

# Responsive Positioning

A User Interface Technique  
Based on Structured Space

Dissertation

submitted to the faculty of Computer Sciences,  
Electrical Engineering and Mathematics of  
Paderborn University to attain the academic  
degree of:

*Doctor rerum naturalium (Dr. rer. nat.)*

By Felix Winkelnkemper

Paderborn, November 2017





# Abstract

This thesis presents, develops and defines a new interactive user interface technique for computers using classical means of input and output such as screen and mouse or touchscreen. As an extension to the repertoire of WIMP techniques, Responsive Positioning facilitates the perception and manipulation of attributes of on-screen objects by perceiving and manipulating their positions in relation to a structured background. The technique is based on in-depth analyses of the technical potentials of interactive user interfaces and the role of space in knowledge work processes. As for both areas there are no theoretical frameworks on the basis of which a proper determination of the status quo let alone the systematic development of a new user interface technique would be possible, innovative approaches are developed and applied. Technical potentials of interactive user interfaces are determined by considering milestones in the history of the development of these interfaces. This approach allows both for a description of the state of the art, as well as for an analysis of unused potentials. Knowledge work processes are examined to what extent they use spatial properties as a means of determining properties about spatially arrangeable objects. A combination of both investigations shows that the commonly used knowledge work technique of structuring space and positioning objects in relation to it can be combined with certain potentials of digital interfaces, thereby building the foundation for a new user interface technique. This technique is subsequently developed and formally specified. Explanations of exemplary applications of the resulting Responsive Positioning technique serve the purpose of proving the feasibility of the approach while at the same time being the basis for an application architecture which allows the integration of the new technique into existing applications in combination with classical user interface approaches.

## **Zusammenfassung**

In dieser Arbeit wird eine neue Technik für interaktive Nutzungsschnittstellen für Computer mit klassischen Ein- und Ausgabegeräten wie Bildschirm und Maus oder Touchscreen motiviert, entwickelt und formal definiert. Als Ergänzung zum klassischen WIMP-Repertoire ermöglicht die Technik des Responsiven Positionierens die Wahrnehmung und Manipulation von Eigenschaften von Objekten am Bildschirm durch die Wahrnehmung und Manipulation der Objektpositionen im Verhältnis zu einem strukturierten Hintergrund. Die entwickelte Technik basiert auf intensiven Untersuchungen der technischen Potenziale interaktiver Schnittstellen und der Rolle von Räumlichkeit in Wissensarbeitsprozessen. Da für beide Bereiche keine theoretischen Grundlagen existieren, auf deren Basis eine grundsätzliche Analyse des Status Quo oder gar die systematische Entwicklung einer neuen Nutzungsschnittstellentechnik möglich wäre, werden eigene Ansätze entwickelt und zur Anwendung gebracht. Die technischen Potenziale interaktiver Schnittstellen werden durch eine historische Betrachtung der Meilensteine der Entwicklung dieser Schnittstellen gewonnen. Dies erlaubt sowohl eine Beschreibung des aktuellen Zustandes als auch eine Analyse ungenutzter Potenziale. Wissensarbeitsprozesse werden dahingehend untersucht, inwiefern in ihnen räumliche Strukturen genutzt werden um etwas über im Raum positionierbare Objekte herauszufinden. Eine Kombination beider Ansätze zeigt, dass in der Wissensarbeit übliche Techniken der Strukturierung des Raumes kombiniert mit Potenzialen digitaler Technologien eine Grundlage für eine neue Nutzungsschnittstellentechnik darstellen können. Eine solche Technik wird im Fortgang der Arbeit formal entwickelt und spezifiziert. Die Erläuterung beispielhafter Anwendungen der resultierenden Technik des Responsiven Positionierens dient abschließend nicht nur dazu, die Anwendbarkeit der entwickelten Technik zu verdeutlichen, sondern vor allem auch, eine Anwendungsarchitektur zu erarbeiten, die die Integration der neuen Technik in bestehende Anwendungen und klassische Nutzungsschnittstellen ermöglichen soll.

**To all those who allowed me to think!**



# Contents

1	Introduction.....	1
2	Interactivity and the Interface.....	9
2.1	Definitions of Interactivity.....	9
2.2	The Interface as an Object World Created for the User.....	14
2.3	Formalisation and Automation.....	19
2.3.1	Written Calculations as Formal Operations.....	20
2.3.2	The Automatic Execution of Algorithms.....	24
2.3.3	Non-Interactive Computers.....	25
2.4	Characteristics of Programmed User Interfaces.....	33
2.4.1	Responsiveness.....	33
2.4.2	Virtual Objects.....	35
2.4.3	Spatial Objects and Direct Manipulation.....	39
2.4.4	Responsive Manipulation.....	45
3	Differential Experience and Meaningful Use of Space.....	51
3.1	Differential Experience.....	52
3.2	Using Media for Differential Experience.....	58
3.3	Object Arrangements.....	61
3.4	Space as a Means of Differential Experience.....	63
3.4.1	The Use of Space on a Desk.....	63
3.4.2	Spatial Arrangements in “Thought Structuring”.....	71
3.4.3	Structured Space in Knowledge Work.....	81
3.5	Of Artefacts and their Genesis.....	88
3.6	The Similarity of Structured Space and the User Interface.....	91
4	Structured Space and Semantic Positioning.....	93
4.1	Engelhardt’s Language of Graphics.....	98
4.2	Structured Space and Semantic Positioning.....	110
4.2.1	Structured Spaces as Combinations of Basic Structures.....	114
4.2.2	Semantic Positioning as an Umbrella Term.....	120
5	Responsive Positioning Semantics.....	129
5.1	Correspondence in Responsive Positioning.....	133
5.2	Suitability Criteria.....	144
5.2.1	Independence.....	144
5.2.2	Unambiguousness.....	145
5.2.3	Absence of Conflict.....	146
5.2.4	Stability.....	146
5.3	Combined Structures.....	151
5.4	Placeability.....	160
5.5	Structures, Sections and Contexts.....	162

6	Applications and Architectures.....	171
6.1	An Example of an Application Logic.....	173
6.2	Responsive Positioning and the Model-View-Controller Paradigm.....	177
6.3	Aspects of Responsive Positioning User Interfaces.....	188
6.3.1	Interrelated Objects.....	189
6.3.2	Manipulation Restrictions.....	191
6.3.3	Multiple Contexts in Multiple Sections.....	192
6.3.4	Automatic Context Switching.....	193
6.3.5	Combinations with other UI techniques.....	199
6.3.6	List Structures.....	200
6.3.7	Views on the Same Object Ensemble.....	202
6.3.8	Objectification and “Unreal Objects”.....	204
6.3.9	Views on Different Object Ensembles.....	210
6.3.10	Configurable Views.....	212
6.4	Model-View-ResponsivePositioningModel.....	219
6.5	WebArena – An Implementation and New Potentials.....	222
6.5.1	Virtual Knowledge Spaces.....	223
6.5.2	Visual Clients for Virtual Knowledge Spaces.....	225
6.5.3	WebArena.....	226
6.5.4	Responsive Positioning in the WebArena Scenario.....	229
7	Summary and Conclusions.....	235
7.1	Open Research Questions.....	238
7.2	Epilogue.....	245
	References.....	247

# 1 Introduction

Despite many improvements in the usability and the ergonomics of user interface design, the concepts behind classical screen and mouse-based interactive interfaces have not changed much in the last decades. ISO 9241, the ISO standard for ergonomics of human-computer interaction, for example, lists four “dialogue techniques”<sup>1</sup>. All of them, menus, command languages, direct manipulation, and form filling, have been around at least since the 1980s.

In this thesis, I motivate a new and more prominent role for the properties of space in such classic user interfaces and develop and substantiate a new user interface paradigm<sup>2</sup> in which object attributes can be manipulated by manipulating the positions of the objects in relation to a purposefully structured space.

Behind this “mission statement” there is a hidden claim, namely that there must be some shortcomings in the way space is used in present-day interactive user interfaces. Identifying those shortcomings requires knowing both the state of the art as well as something to compare it with. There is no coherent theoretical framework which could be the basis for describing spatial aspects of interactive user interfaces and which could eventually be a basis for the conception of a new user interface technique. I therefore develop own approaches which motivate the new interface technique and provide guidelines and criteria for its development.

In **chapter 2**, I focus on interactive user interfaces and determine, which technical characteristics make such interfaces possible. A typical first attempt to identify these characteristics is, of course, considering existing definitions. In case of interactivity, an inquiry reveals a large variety of definitions which can hardly be brought together as they differ both in perspective as well as in their intentions.

---

1 See e.g. ISO 9241-110 (2006)

2 In proclaiming a new paradigm, I do not want to encourage a paradigm shift in Kuhn’s sense but would like to see it as the introduction of a complementary technique which can be added to the repertoire of techniques a user interface designer can use and combine with other techniques in order to provide the best user interface in a given context. The term “paradigm shift” has been shaped by [Kuhn (1962/1970)], who tried to determine the role of “revolutions” in the sciences. A general idea which can be found throughout his book is maybe best described as “though the world does not change with a change of paradigm, the scientist afterwards works in a different world” (page 122). According to Kuhn, paradigm shifts are not proclaimed (as they often are today) but only occur, when a previous paradigm, a previous model of explanation, cannot explain the phenomena anymore and if, at the same time, a new paradigm is around, which can take its place (pages 144-145).

Most definitions of interactivity quite rightfully point out, that the concept cannot be explained in purely technical terms, as interactivity involves both the computer as well as their human users. This insight led to definitions which are quite holistic and thereby do not clearly distinguish between technical and human properties. While, on the one hand, many of the existing definitions provide useful factors using which the interactivity of specific systems can be measured or compared, they do not provide any directions for the development of interactive technology.

