006). *Genetic Algorithms*. Pearson Education. ISBN: 9788177588293.

Goldschmidt, G (1991). "The dialectics of sketching". English. In: *Creativity research journal* 4.2, pp. 123–143. DOI: 10.1080/10400419109534381.

— (2003). "The backtalk of self-generated sketches". In: *Design Issues* 19.1, pp. 72–88.

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep Learning*. Adaptive Computation and Machine Learning Series. MIT Press. ISBN: 9780262035613.

Grossman, Tovi, Justin Matejka, and George Fitzmaurice (2010). "Chronicle: capture, exploration, and playback of document workflow histories". In: *Proc. UIST 2010*. ACM.

Heer, Jeffrey et al. (2008). "Graphical Histories for Visualization: Supporting Analysis, Communication, and Evaluation". In: *Visualization and Computer Graphics, IEEE Transactions on* 14.6, pp. 1189–1196.

Hewett, Tom et al. (2005). "Creativity support tool evaluation methods and metrics". In: *Creativity Support Tools, A workshop sponsored by the National Science Foundation*, pp. 10–24.

Hillier, B, J Musgrove, and P O'Sullivan (1972). "Knowledge and design". In: *Environmental Design Research and Practice*. Ed. by J W Mitchell. Los Angeles: University of California Press.

Hutchins, Edwin (2008). "The role of cultural practices in the emergence of modern human intelligence". In: *Philosophical Transactions of the Royal Society of London B: Biological Sciences* 363.1499, pp. 2011–2019. ISSN: 0962-8436.

— (2014). "The cultural ecosystem of human cognition". In: *Philosophical Psychology* 27.1, pp. 34–49.

Jordanous, Anna (2012). "A standardised procedure for evaluating creative systems: Computational creativity evaluation based on what it is to be creative". In: *Cognitive Computation* 4.3, pp. 246–279.

Kantosalo, Anna Aurora, Jukka Mikael Toivanen, Hannu Tauno Tapani Toivonen, et al. (2015). "Interaction Evaluation for Human-Computer Co-Creativity". In: *Proceedings of the Sixth International Conference on Computational Creativity*.

Kantosalo, Anna et al. (2014). "From Isolation to Involvement: Adapting Machine Creativity Software to Support Human-Computer Co-Creation." In: *ICCC*, pp. 1–7.

Karmiloff-Smith, Annette (1992). *Beyond Modularity*. English. Adevelopmental Perspective on Cognitive Science. The MIT Press. ISBN: 9780262611145.

Kaufman, James C and Ronald A Beghetto (2009). "Beyond big and little: The four c model of creativity." English. In: *Review of General Psychology* 13.1, pp. 1–12. DOI: 10.1037/a0013688.

Kaufman, James C and Robert J Sternberg (2010). *The Cambridge Handbook of Creativity*. English. Cambridge University Press. ISBN: 9780521513661.

Kendon, A (2004). *Gesture: Visible action as utterance*.

Kingma, D. P. and J. Ba (2014). "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980. arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980.

Kingma, D. P. and M. Welling (2013). "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114*.

Kirsh, David and Paul Maglio (1994). "On distinguishing epistemic from pragmatic action". In: *Cognitive science* 18.4, pp. 513–549.

Klemmer, Scott R et al. (2002). "Where do web sites come from?: capturing and interacting with design history". In: pp. 1–8.

Klyubin, Alexander S, Daniel Polani, and Chrystopher L Nehaniv (2005). "Empowerment: A universal agent-centric measure of control". In: *Evolutionary Computation, 2005. The 2005 IEEE Congress on*. Vol. 1. IEEE, pp. 128–135.

Kong, Nicholas et al. (2012). "Delta: a tool for representing and comparing workflows". In: *CHI '12: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM Request Permissions. DOI: 10.1145/2207676.2208549.

Kowalski, Robert and Marek Sergot (1989). "A logic-based calculus of events". In: *Foundations of knowledge base management*. Springer, pp. 23–55.

Kranstedt, A, A Lücking, and T Pfeiffer (2006). "Deictic object reference in task-oriented dialogue". In: *TRENDS IN …*.

Lawson, Bryan (2006). *How designers think: the design process demystified*. 4th. Routledge.

— (2012). *What Designers Know*. English. Routledge. ISBN: 1136349006.

Li, Wei et al. (2011). "Design and evaluation of a command recommendation system for software applications". In: *ACM Transactions on Computer-Human Interaction (TOCHI)* 18.2.

Malinin, Laura H (2016). "Creative practices embodied, embedded, and enacted in architectural settings: toward an ecological model of creativity". In: *Frontiers in psychology* 6, p. 1978.

Mann and Thompson (1988). "Rhetorical structure theory: Toward a functional theory of text organization". In: *Text* 8.3, pp. 243–281.