The lack of an existing coherent theory thus demands a different approach. A technical (yet theoretical) framework for interactive user interfaces does not necessarily rely on a universal definition of the concept of interactivity but can rather focus on which technical aspects are necessary to build interactive computer systems. Thus, in my approach, technical potentials are carved out by considering important milestones in the development of interactive user interfaces and explaining which technical innovations were necessary to make them possible. A systematic examination of these potentials does not only allow for the description of the state of the art from a technical perspective but can also be used to reveal “voids”, in which the combination of technical potentials would allow for certain use cases which are not yet realised in current computer systems.

Virtual objects, which are objects provided by the computer, play a central role in my approach of describing user interfaces. While the term object became popular only much later, such virtual objects can already be identified in the command line interfaces of time-sharing computers in the 1960s and 1970s. These computers became popular as they exploited the technical potential of *responsiveness*. Connecting a terminal to a time-sharing computer and invoking commands allowed much faster response times than before, where the time between providing an input in the form of punchcards and getting a printed result could take up to a day. Using a time-sharing computer, users did not create a stack of punchcards but had to use a command line interface. Such interfaces still are around today. They provide virtual objects, such as text files or directories, which can be manipulated by invoking commands on them. By providing such objects, the interface creates virtual object worlds within the computer. Users can perceive and manipulate the objects provided by the interface without having to care about



technical aspects such as memory organisation or how letters are internally encoded as numbers.

Objects in a command line user interface have to be addressed textually, which means that in order to perceive an object an explicit command has to be called. The resulting output on the screen can be perceived, but what is visible on the screen cannot be manipulated. Users are required to textually refer to the object, e.g. by providing a name or a path, and have to textually ‘describe’ their intended manipulation. As executing such a command typically does not alter the previous output on the screen, to be sure about the changes users have to request another output of the changed state of the object. A clever way of coupling high processing power, graphical screens and new input devices later allowed for user interfaces which featured spatially selectable and spatially manipulable objects on the screen. Such an interface is an enormous advantage compared to command line interfaces. Objects are constantly visible on the screen. Their existence and their state thus do not have to be remembered. They can be selected instead of having to be described using a name or a path, and they can be manipulated directly in place, thus avoiding a complicated command syntax.

The possibility of such a manipulation in-place, the potential of *direct manipulation*, shows an importance of space in modern user interfaces. **Chapter 3** focuses on concepts of spatiality in a different context by examining how space is used in knowledge work. The term knowledge work here refers to that kind of work which is not associated with muscle or machine power but in which mental processes are in focus. Typical examples of knowledge work are teaching, learning, performing research, but also every kind of office work<sup>3</sup>. I develop “means of differential experience” as the central concept of knowledge work. In simple terms, a means of differential experience is a means using which knowledge workers can compare their assumptions about an object or a process with a perception of that object or process which is independent of their assumptions. A magnifying glass is a means of differential experience as it allows the perception of structures which are too small for normal human perception. Space plays a very important role in knowledge work and thus in making differential experiences. The main part of the chapter thus examines how space is

---

3 See „knowledge workers“ in Drucker (1959) and Drucker (1970).

used in knowledge work. A special focus is put on techniques where the potential of space getting structured is used as a means of differential experience.

The examination in chapter 3 shows that in knowledge work structured spaces are used to both visualise and manipulate properties of objects. This aspect can best be illustrated by considering a simple example. A table contains columns labelled Monday to Friday. Such a table can be used for a number of purposes, one of which could be the determination of which person has to clean the office on which day of the week. Assume the table is attached to (or is painted on) a magnetic surface. Small magnets labelled with the names of people, thus representing them, can be moved around on this surface. Placing an object within one of the columns gives it a meaningful interpretation. It assigns the office cleaning duty to the respective person for a day of the week. Moving an object from one column to the other is interpreted as a change of the assignment, accordingly. Objects in this scenario are meaningfully placed. By moving them into a certain position in relation to their surface or their painted background, ‘meaning’ is assigned to them.

A situation where something needs to be made perceivable and manipulable is not only typical in knowledge work scenarios but is also very common in user interfaces which visualise virtual objects and have to provide means of making object attributes both visible as well as manipulable. Imagine, as a simple example, a todo application in which tasks are either still to be done, are in progress, have already been done or have been rejected. This object status needs to be both visualised as well as made manipulable by the todo application. The interface of the todo-applications most likely would not look anything like the magnetic table for the duty assignments, but it serves quite a similar purpose. Both are means of making some otherwise hidden object properties perceivable and manipulable. Generalizing this thought gives reason to formulate the hypothesis that *positioning objects in relation to structured spaces can be a proper user interface technique*.

Even though interactive user interfaces have the potential of evaluating object properties and thereby reacting to property changes (the technical potential of *responsive manipulations*), the manipulation of object positions in relation to structured spaces, and here I come back to the “hidden claim” from the beginning, are not systematically used in interactive user interfaces today.

For transforming the knowledge work technique into a user interface technique, structured backgrounds and how they work as well as the mechanics of positioning objects in relation to them must be formally described. In **chapter 4**, I therefore, examine a comprehensive concept of information graphics, which is a field in which semantic object relations are depicted as spatial object relations, and adapt it for my own definition of structured space. The clue of this definition is its focus on what I will explain as object-to-space relations which, to give a brief characterisation, means that objects gain some meaning not by considering the positions of other objects but by their relation to a structured background only. In the example above, the magnets gained their attributes through their positions not in relation to each other but only in relation to the table.

Just positioning objects in relation to some structured space is not yet a user interface. In a user interface using which object attributes can be perceived and manipulated, the arrangement of objects in relation to the structured space must always mirror the state of the object attributes of interest. In case of a change of an attribute, an object's position must instantly be updated. This does not mean that an object position must change suddenly. There may, of course, be an animation which supports the users' perception of the change, but must not be a considerable delay in which the object dwells in the wrong place. In the other direction, every spatial manipulation of an object must instantly be translated into a change of the attributes. Both the automated positioning as well as the attribute manipulation by performing a spatial manipulation must happen without any perceivable delay. As described above, I identified the quality of reducing response times as *responsiveness*. While in command line interfaces responsiveness requires a program to constantly await user input and by reacting to it without long delays, in this case it is made possible through a technical design which monitors changes in positions and attributes and immediately triggers actions in the other. The coupling between object attributes and object positions must be responsive, hence the name *Responsive Positioning* for the user interface technique. When an interface exhibits this form of responsiveness, users can perceive object attributes by perceiving object positions and can manipulate object attributes by manipulating object positions. When attributes and positions correspond to each other this way, object attributes and object positions virtually become the same.

**Chapter 5** of the thesis describes, in-depth, how evaluation semantics must be defined in order to support such a correspondence. It also covers, how complex structured backgrounds can be created. Such structured backgrounds describe more complex forms of correspondence in which, unlike in the simple example given here, more than one attribute can be perceived and manipulated using a single interface. To achieve this, I develop formal definitions for basic structured backgrounds and formally describe how correspondence between attributes and positions in complex structured backgrounds can be assured by combining complex backgrounds from simpler ones.

In **chapter 6**, I extend Responsive Positioning from a technique which only allows the perception and manipulation of object attributes to a user interface paradigm which allows controlling applications by manipulating object positions and by perceiving object movements in relation to structured backgrounds. To do that, the Responsive Positioning concept is complemented by an application layer which encapsulates an application logic. This application logic can access the user interface objects but does not directly manipulate the aspects of visual presentation as it can only create or remove objects and read or manipulate their attributes. In this context, the chapter examines the compatibility of the Responsive Positioning user interface technique with different incarnations of the popular Model-View-Controller concept. This comparison shows that Responsive Positioning is not compatible with many historical and common Model-View-Controller interpretations. In Responsive Positioning aspects of the functionality of a user interface cannot be strictly separated from visual aspects of that user interface because visual aspects include the structures of the background which are, of course, the very centre of the functionality of the user interface. On the basis of the modern Model-View-ViewModel paradigm, I develop an adaptation which can serve as a design pattern for applications using Responsive Positioning and which allows Responsive Positioning user interfaces to be embedded into applications using classic means of user interaction.

While the main part of the chapter carves out aspects of Responsive Positioning for user interfaces which are predetermined by the applications, a final part of the chapter motivates a scenario which is possible when combining Responsive Positioning with WebArena, a software which I developed while working on this thesis. WebArena builds the bridge between the field of knowledge work, from

which Responsive Positioning inherits its spatial-organisational concepts, with an implementation of the Responsive Positioning semantics for user interface purposes. On the one hand, it allows a flexible spatial organisation of knowledge artefacts by using individually structured spaces, while on the other hand, it supports applications which have access to the attributes these objects have been assigned with. This combination allows for scenarios in which, instead of being provided with a complete user interface, users have the means of creating their user interface on an on-the-fly basis.

**Chapter 7** concludes the thesis mainly with ideas and suggestions for research questions which were left open or which extend the scope of this thesis. Many of the open issues cover the brought questions like what kind of user applications Responsive Positioning is good for, for which kinds of applications it cannot be used, and whether Responsive Positioning as a user interface is a better choice than the already existing user interface solutions and to what extent this could be tested.



## 2 Interactivity and the Interface

In this thesis, I introduce a new technique for interactive user interfaces which aims at tapping spatial user interface potentials by evaluating object attributes and object positions in such a way that object attributes can be perceived and manipulated by perceiving and manipulating object positions in relation to a structured space.

As a first building block of this concept, in this chapter, I examine the technical potentials of interactive user interfaces and thereby describe the role of spatiality in current user interfaces. In a first attempt at finding out the technical properties which constitute interactive user interfaces, I examine existing definitions of interactivity, before continuing with an approach of my own which carves out the technical potentials of interactive user interfaces by considering important milestones in their development and explaining, which technical developments were necessary to make them possible.