March, Lionel (1984). "The logic of design". English. In: *Developments in Design Methodology*, pp. 265–276.

Matejka, Justin et al. (2009). "CommunityCommands: command recommendations for software applications". In: *Proc. UIST 2009*. ACM, pp. 193–202.

McNeill, David (2005). *Gesture and Thought*. Chicago: The University of Chicago Press. ISBN: 0-226-51462-5.

Menary, R. (2010). *The Extended Mind*. Bradford Books. Bradford Books. ISBN: 9780262014038. URL: https://books.google.co.uk/books?id=DxQNhOS8VjQC.

Mendels, Philip, Joep Frens, and Kees Overbeeke (2011). "Freed: a system for creating multiple views of a digital collection during the design process". In: *Proc. CHI 2011*. ACM, pp. 1481–1490.

Mueller, Erik T (2006). *Commonsense Reasoning*. English. Morgan Kaufmann. ISBN: 9780080476612.

Mueller, E.T. (1990). *Daydreaming in Humans and Machines: A Computer Model of the Stream of Thought*. Ablex. ISBN: 9780893915629.

Nabavian, Shahin and Nick Bryan-Kinns (2006). "Analysing Group Creativity: A Distributed Cognitive Study of Joint Music Composition". In: *Proc. of Cognitive Science*, pp. 1856–1861.

Nakamura, Toshio and Takeo Igarashi (2008). "An application-independent system for visualizing user operation history". In: *Proceedings of the 21st annual ACM symposium on User interface software and technology*. ACM, pp. 23–32. ISBN: 159593975X.

Oudeyer, Pierre-Yves and Frederic Kaplan (2009). "What is intrinsic motivation? A typology of computational approaches". In: *Frontiers in Neurorobotics* 1, p. 6. ISSN: 1662-5218. DOI: 10.3389/neuro.12.006.2007. URL: https://www.frontiersin.org/article/10.3389/neuro.12.006.2007.

Oudeyer, Pierre-Yves, Frederic Kaplan, and Verena V Hafner (2007). "Intrinsic motivation systems for autonomous mental development". In: *IEEE transactions on evolutionary computation* 11.2, pp. 265–286.

Pascanu, Razvan et al. (2017). "Learning model-based planning from scratch". In: *arXiv preprint arXiv:1707.06170*.

Pérez y Pérez, Rafael (1999). "MEXICA". English. PhD thesis.

Plaisant, Catherine et al. (1999). "The Design of History Mechanisms and Their Use in Collaborative Educational Simulations". In: *Proc. CSCL 1999*. International Society of the Learning Sciences.

Plucker, Jonathan A and Joseph S. Renzulli (1999). "Psychometric Approaches to the Study of Human Creativity". In: *Handbook of Creativity*. ISBN: 9780521576048.

Polani, Daniel (2009). "Information: Currency of life?" In: *HFSP Journal* 3.5. PMID: 20357888, pp. 307–316. DOI: 10.2976/1.3171566.

Popper, Karl (1959). "The logic of scientific discovery". In:

Purcell, A T and J S Gero (1998). "Drawings and the design process: A review of protocol studies in design and other disciplines and related research in cognitive psychology". English. In: *Design Studies* 19.4, pp. 389–430. DOI: 10.1016/S0142-694X(98)00015-5.

Raskin, Jef (2000). *The Humane Interface*. English. New Directions for Designing Interactive Systems. Addison-Wesley Professional. ISBN: 9780201379372.

Reichertz, J (2009). "Abduction: The Logic of Discovery of Grounded Theory". English. In: *Forum Qualitative Sozialforschung / Forum: Qualitative Social Research* 11.1.

Ritchie, Graeme (2007). "Some empirical criteria for attributing creativity to a computer program". In: *Minds and Machines* 17.1, pp. 67–99.

Rowlands, Mark (2010). *The new science of the mind*.

Salge, Christoph, Cornelius Glackin, and Daniel Polani (2014). "Empowerment–An Introduction". In: *Guided Self-Organization: Inception*. Springer, pp. 67–114.

Sawyer, Robert Keith (2003). *Group Creativity: Music, Theater, Collaboration*. Music, Theater, Collaboration. Lawrence Erlbaum Associates. ISBN: 9780805844368.

Schmidhuber, Jürgen (2006). "Developmental robotics, optimal artificial curiosity, creativity, music, and the fine arts". In: *Connection Science* 18.2, pp. 173–187. DOI: 10.1080/09540090600768658. eprint: https://doi.org/10.1080/09540090600768658. URL: https://doi.org/10.1080/09540090600768658.

Schmidhuber, Jürgen (2007). “Simple Algorithmic Principles of Discovery, Subjective Beauty, Selective Attention, Curiosity & Creativity”. In: *CoRR* abs/0709.0674.

— (2014). “Deep Learning in Neural Networks: An Overview”. In: *CoRR* abs/1404.7828.