### 2.1 Definitions of Interactivity

While the term “interactivity” is widely used today, a quick web search reveals there is no clear understanding, let alone a common definition, behind it. From an “interactive tax assistant” to “interactive television”, an “interactive chat” and “interactive systems”, everything seems to be interactive, yet it remains unclear what the interaction is, what or who is interactive and among whom interactions might take place. A similar ambiguity can be found in corresponding scientific literature. While Rafaeli (1988) found that “interactivity is a widely used term with an intuitive appeal, but it is an underdefined concept,” and then humorously concluded that “interactivity, like news, is something you know when you see it”, today, 30 years gone by, there is, of course, no lack of definition attempts. Quiring&Schweiger (2008) thus considered available definitions, referred to Rafaeli’s comment and concluded that interactivity is a “rather overdefined concept”.

In this thesis, I cannot give a comprehensive overview of all definition attempts out there. Authors who provide such overviews, like Kioussis (2002) and McMillan&Hwang (2002), for example, considered definitions by 22 or 30

authors, respectively, but even they only provide a selective picture. The different definitions and characterisations found in the literature depend very much on the scientific backgrounds of their authors. Kiouisis' claim that interactivity "is associated with new communication technology, especially in the internet and the world wide web"<sup>4</sup> has to be interpreted knowing his background of new media and public relations. His view, therefore, differs strongly from a technician like Franchi (1975), who defined an "interactive environment" as one in which "the effort to reduce the detection and intervention delays [in finding programming errors] does not cause a substantial increase in the use of resources."<sup>5</sup>

In this section, I look into some of the existing definitions and describe, why they do not suffice to provide technical requirements for interactive user interfaces and thus cannot be the basis for the formulation of technical potentials of such interfaces. The presentation of these definitions and characterisations also shows that within the spectrum of interactivity concepts there have been significant changes over time and that there are some trends which are more suitable to the interface idea presented later in this chapter, while others go into different directions.

An unfortunate decision during the transition from batch computing to the first text-based interactive systems, which will be explained in some detail in the next sections, was describing the back and forth of text input and text output as somehow being similar to human communication<sup>6</sup>. I can only assume that this was the reason for authors from Rafaeli in 1988 to Quiring and Schweiger in 2008 to relate their definition attempts to the sociological concept of human-to-human interaction. Because of that, their understandings of interactivity refer to the *process* of interaction, so even when the concept is extended to human-machine relations, they consequently claim that interactivity cannot be explained in purely technical terms as such an approach would inevitably focus on the machine rather than on the process.

---

4 Kiouisis (2002), page 356

5 Franchi (1975), page 122 (the words in the brackets have been inserted in order to provide the context of the definition)

6 Streeter (1974), for example, distinguishes 3 modes of usage (batch, interactive and real-time) and defines the interactive mode, which he also calls "conversational", as "a mode involving dialog between user and computer which resembles interpersonal conversation in speed and quantum of communication (i.e. a single statement or question followed by acknowledgement of [sic!] answer." (page 12).



Rafaeli based his definition and distinctions explicitly on the concept of communication<sup>7</sup> and, therefore, concluded that human-computer relations are mostly not interactive but should rather be called reactive only. He defined interactivity as “feedback that relates both to previous messages and to the way previous messages related to those preceding them”<sup>8</sup>. In other words, for Rafaeli interactivity requires a conversational context which was neither given in command line based user interfaces of the time nor is it in today’s modern graphical user interfaces.

Quiring&Schweiger (2008) do allow the term to be used for “user-system interactivity”, which they put in contrast to other forms of interactivity based on human-human communication<sup>9</sup>. They define user-system-interactivity as “a communication scenario where humans interact with a media system, which represents content and may react to input by the user”. This communication, they point out, happens without exchanging any meaning as the media system is not capable of understanding meaning like a human counterpart would. Notice that both Rafaeli and Quiring&Schweiger define interactivity as a property of the act of communication, so it is neither the user nor the system which is interactive but rather the relation between the two.

In contrast to Rafaeli (1988) and Quiring&Schweiger (2008), many other researchers do accept interactivity as a property of technical artefacts. Schelhowe (1997) found interactivity as a property attributed to a thing or to a person to be a re-invention in the wake of the modern computer<sup>10,11</sup>. Though not providing a

---

7 Rafaeli (1988), page 119

8 Ibid., page 120

9 Quiring&Schweiger (2008), pages 151,152

10 Schelhowe (1997), page 158

11 Doing a web-based book search for occurrences of the terms “interaction”, “interactive” and “interactivity”, it becomes clear that the term has long been used before being attributed to human communications and eventually being adapted to a property of a computer system. Interactions with the environment were often spoken about in scientific literature even before the computer age and while, of course, the use of the word interactivity increased strongly with the advent of interactive computers, there are examples where the word has been used to describe a property of things and people even before that. In the introduction to Dewey& Ratner (1939), for example, Joseph Ratner describes the scene of “a horse, stone and stretched rope between”: “Horse, rope and stone are organized together and are interacting with each other. [...] They are interacting with each other and also with other things. Each thing in the interactive system constituting that situation is an individual thing with its own distinguishable boundaries because of multitudes of interactivities going on within it.” (page 146)

definition of her own, she characterises interactivity as “the crucial link in order to complement automatic processes with human actions” in order to “make processes which are only partially, yet not fully computable” become accessible to computer-based organisation<sup>12</sup>. This, as she added in Schelhowe (2007), makes the user “behave like a cyborg” in a “symbiosis between automatic operations and human actions”<sup>13</sup>.

Janlert&Stolterman (2017) made some steps towards a definition of interactivity which refers to properties of the artefacts which exhibit such a behaviour. They complement the term interactivity, which they restrict to the ongoing process of interaction between a human user and a computer system, with the two rather uncommon terms “interactability” and “interactiveness”. By *interactability* they refer to the “intrinsic quality of an artefact or system, their potential, to engage in interaction”. In order to rule out rather passive artefacts like pieces of paper which could nevertheless be called “interactable” using the definition given above, they sharpen their definition by linking interactability to *agency*, which they characterise as user actions resulting in recognisable and reasonably predictable reactions. As they connect agency with attributes like having a “will of its own” or even “acting up”, fully predictable reactions would result in a low attribution of agency, yet scarcely predictable reactions would appear random and leave the users of an artefact with the impression that their inputs and perceived system behaviour are unrelated, which again would result in a low attribution of agency. Their definitions of interactivity and interactability are accompanied by the notion of *interactiveness* which describes how much a system or artefact forces a user to interact with it. A ringing telephone, for example, exhibits a high degree of interactiveness while a word processor exhibits hardly any.

While Janlert&Stolterman’s characterisation of interactability as the attribution of agency to some degree describes properties of the technical system, like producing reactions, it also shows a common phenomenon among many

---

12 Schelhowe (1997), page 148: “Interaktivität ist das entscheidende Bindeglied, um die maschinellen Prozesse durch menschliches Handeln zu ergänzen und die Maschine in menschliches Handeln einzubetten. Interaktivität ist das Mittel, um Vorgänge, die nur in Teilen, nicht aber in ihrer Gesamtheit berechenbar sind, dennoch mit Hilfe von Computern zu organisieren.”

13 Schelhowe (2007), page 154: “Sich als Cyborg zu verhalten, ist diesem Medium angemessen, das nicht – wie in der Frühzeit angenommen – als autonome Rechenmaschine erfolgreich wurde, sondern in der ‚Symbiose‘ mit dem Menschen, in der Interaktion zwischen maschinellen Operationen und menschlichem Handeln.”

definitions. In their quest for definitions of interactivity (including interactability and interactiveness), at some point, they end up with a statement which says that an artefact is interactive or interactable if their users *ascribe* interactive behaviour or agency to it. If, for example, a system reaction is not perceivable enough, cause and action *feel* unrelated, thereby diminishing the interactiveness of the system. Laurel (2014) was confronted with a similar problem when having to confess she had to give up on her previous approach of determining variables which constitute interactivity. She concluded that, at the end of the day, the question whether or not something is interactive is just a matter of whether a user *experiences* it as such:

“You either feel yourself to be participating in the ongoing action of the representation or you don’t. Successful orchestration of the variables of frequency, range, and significance can help to create the feeling, but it can also arise from other sources – for instance, sensory immersion and the tight coupling of kinesthetic input and visual response.”<sup>14</sup>

While Janlert&Stolterman’s and Laurel’s approaches are similar in referring to a feeling or attribution, Laurel’s initial variables for interactivity, which are frequency (how often you interact), range (how many choices are available), and significance (how much choices really affect matters)<sup>15</sup>, have quite a different focus than the agency described by Janlert&Stolterman. Agency focuses on reactions of artefacts on some input while frequency, range, and significance focus on to what extent something going on can be influenced. Thinking about interactiveness and agency as well as interactability implies a system of cause and effect, while frequency, range and significance suggest an active world within the computer which can be influenced to a certain extent.

This brief look into relevant literature already offered a cornucopia of interpretations and perspectives. All of them describe interactivity from its phenomenological side thereby producing valuable knowledge about how to design interactive systems. Having said this, they do not systematically describe technical characteristics which are needed to create interactive systems and thus they cannot be a means of uncovering unused potentials in current interactive technology. As this is my goal though, I cannot rely on the definitions provided

---

<sup>14</sup> Laurel (2014), page 29

<sup>15</sup> Ibid.

here. I therefore rather take a different approach by looking onto interactivity “from the other side” by describing the historical development of the user interface. As the concepts behind interface and interactivity, of course, are not the same, an examination of the historical development of interfaces will not result in a new definition of interactivity. Such a definition thus cannot be the goal here. For my argumentation, I do not necessarily have to define interactivity to its full extent, but rather “only” need to understand which technical aspects make it possible. By considering the technical milestones in the development of the interface, I thus assume that those technical properties which made the user interface evolve constitute the technical potentials which allow for an interactive use of an artefact.

### **2.2 The Interface as an Object World Created for the User**

Using a modern computer, one is constantly working with or referring to objects which are displayed on the screen. Some of these objects are part of what is generally called the content. Examples of such objects are graphical circles, squares and text boxes which can easily be selected, moved around, or resized. Other objects on the screen, such as menus or buttons, are not part of the content but are what Nielsen (2012) calls “GUI chrome”<sup>16</sup>. Both content objects and chrome objects constitute the user interface of an application. The computer is creating those objects solely for the purpose of users being able to perceive and manipulate them. Without such perceivable and manipulable objects the computer would be a complete black box whose internal affairs could neither be perceived nor manipulated.

Every modern computer needs an explicitly programmed user interface. Data in the working memory and on the storage devices of a computer, the *internal memory*, can neither be seen nor manipulated directly. Of course, this data does exist physically in the form of voltages, electric currents, or magnetisations, and thus theoretically could be manipulated in that form<sup>17</sup> but doing so would mean

---

16 Nielsen (2012) describes GUI chrome as “the visual design elements that give users information about or commands to operate on the screen's content (as opposed to being part of that content).”

17 In that sense software and data are hardware. Keil-Slawik (2000) states that typography, in the form of signals, influences “energetic processes” necessary for automatic processing and

one would need to have means of switching bits in internal memory or magnetising a tape or a disk. Such forms of directly manipulating the contents of memory may have been possible among the very early computers like the Eniac or Zuse Z3 where the content of the memory was visualised through lights and where input and output storage media existed in the forms of cables, resistor-matrices, switches, punched cards, or punched tape which were perceivable and manipulable outside the computer<sup>18</sup>. In the modern computer this is no longer the case, so a program and its data are invisible unless someone develops an explicit user interface. From this perspective, the purpose of a user interface can be described as making a user able to access and manipulate the internal memory of a computer.

Today's user interfaces, of course, have quite a different focus. They do not provide means of accessing any internal structure directly. Instead, users find themselves in a world of manipulable objects displayed on a screen. These objects are virtual<sup>19</sup>, which in this case means they are the product of a skilfully created program running on a fast processor which orchestrates input and output in a way which results in the impression of perceivable and manipulable objects. Tognazzini (1991 and 1995)<sup>20</sup> pointed out that what users interact with neither is the hardware nor the operating system but rather a well designed artificial world, a "well constructed illusion" for the user. While with this description Tognazzini made a clear distinction between the user sphere and the technical sphere of a computer system, the "illusions" he spoke about themselves have to be created by technical means, by the technical construction of a computer generated "in between", the *user interface*. This interface does not only specify what one can

---

concludes that "in this sense, software is hardware" ("Typographien sind physische Gebilde, denn nur so können sie als Signale energetische Prozesse beeinflussen, die letztlich für eine maschinelle Verarbeitung erforderlich sind. Insofern kann man feststellen, daß auch Software Hardware ist.")

18 In fact, program and input data were manipulable *only* outside the computer, as will be shown a little later in the argumentation.

19 A detailed definition of what virtuality means and how it found its way into the computer discourse can be found in Skagestad (1998). For my purpose, the definition found in the German Wikipedia describes it well as "that property of a thing, not to exist in the form in which it appears to exist, yet to resemble a thing existing in this form in characteristics and effect." ("Virtualität ist die Eigenschaft einer Sache, nicht in der Form zu existieren, in der sie zu existieren scheint, aber in ihrem Wesen oder in ihrer Wirkung einer in dieser Form existierenden Sache zu gleichen.")

20 Tognazzini (1991), page 132, and Tognazzini (1995), page 302ff

see but also which kinds of manipulations are possible. Interpreting the user interface this way puts user interface designers into a very responsible position. It is due to their decisions whether the users feel like they are interacting with meaningful objects in a way that feels natural to them or if they feel like having to control complicated machinery they do not understand.

The question whether a user interface feels sufficiently non-technical is, of course, one of perspective and degree. When Tognazzini (1991) claimed “Macintosh users don’t know the operating system. They know a fanciful illusion called the desktop”<sup>21</sup>, what he had in mind about what Mac users did not know was a command line interpreter and a complex directory structure imposed by the operating system. From today’s point of view, computer systems which can only be controlled using a command console appear to be very technical but, of course, users of such systems do not operate their machines on the basis of memory registers and processor commands but use some, maybe a little less fanciful, “illusions” like a command shell and a file system. Nevertheless, these user interfaces appear to be at the low fidelity end of virtual user worlds. This becomes especially obvious when comparing them with those programmed worlds you find in the early desktop systems developed by Xerox and Apple.

On the Xerox Star, which was presented in 1981, according to Johnson et al. (1989), users were completely “oblivious to concepts like software, operating systems, applications and programs”.<sup>22</sup> Instead, they found themselves in a document world<sup>23</sup> without any concept of programs or directories, let alone something like a command console. New documents were created by duplicating existing documents on the so-called “desktop” (see figure 1). The desktop was a graphical space which was provided by the system for the purpose of providing the users with a distinct location where they could manage those documents and folders they used for their current work. In order to make this possible despite the restrictions of the screen size, the objects were not depicted full size but rather as small pictograms they called icons. A catalogue contained a number of standard objects such as empty text documents, empty paintings and empty folders. Those elements could be copied onto the desktop where they could be opened to (which means enlarged into) a window in which they could subsequently be edited. The

---

<sup>21</sup> Tognazzini (1991), page 236

<sup>22</sup> Ibid., page 11

<sup>23</sup> see Johnson et al. (1989), page 13

idea behind this system was the creation of template documents for specific tasks, from which concrete documents could be created by duplication. The desktop allowed the users of the XEROX Star to organise their working material according to their individual needs, so, for example, the desktop of people who frequently had to write letters most likely contained a template document for letters which they could duplicate in order to create a new letter document.

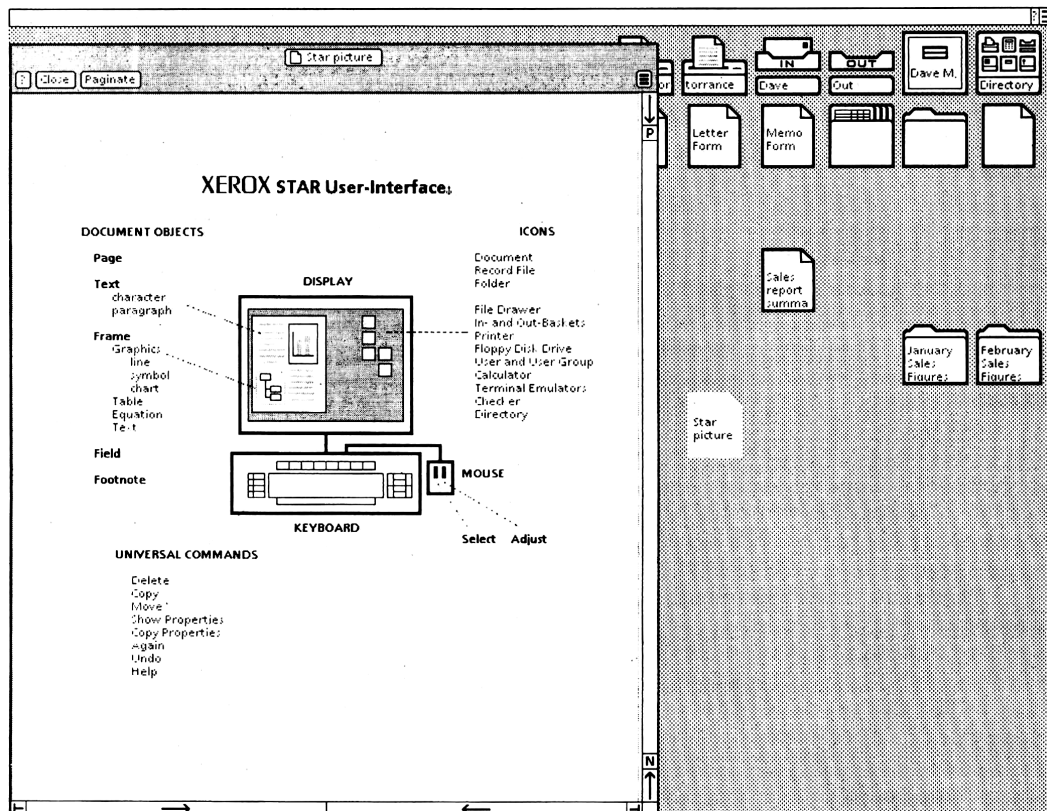


Figure 1: A Desktop as it appears on the Star screen. Several commonly used icons appear across the top of the screen, including documents to serve as "form-pad" sources for letters, memos and blank paper. An open window displaying a document containing an illustration is also shown. (Screenshot and caption from Smith et al. 1982)

The Apple Lisa, which was presented in 1983, not coincidentally offered similar methods of organising one's personal working space. Just as on the Xerox Star, new documents were created by duplicating existing ones. In contrast to the Star, where this was done using a generic duplication command which for that purpose not only worked for documents but, for example, also for graphical objects or text

## 2 Interactivity and the Interface

inside those documents, Apple's user interface used the metaphor of a stationery pad which is explained in the operating manual as follows:

"Each stationery pad represents an infinite supply of either blank or customized paper, which you use for creating new documents. The design on the pad matches the design on the tool and documents associated with the pad. [...] Each tool comes with a pad of blank stationery, and you can make your own custom stationery pads."<sup>24</sup>