Schön, Donald A (1991). *The Reflective Practitioner: How Professionals Think In Action*. English. London, England: Ashgate Publishing Limited. ISBN: 978-0-85742-319-8.

— (1992). “Designing as reflective conversation with the materials of a design situation”. In: *Knowledge-based systems* 5.1, pp. 3–14.

Shannon, Claude E (1938). “A symbolic analysis of relay and switching circuits”. In: *Electrical Engineering* 57.12, pp. 713–723.

Shannon, Claude Elwood and Warren Weaver (1948). “A mathematical theory of communication”. In: DOI: 10.1109/9780470544242.ch1.

Sharples, Mike (1999). *How We Write*. Writing as creative design. Routledge. ISBN: 0-415-18586-6.

Shneiderman, Ben (2000). “Creating creativity: user interfaces for supporting innovation”. In: *ACM Transactions on Computer-Human Interaction (TOCHI)* 7.1, pp. 114–138.

Shneiderman, Ben et al. (2006). “Creativity support tools: Report from a US National Science Foundation sponsored workshop”. In: *International Journal of Human-Computer Interaction* 20.2, pp. 61–77.

Silver, David et al. (2016). “Mastering the game of Go with deep neural networks and tree search”. In: *nature* 529.7587, pp. 484–489.

Simon, Herbert A (1977). “The structure of ill-structured problems”. In: *Models of discovery*. Springer, pp. 304–325.

Spivak, David I (2014). *Category theory for the sciences*. MIT Press.

Sternberg, Robert J, ed. (1999). *Handbook of Creativity*. Cambridge University Press. ISBN: 9780521576048.

Strauss, Anselm L and Juliet M Corbin (1998). *Basics of Qualitative Research*. English. Techniques and Procedures for Developing Grounded Theory. SAGE Publications, Incorporated. ISBN: 9780803959392.

Suwa, Masaki and Barbara Tversky (1997). “What do architects and students perceive in their design sketches? A protocol analysis”. English. In: *Design Studies* 18.4, pp. 385–403. DOI: 10.1016/S0142-694X(97)00008-2.

Taboada, Maite and Mann (2006a). “Applications of rhetorical structure theory”. In: *Discourse Studies*. DOI: 10.1177/1461445606064836.

— (2006b). “Rhetorical structure theory: Looking back and moving ahead”. In: *Discourse Studies* 8.3, pp. 423–459. DOI: 10.1177/1461445606061881.

Terry, Michael A, Matthew Kay, et al. (2008). “Ingimp: introducing instrumentation to an end-user open source application”. In: *CHI ’08: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM Request Permissions. DOI: 10.1145/1357054.1357152.

Terry, Michael A and Elizabeth D Mynatt (2002). “Recognizing creative needs in user interface design”. In: *Proc. Creativity & Cognition 2002*. ACM, pp. 38–44.

Varela, Francisco J., Evan Thompson, and Eleanor. Rosch (1993). *The Embodied Mind: Cognitive Science and Human Experience*. Cognitive science: Philosophy, psychology. MIT Press. ISBN: 9780262720212.

Verstijnen, I M, C van Leeuwen, and G Goldschmidt (1998). “Sketching and creative discovery”. English. In: *Design Studies* 19.4, pp. 519–546. DOI: 10.1016/S0142-694X(98)00017-9.

Wallas, G (1926). *The art of thought*. J. Cape.

Wang, Jane X et al. (2016). “Learning to reinforcement learn”. In: *arXiv preprint arXiv:1611.05763*.

Ward, Thomas B, Steve M Smith, and Ronald A Finke (1999). “Creative cognition”. English. In: *Handbook of Creativity*. ISBN: 9780521576048.

Weber, Théophane et al. (2017). “Imagination-augmented agents for deep reinforcement learning”. In: *arXiv preprint arXiv:1707.06203*.

Wiggins, Geraint A (2006a). “A preliminary framework for description, analysis and comparison of creative systems”. In: *Knowledge-Based Systems* 19.7, pp. 449–458. DOI: 10.1016/j.knosys.2006.04.009.

— (2006b). “Searching for computational creativity”. In: *New Generation Computing* 24.3, pp. 209–222.

— (2012a). “Crossing the Theshold Paradox: Modelling Creative Cognition in the Global Workspace”. In: *International Conference on Computational Creativity*, p. 180.

— (2012b). “The Mind’s Chorus: Creativity Before Consciousness”. In: *Cognitive Computation* 4.3, pp. 306–319. DOI: 10.1007/s12559-012-9151-6.

Wiggins, Geraint A and Jamie Forth (2018). “Computational Creativity and Live Algorithms”. In: *The title of the bookThe Oxford Handbook of Algorithmic Music*. Ed. by Roger T. Dean and McLean Alex. Oxford University Press.