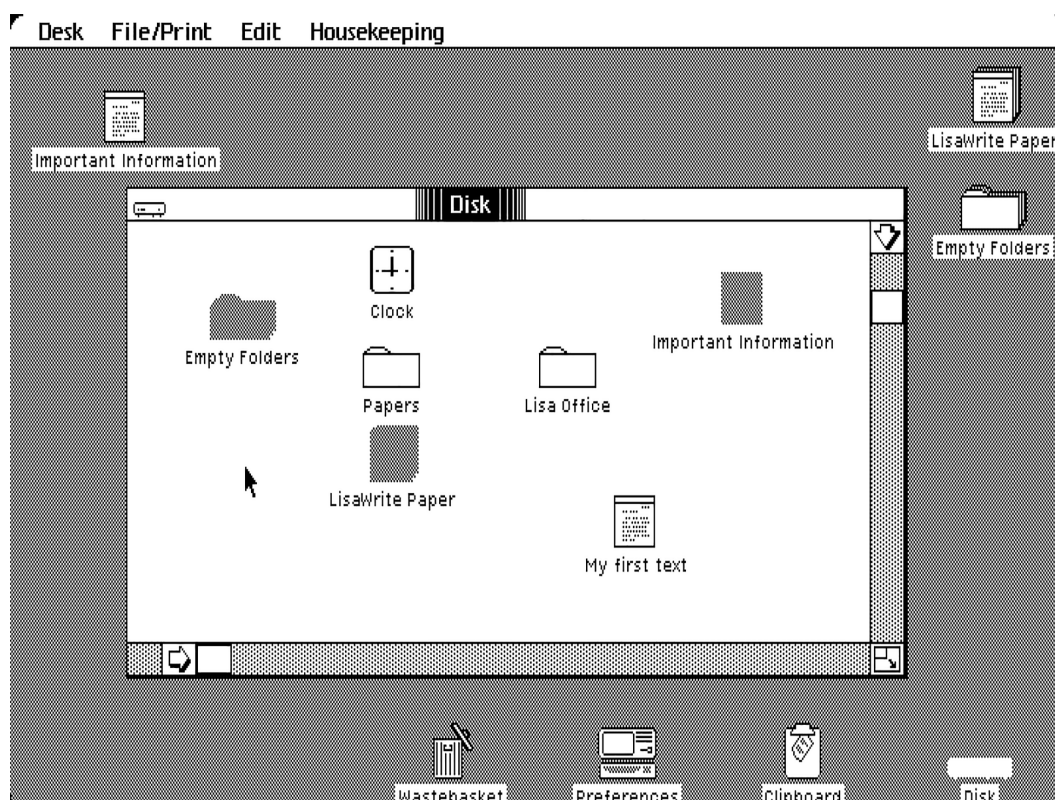


Figure 2: The desktop of an Apple Lisa showing Stationery Pads for empty document and folders (top right) and an opened window showing the content of the hard disk drive. The greyed out icons within the window represent the filing locations of objects which are currently placed on the desktop.

On the Lisa, every document and even folders including all the documents they contain could be made stationery pads which then served as templates for new objects of the same kind. Double-clicking a stationery pad icon or invoking the associated menu command created a duplicate document or folder placed close to

<sup>24</sup> Apple (1983), page B-16



the location of the pad. It metaphorically resembled tearing off a piece of paper from a real-world form pad<sup>25,26</sup> (see figure 2).

In order to understand how the computer progressed from an automated calculator to a device which creates virtual object worlds for the user, it is necessary to understand the technological developments and the forces which stood behind the evolutionary steps from cable programmed computers like the ENIAC in the 1940s and 50s to object-oriented graphical systems we use since the 1980s. In order to do that, I start with theoretical concepts which by far predate early digital computers. Computers are built on the foundation of formal definitions and formal transformations. The clue of such transformations is that they only depend on the form and the arrangement of symbols. I will argue that in evolutionary steps the computer became a machine which provides the means of creating and manipulating such symbols and objects. Objects which were created and manipulated using a computer, of course, can themselves be evaluated by the same computer which brought them to existence. Not only does this allow the computer to be the machine which is used to create the programs which run on it but it also allows for a formal spatial evaluation of objects, which is one part of the foundation for what is described in this thesis.

### **2.3 Formalisation and Automation**

Imagine one measures the height of a child every month by making markings on a doorframe. Those markings on the doorframe allow the perception of a process which would otherwise not be perceivable. As the markings remain persistent—they do not disappear by themselves—one can compare them with each other. While the growth of a child is too slow to be perceived just by observing the child, now there is a trace of the long-running growth process “written into” the doorframe. These doorframe inscriptions are analogue in nature. A child is asked to stand at the doorframe, and the proud parent creates a mark in the frame directly above the child’s head. The markings can be directly perceived and compared, yet if one wanted to determine whether the growth per month is

---

25 The stationary pad functionality still exists on modern Macintosh systems as one of the maybe most forgotten features. Every file can be made a stationary pad by setting a flag in its properties dialogue. When such a file is opened, e.g. by double-clicking it, a duplicate of the file is created and opened.

26 The “user world” of the Apple Lisa provided a number of additional features not found in any modern digital desktop system. More on that can be found in chapter 3.4.1.

increasing or decreasing and by what amount, one cannot do that by just looking at the markings. Instead, one would have to do some calculations, which means translating the heights of the markings into numbers by using some means of measurement such as a yardstick, write them down on a piece of paper, and then start a calculating process.

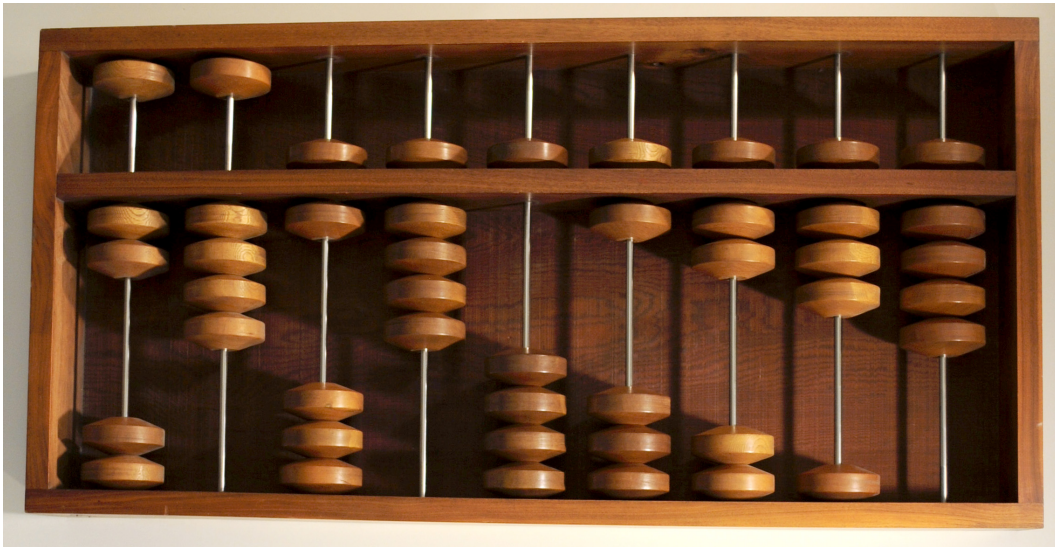
### 2.3.1 Written Calculations as Formal Operations

Humans perform written calculations by writing a number of cyphers and operators onto a piece of paper according to a strict ruleset. The fact that calculations are performed on a piece of paper or anything which allows symbols<sup>27</sup> to remain persistent at least for the time of the calculation has implications on the organisation of the calculating process. Calculating just using one's head without any media support, and thus without technology, is very limited. Mistakes are common, and if one comes up with a result, there is no easy way of knowing if it is valid because there is nothing one can check it against.

Figure 3 shows a Japanese abacus. Using such an abacus makes calculations easier and more reliable because an abacus provides a limited persistence. The last intermediate result does not have to be remembered until the next calculation step is taken because it is visually persistent in the positions of the beads, yet there still is no trail of the calculation process itself, neither of the operations which have been performed nor of any previous intermediate results. When a calculation process is finished and the result seems to be strange, there is no way of determining a possible mistake. The calculation thus has to be performed over and over again until one can be sure the result is correct. In written calculations, in contrast, each intermediate result remains visible and can thus be checked later either by the calculators themselves or by anyone else.

---

<sup>27</sup> I make no differentiation between symbols and signs. According to some authors, signs are indicators we can perceive, like smoke being a sign of fire, whereas a symbol is used in order to symbolise something, so a stylised flame would stand for the concept of fire even when there is none. Others describe signs as standing for something without themselves being what they stand for, whereas symbols are arbitrary, standing for nothing but themselves. The usage of the words does not only differ from author to author but also seems to differ from language to language. I do not need any of these differentiations, so for this thesis, I decided to always use the word symbol only.



*Figure 3: A Japanese abacus at Soda Hall at the University of California, Berkeley (Photo by D Coetzee on Flickr)*

In order to perform written (persistent) calculations, a formal symbolic system has to be established. The cyphers and operators used for written calculations are arbitrary symbols, as are the letters of the alphabet. The forms of those symbols may once have been chosen purposefully in order to be easily distinguishable in a certain environment<sup>28</sup> or may have a certain appearance due to properties of the techniques of how they are created or reproduced<sup>29</sup>, yet these circumstances of their genesis do not make the symbols meaningful by themselves. The symbol “A”, for example, has no meaning by itself, and that “M” and “W” look somewhat similar is most likely random. As the symbols do not carry meaning by themselves, the meanings of every one of the symbols and that of their possible arrangements in a specific context have to be learned and communicated to those who need to understand them.

One of the most interesting aspects of calculations, which is due to their formal nature, is that the person performing a calculation does not have to be among those who know the meaning of the symbols in their context. Before a calculation starts, someone with knowledge about the symbolic system, who knows how to *formulate* the problem, has to “translate” the real-world problem into its symbolic representation. After the calculation has ended, someone with knowledge about

<sup>28</sup> Frutiger (1989) describes traffic signs as examples of such symbols (page 346).

<sup>29</sup> See *ibid.* page 159ff

the symbolic system of the result, which does not necessarily have to be the same system, has to understand those symbols and their arrangement in order to translate them back into something meaningful in the given context. In contrast to all of those, the people who perform the actual calculations do not have to know what the symbols are, how they are called or what they might mean. Everything necessary to perform a written calculation is an arrangement of symbols and some rules which specify how the symbols have to be processed.

As indicated, written calculations are performed formally. The term “formal” can be understood quite literally. The formal interpretation of the writing relies only on *form and arrangement* of the symbols<sup>30</sup>. Calculations, as any set of formal operations, can be described as transformations of one arrangement of symbols into another arrangement of symbols. The initial arrangement, e.g. the arithmetic problem, is written down according to formal rules, which means the problem has to be written down in a well-defined way. While there may be different formalisms for the same purpose, there must be a strict and coherent ruleset within each of those. Krämer (1988) formulates a number of preconditions which have to be fulfilled for a symbol arrangement to be a formal arrangement<sup>31</sup>:

**Writtenness:** A formal representation requires a typographic medium, meaning it has to be written down on something which allows for the arrangement of distinguishable symbols which remain stable over a period of time.

**Schematizability:** A formal representation describes a problem schematically, which means several concrete problems have the same formal representation. Both “I have three bags containing eight apples each. How many apples do I have?” and “Peter, Paul and Mary each have eight Euros in their pockets. How much money do they have in total?” are both formalised into the same arrangement of symbols, “ $3 \times 8$ ”, thereby losing their respective contexts. For the calculation process, it does not matter whether the symbols refer to apples, bags, money, people or nothing at all.

**Absence of interpretation:** The (formal) correctness of a formal term can be decided without having to know anything about what the symbols refer to. “ $3 \times 8$ ”

---

30 See Krämer (2014), page 349: “In calculations, the operational rules only refer to the syntactical and that means to the perceptible shape and the spatial ordering of the signs. These rules are invariant with respect to the interpretation of the signs.”

31 In the German original they are *Schriftlichkeit*, *Schematisierbarkeit*, and *Interpretationsfreiheit*. See Krämer (1988), pages 1,2.

is formally correct, “3 8 x” would not be, at least not according to the common formalism we learn in school. Formal correctness can be determined by evaluating it according to a formal description regardless of what 3, 8 and x refer to.

Calculating is done by following formal transformation rules which operate on the formal arrangement of symbols. A coherent set of such rules is called *formal calculus*<sup>32</sup> while the sequence of steps using which, for example, written multiplications are performed is called an *algorithm*<sup>33</sup>. An algorithm can be interpreted as a formal description of a strategy to solve a certain set of problems. Krämer defines the following characteristics for algorithms<sup>34</sup>:

**Elementary:** Algorithms can be broken down into simple operations. In the case of the multiplication of two natural numbers, it can be reduced to simple multiplications for which a solution is already known (e.g. by looking it up in a table) and an addition, which can itself be performed by executing another algorithm.

**Deterministic:** For the same input the algorithm produces the same result every time it is performed. When the same set of numbers is multiplied, it always has to be done the same way, and it always has to produce the same result.

**General:** An algorithm is a solution for a class of problems, rather than for one single problem. One does not have an algorithm which describes how to multiply 387 by 298 but how to multiply natural numbers in general.

---

32 The terms formalism, formal calculus and algorithm are used with different, partially overlapping meanings throughout literature, so their distinction is not always clear. In this chapter, a formalism is that kind of writing in which the interpretation just relies on the form and the arrangement of symbols, formal calculus is a set of coherent transformation rules which can be applied to formal notations, and an algorithm is a formal definition of a strategy for solving a problem. Applied to calculations, arithmetic problems are written down according to the rules of a formalism, calculation rules are formal calculus, and the ways of doing written additions or multiplications are algorithms.

33 see Krämer (1988), page 159

34 This is a simplified characterisation, as definitions, as always, are not quite clear. In practice, there are algorithms which use random numbers and others which do, at least theoretically, never terminate. In concrete calculations, though, termination definitely is a goal, and even when random numbers are used, it is a requirement of any sensible algorithm not to produce purely random results.

**Finiteness:** The way Kramer puts it, this criterium is threefold:

- One must be able to write down the full algorithm, which means it must be possible to write it down using a limited number of symbols. This number can be extensive, but it may not be infinite.
- Each of the steps of the algorithm (e.g. each of the calculation steps) must be executable in limited time. If one of those steps would literally take forever, the whole algorithm would, of course, itself never come to an end.
- The algorithm itself must be defined in a way it creates a result in a limited number of steps so anyone who performs the algorithm can know it will eventually come to an end.

Even though calculations can be performed without having to know what they are about, someone has to decide which calculation solves the problem at hand. To “use” arithmetic appropriately, a lot of intelligent, meaningful and non-stupid work has to be performed.

### 2.3.2 The Automatic Execution of Algorithms

Calculations, as well as all other formal operations, can be executed disregarding any context. The people doing calculations do not have to know what the calculations are about and in a strict sense, they could be trained to perform calculations without even having to know what a calculation is. Performing such context-free operations like calculations is tedious work for humans, so early on scientists and engineers looked into ways of automating them.

In order to automate calculations, the calculation rules must be materialised in the true sense of the word. They have to be physically modelled so that they can, for example, be performed mechanically or electrically. Early calculating machines by Pascal and Schickard, for example, used mechanical gear-wheels for carrying out additions and subtractions<sup>35</sup>. In order to perform complex calculations, a sequence of steps, the “plan” according to Grier (1996)<sup>36</sup>, had to be performed one after the other by a human operator. When the same kind of calculation had to be performed for a number of different starting parameters, the same sequence of calculation steps had to be performed over and over again. This means that while the

---

<sup>35</sup> More on early calculating machines can be found in Williams (1990).

<sup>36</sup> See Grier (1996), page 53

calculating machine could save a lot of time and reduce the chance of errors during the respective steps, algorithms still had to be executed manually.

A digital computer can execute algorithms and apply them to different sets of input data as long as the algorithm itself is provided based on a formalism the computer can translate into its inner operations. Algorithms in that sense are not necessarily limited to solving mathematical problems but are more universal. Computers are often described as being universal machines, which in fact means they are universally programmable and thus can be programmed to perform almost any kind of formal operations<sup>37</sup>. According to Ceruzzi (1998), the first electronic computers were, as the name suggests, “invented to ‘compute’: to solve ‘complex mathematical problems’”<sup>38</sup>. These computers were mostly used in military facilities, e.g. in order to calculate projectile trajectories, but soon, computers like the Univac were also used in scientific and commercial contexts<sup>39</sup> for numerous purposes from managing inventories to producing serial letters. All these functions a computer can carry out, be they genuinely mathematical or not, have to be provided in a formal way, which means they have to be programmed.

### 2.3.3 Non-Interactive Computers

A computer by itself, of course, does not do anything without a program running on it. Providing a computer with programs and perceiving the results of a program run, in this sense, can be interpreted as an early kind of user interface.

Programming the very early computers was done by plugging in cables, manipulating resistor matrices, flipping switches, etc. In the next evolutionary step, program and data input into the computer was realised by using punchcards or punched tape. The resulting operation mode can be described by referring to the classic IPO paradigm depicted in figure 4. The input, which in this case consists of the program and its data, has to be created in every detail before even starting the computer. While the program runs, there are no means of intervention. The computer executes the program and, while doing so, produces a set of symbols called the *output*, so when processing is finished, the user is left with two sets of symbols, the input and the output. Those two could be in the same form,

---

37 At this point, one can disregard that in the beginning a number of computers like the Colossus and the Zuse Z3 were not designed to be Turing complete.

38 Ceruzzi (1998), page 1

39 Ibid., pages 30, 31

e.g. on punched cards, or in a different form, e.g. when the output is supposed to be human-readable, in which case a typewriter or a fast printer was used.



Figure 4: The classic IPO paradigm

This IPO-style can be easily made out in the famous paper by von Neumann (1945)<sup>40</sup> who describes what was later called the von Neumann architecture. Von Neumann starts with a general definition of an “automatic computing system”:

“An automatic computing system is a (usually highly composite) device, which can carry out instructions to perform calculations of a considerable order of complexity.”

According to von Neumann, a lot of effort has to be put into the description of the instructions, which is what we today would call the program<sup>41</sup>, and the “numerical information”, which we today call input data. The combination of both thus is the I in the IPO paradigm.

“The instructions which govern this operation must be given to the device in absolutely exhaustive detail. They include all numerical information which is required to solve the problem under consideration: Initial and boundary values of the dependent variables, values of fixed parameters (constants), tables of fixed functions which occur in the statement of the problem.”

He then describes the means of input and the necessity to use certain codes which describe the problem in a formal way. Today we would call these codes the commands of the program.

“These instructions must be given in some form which the device can sense: Punched into a system of punchcards or on teletype tape, magnetically impressed on steel tape or wire, photographically impressed on motion picture film, wired into one or more fixed or

---

<sup>40</sup> von Neumann (1945), section 1.2

<sup>41</sup> The word program in today’s sense of the word was not yet common at the time as it, according to Grier (1996), only became widely used in the beginning of the 1950s.



exchangeable plugboards—this list being by no means necessarily complete. All these procedures require the use of some code to express the logical and the algebraical definition of the problem under consideration, as well as the necessary numerical material.”

Due to the fact that while the computer is processing the instructions, the P in IPO, no human intervention is possible and all data must be given in advance.

“Once these instructions are given to the device, it must be able to carry them out completely and without any need for further intelligent human intervention.”

When processing has finished, there are results which can be “recorded”. This is the output, the O in IPO:

“At the end of the required operations the device must record the results again in one of the forms referred to above. The results are numerical data; they are a specified part of the numerical material produced by the device in the process of carrying out the instructions referred to above.”

A von Neumann computer, which works without the need for “intelligent human intervention”, allows for an automatic processing of algorithms devised by intelligent beings. Such a machine is controlled by a formal arrangement of symbols which represents the program. Keil-Slawik (2000) calls such a computer a typographic machine, indicating it is a machine which is controlled by “typography” by which he means text in a wider sense. As the text which controls a computer is purely formal, it could be quite difficult to understand for a human being because, by definition, formal texts are free of semantic meaning while humans usually do not define a problem or a sequence of events as formal operations but interpret them in their semantic contexts. Early programming languages were only understandable if one knew the structure of the computer’s hardware very well<sup>42</sup>, as their commands directly referred to the inner structure of the machine.

Programmers who wanted to solve a problem using the computer had to know how many registers a machine had, how memory was organised and what

---

42 See Mahoney (2005)

command set the machine supported. By doing their programming, they effectively deprived the problem of its context thereby making it difficult to understand. Take the condition of “being broke” as an example which I want to “computerise” step by step. Still in the realm of normal language, one could be more precise and formulate “If one spends more than one earns, one is broke”. Formulated mathematically, one could write something like “*broke*:  $s > e$ ”. This description already looks very formal, which in fact it is, but notice that there still are hints to human semantics. The word “broke” as well as  $s$  for spending and  $e$  for earnings refer to the respective source and the target domains. Translating the condition into a basic computer language, one ends up with something like this<sup>43</sup>:

```
0: LOAD 2
1: SUB 1
2: JGTZ 4
3: WRITE "broke"
4: HALT
```

This way the program is broken down into a number of instructions which are written down line by line. The values for earning and spending are supposed to be stored in two registers of the supposed register machine. A register can be thought of as a memory slot for exactly one value. The value of the earnings is stored in register 1, while the spending value is stored in register 2. A register machine has one distinct memory slot which is called the accumulator. All calculating operations the machine can perform are applied to this accumulator. In line 0 the value of register 2 is loaded (i.e. copied) into the accumulator thus the value of register 2, the amount of spending, now is both in register 2 as well as in the accumulator. In line 1, the value of register 1 is subtracted from the value in the accumulator. The result of this calculation is automatically stored in the accumulator, so effectively the program so far has subtracted the spending value from the earning value and has stored the result of this subtraction in the accumulator. Line 2 contains a conditional jump which specifies that if the value in the accumulator is greater than zero (JGTZ for Jump if Greater Than Zero), the computer is to continue in line 4 of the program. If this was the case, the program would stop without performing any further operations as line 4 contains the halt command. In the other case, the program continues in line 3 with a printout of the

---

43 This is code for an assumed assembler for a register machine my colleague Dr Harald Selke uses in one of his classes at Paderborn University. Notice that in this example, the line numbers are part of the actual program.

word “broke” and then, like in the other case, terminates in line 4. Notice that to be even able to write this program you have to know a lot about how the machine works (such as registers and the accumulator), have to have a lot of things in mind (such as which value is stored in which register) and have to know how to program without any references to humanly understandable semantics.

Programming improved, as Mahoney (2005) describes, with the advent of higher programming languages. Their big advantage did not only lie in abstracting from the internal structure of the machine but also in linguistic aspects itself. Higher programming languages fulfil two roles at the same time. They are close enough to written texts humans normally read and write, so they are somewhat understandable by humans, while at the same time they are formally defined and thus processable by a computer. Notice, for example, how just the possibility of defining functions and giving them names which represent their functionality in the given context allows a human programmer to understand a problem while being completely irrelevant for the execution of the program. The above problem in a higher programming language could be written down as

```
var broke = (spendings > earnings);
```

Without having such a “common language”, Keil-Slawik (2000) points out, complex programs could not be created even though theoretically hardware-near programming languages or even programming by plugging cables could provide all necessary operations.

Hardware specific programming was not the only disadvantage when it came to programming early computers. One of the biggest problems, even when using higher programming languages, was the process of creating the program itself and the way it got into the computer, as the computer did not support creating or editing the program and its input data. Program and data had to be written out on paper and then had to be manually encoded and punched into cards or tape. In case a programmer made a mistake, even if it was just a missing symbol or a minor typo, a great amount of punching had to be repeated. When using punchcards and punched tape, symbols are written into the carrier in the most brutal way possible by mechanically punching holes into it. This, of course, makes a later manipulation of these symbols virtually impossible. While, using Keil-Slawik’s terminology, the computer was (and is) a typographic machine, a

machine being controlled by symbols and which itself creates symbols, the very symbols which control the computer, the program, could not be manipulated using the computer.

Only for very early computers like the Zuse Z3, which had no persistent means of storage, the program and all input data had to be read in directly before execution or at runtime<sup>44</sup>. Soon, persistent memory in the form of magnetic tape, magnetic drums, and magnetic discs was added, and first operating systems provided libraries and file systems, so instead of having to provide the same program segments over and over again, library programs were stored on the disk from where they could be loaded when necessary. As data could be saved on the disc, programs were able to manipulate these data or use them as the source of its evaluations. Instead of having to provide all data and each instruction at every program run, programmers could refer to predefined routines and to persistent data. On the one hand, this reduced the programming complexity of standard tasks, and on the other hand, it allowed for more complex programs to be even possible. The operation of databases is an example of the new potentials this persistent storage offers. Databases allow the evaluation of great amounts of data. Not having to input these data before every program run immensely improved performance. The database management logic itself is encapsulated in a library, so programmers can rely on this functionality without having to provide the code at every execution of the program. This way, programmers can concentrate on the business logic and consequently create more complex programs in that respect.

Persistent storage allowed for the manipulation of data inside the computer and allowed programs or program routines to be saved in the computer. Yet the programming itself still had to be done outside the computer as the machine did not provide an interactive interface which would have allowed the manipulation of program code using an editor, which is nothing else but a program which has a set of symbols loaded into memory and which reacts to user input by transforming it into manipulations of these symbols. The disadvantages of not being able to use the computer as a programming device became even more problematic when, due to commercial success and the need for more and more people to use a computer while writing ever more complex programs, the so-called batch operating mode

---

<sup>44</sup> As the Z3 did not yet store the program in internal memory, the program was running through the punched tape reader while it was running.

was implemented. Up till then, a human operator fed one program after the other into the computer and could intervene or inform the programmer of a faulty program as soon as a problem occurs. In the batch operating mode, programs which were initially written down on paper and then coded on punchcards were put into a relatively simple machine whose only purpose it was to copy them one after another onto a magnetic tape. This tape was eventually put into the main computer which executed the programs sequentially resulting in printed out feedback hours or even a day after the program was created. Programming this way suffered greatly from the indirectness and long waiting periods<sup>45</sup>.

The indirectness of computer operation and response was overcome with the introduction of time-sharing which main idea, according to Corbató et al. (1962), was having “a computer made simultaneously available to many users in a manner somewhat like a telephone exchange. Each user would be able to use a console at his own pace and without concern for the activity of others using the system”. This need for more immediate interaction with the computer was not driven by a need for interactive software in today’s sense but by the programming issues mentioned above. This is evident in Franchi (1975), who describes that an interactive environment offers “advantages in all computer installations where program development, as opposed to production work, is one of the main tasks”<sup>46</sup> because in these cases it can “reduce the detection and intervention delays” in finding errors without causing “a substantial increase in the use of resources”<sup>47</sup>. For the program to be editable using a computer, the program, of course, has to be in the computer in a manipulable form, which means the computer must be able to access a program as its data. This *stored program concept* which Ceruzzi (2012) describes as “programs and data [...] treated the same inside the machinery because fundamentally programs and data *are* the same”<sup>48,49</sup> was already given with the von Neumann architecture as described by Burks et al. (1946/1987):

---

45 See Corbató et al. (1962), who noted: “Using current batch monitor techniques, as is done on most large computers, each program bug usually requires several hours to eliminate, if not a complete day.”

46 Franchi (1975), page 121

47 Ibid., page 122

48 Ceruzzi (2012), pages 29,30

49 While the interactive operating mode was originally implemented in order to improve programming, for a “major advance in programming intimacy” as Corbató et al. put it, they also recognised potentials for the way the programs themselves are used: “Programs containing decision trees which currently must be specified in advance would be eliminated and instead the particular decision branches would be specified only as needed.”

“It is evident that the machine must be capable of storing in some manner not only the digital information needed in a given computation [...] and also the intermediate results of the computation (which may be wanted for varying lengths of time), but also the instructions which govern the actual routine to be performed on the numerical data. [...] Conceptually we have discussed [...] two different forms of memory: Storage of numbers and storage of orders. If, however, the orders to the machine are reduced to a numerical code and if the machine can in some fashion distinguish a number from an order, the memory organ can be used to store both numbers and orders.”

From today’s perspective the stored program concept can be seen as the greatest achievement of the von Neumann architecture, yet for the protagonists at the time, it seemed “evident” to a degree it was not even explicitly mentioned in von Neumann’s original paper. Von Neumann and his colleagues initially proposed the stored program mainly for speed reasons but also recognised it provided the possibility for programs to configure themselves, at least to a small degree<sup>50</sup>. Using the computer for programming, which means using one program in order to create and manipulate *another* program, was nothing they had in mind at the time, as doing so not only requires the program which is to be edited to be in memory, but it cannot work “without intelligent human intervention” because this intervention is the very purpose of an editor which, as a consequence, necessarily has to be interactive.

---

<sup>50</sup> See Ceruzzi (2012), page 58

## 2.4 Characteristics of Programmed User Interfaces

The step from a symbol transforming machine to an interactive device which constantly awaits user input and reacts to it, despite some prior prototypes which I speak about further down, coincides with the switch from batch processing to time-sharing operation which, as described above, was introduced due to increasing problems of programming without computer support

### 2.4.1 Responsiveness

Time-sharing, as defined in Ceruzzi (1998), means the computer is continuously running and sharing its resources among many clients, so “each user [has] the illusion that a complete machine and its software [is] at his or her disposal”.<sup>51</sup> The purpose of this kind of operation is that the computer becomes *responsive*. Instead of having to wait for several hours in order to get a response, with time-sharing it is a mere question of seconds or even less. Instead of having to provide all input data and thus having to make all decisions in advance, programs can now be programmed in a way they await user input while they are running. Early interactive time-sharing systems used an electronic typewriter or a teletype machine as its input and output device<sup>52</sup>. This choice of equipment resulted in the command line operating mode, which is still known today. In this mode, the computer processes textual user input made on a typewriter as soon as the user hits the enter or line-feed key. Later incarnations replaced the bulky typewriter with keyboards and fast printers or consoles equipped with keyboard and screen, yet the command line operating mode itself kept the characteristics of the early teletype-based solutions, which is why screen based command line operation is sometimes humorously called “glass-teletype”<sup>53</sup>.

With the switch from batch processing to on-line operation, programs became interactive, so computer systems had to provide output and had to allow for related input to be a possible control command for themselves, so with interactivity came the need for an explicit interface. Just like interactivity, the word *interface* can be (and has been) interpreted in numerous ways.

Janlert&Stolterman (2015) looked into different interpretations of the term and

---

51 Ceruzzi (1998), page 154

52 see also Corbató et al. 1962

53 See Ceruzzi (1998), page 251

came up with a number of different “thought styles”<sup>54</sup> of which “means for controlling an object” seems to be the best fitting interpretation in the sense of the argumentation of this thesis as it describes the interface as a kind of mediator between the machine and the user:

“The interface becomes a manifestation of what can be done with the machine. The interface controls functionality and is governed by the intention of the user. The user is ‘using’ the machine to achieve something by manipulating and controlling the interface.”<sup>55</sup>

Notice that Janlert&Stolterman’s definition of interactivity, “controlling an object by manipulating and controlling the interface”, can only be applied to the “chrome” part of an application but cannot adequately describe the objects of the content which nevertheless are part of the user interface. Interpreting the manipulation of these objects as means of controlling the machine would reverse the purpose of the user interface in this respect. The content objects are the object of the users’ work or what the users are interested about. When they, for example, use the computer to compose a doctoral thesis, the object manipulations which are performed in order to do that, the creation and arrangements of letters and graphics, cannot be sensibly thought of as a means of controlling the computer. On the contrary, the structures and programs in the computer are a means of providing the objects.

In the following sections, I will show that the interface can be interpreted as that part of a computer system which generates the world of virtual objects, which from a user’s point of view are the objects of manipulation independent of whether they are part of the GUI chrome or of the content. In this perspective, an interface is not only, as Janlert&Stolterman (2015) put it, a manifestation of what *can be done* with a program but also and foremost a manifestation of what a program *is*, in terms of what can be seen on the screen, and which objects it provides.

---

54 The full list is: (1) a surface of contact between matching objects (from the tradition of industrial machine making); (2) a boundary of an independent (self-sustained) object (from biology and traditional artifact design); (3) a means for controlling (operating, checking, steering) an object (from design of complex machines); (4) a means for expressions and impressions, a target of interpretations and affectations (from human communication, architecture, and art).

55 Janlert&Stoltermann (2015), page 514



### 2.4.2 Virtual Objects

Interactive computer systems are constantly running and awaiting user input. This user input is quite different from what I called “the input” in von Neumann’s sense when speaking about the old punchcard-based machines. In an interactive system, the main type of input is not raw data but manipulation commands. Manipulation commands must have a target, so with interactive interfaces necessarily arises a concept of manipulable entities which I want to call *objects*<sup>56</sup>. Objects are provided by the interface explicitly in order to be perceived, addressed and thereby referred to by a command. While operating systems for batch operated computers already supported file systems which provided file-objects and folder-objects, or databases which provided datasets, in interactive command line interfaces those became available not only for the programmers but also for the users<sup>57</sup>. A user can refer to an object like a file, a directory, a database entry, a character, a line, or a paragraph, which is constructed by software and is part of a developer-programmed, computer-generated user world. In this thesis, I use the term object whenever there is such a programmed entity created for the user. Notice that when speaking about objects I am not referring to object-oriented programming. While there are certain similarities between the two concepts, virtual objects as described here are a construct of the “world” users perceive through the interface, not the structures programmers chose to use to structure their code. Virtual objects can be found in computer systems which have been programmed in object-oriented programming style as well as in others which have not.

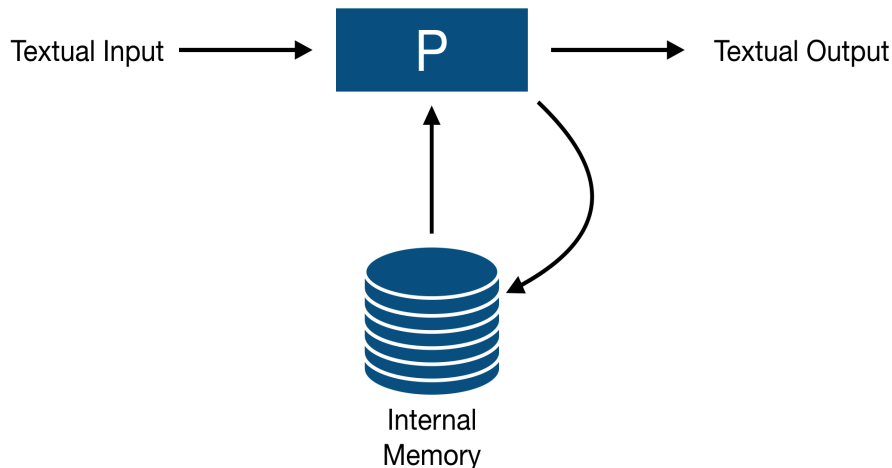
The most important characteristic of virtual objects is that they hide the circumstances of their existence. When using the computer, a document appears to be an object, a virtual artefact, yet if one took the computer apart, one would not find the document. It only exists because there is a program which makes it exist for the user to perceive and manipulate, so from a user’s perspective there is a persistent document. Users do not have to care about how the letters in the

---

56 The concepts of files, directories, database entries, or paragraphs were already known in command line interfaces but were not called objects at the time. Still, in the context of my concept of interactive user interfaces, even at this early stage object characteristics can be made out, even though they were not yet fully exploited.

57 I assume that “before the interface” there was no distinction between a user and a programmer, yet this question would require a historical review of sources, which I do not want to pursue here.

document are translated into numbers, how those numbers are translated into voltages, and how those voltages are translated into magnetisations of the surface of a magnetic disk. For them, when they start the computer, the document appears as nothing but a document, so the user perceives the computer system as a stable environment of persistent, yet manipulable and well-defined objects.



*Figure 5: A schematic view of the command line interface mode. Textual input is directed at a central processing entity (the command line interpreter), which also provides all output. Commands executed by the command line interpreter have access to an internal memory which is invisible to the user. Individual commands of the command line processor are responsible for the interpretation of parts of the internal memory as objects which can be manipulated by applying commands onto them.*

While the command line mode provides the users with virtual objects they can refer to, their access to this world is indirect at best. The objects are not directly visible and, therefore, cannot be directly manipulated either. Users can only enter commands into a console and expect responses from it (see figure 5). In a command line based user interface, like an MS-DOS or a Unix shell, there are objects, like directories and files, which are not directly perceivable by the user. Instead of that, the user has to ask for them by invoking a command like “dir” or “ls” which makes them visible by listing them. One cannot directly manipulate an object but has to describe the desired manipulations verbally. As action and perception take turns, the screen is filled with a kind of record of the interaction. The input never manipulates what is seen on the screen. If a directory listing reveals a file “resr.txt” and one calls “mv tesr.txt test.txt”, the initial listing

does not change, so in order to be sure about the outcome of the command, another directory listing has to be requested.

Imagine one wanted to edit a text using a command line interpreter. One had to write commands in order to manipulate a text one is not even seeing. Such editors actually do exist. The BASIC interpreter of the Commodore 64, for example, worked that way<sup>58</sup>. Take the simple example of a program which should print out “COMMODORE” five times. When the Commodore 64 was turned on, one could directly start programming by specifying the program line by line<sup>59</sup>:

```
10 FOR I = 1 TO 5
20 PRINT "COMMODORE"
30 NEXT I
RUN
```



Figure 6: Editing a program on a Commodore 64 (colours adapted for readability)

As depicted in figure 6 the program does what is expected. If one wanted to change the program to print out “PADERBORN” instead of “COMMODORE”, one had to change line 20 which could be done by redefining it.

```
20 PRINT "PADERBORN"
```

As the screenshot in figure 6 shows, redefining line 20 of the program does not alter the display of the original line 20 at the top of the screen. In order to check the current state of the whole program, calling a LIST command triggers a printout of the whole program including the changed line 20 (as depicted on the right-hand side of the figure).

<sup>58</sup> See Commodore (1982), chapter 3

<sup>59</sup> Again, in this example the line numbers are part of the actual program.

While the Commodore 64 is, of course, a system of the past, such editors still exist today. The Unix editor `ed` is an example of that. In the example below<sup>60</sup>, a text file is read and displayed on the screen. After that, two changes are made. First, some plusses are inserted into the second line and finally, in what then will be line 3, the misspelt word “hire” is changed into “here”. After checking if everything went according to plan by having the text displayed a second time, the file is eventually saved to disk, and the editor is quit.

(Lines starting with a `>` are those lines the user types in)

```
> r textfile.txt
85
> ,1
This is the heading.$
The text starts hire. There may be many important things to say.$
> 2i
> ++++++
> .
> ,3
The text starts hire. There may be many important things to say.$
> 3s/hire/here/
> ,1
This is the heading.$
+++++$
The text starts here. There may be many important things to say.$
> w textfile.txt
108
> q
```

As editors like `ed` and other pure command line interfaces are “glass teletype” style, editing a text using them is somewhat complicated because it is not possible to manipulate the actual letters or lines of the text<sup>61</sup>. Instead of that, one has to know a number of commands which are typed in in order to have the program do the changes<sup>62</sup>.

---

60 The example is inspired by an example in the article to “`ed`” in the English Wikipedia.

61 On the Commodore 64, this was partially possible, as the system allowed the users to use the arrow keys to move the cursor from line to line. It was possible to invoke the list command, go the line which was to be edited, do the manipulations and redefine the line by pressing the enter key just there.

62 The reason why editors such as `ed` still exist today is that they can be scripted. It is possible to write commands into a text file and then start `ed` providing a text file and the command file. This way, it is possible to apply the same text manipulations to a number of text files without having to manually edit all of them separately.

Summing this up, interfaces based on command line interpreters have quite a number of disadvantages, which makes their use feel overly technical and complicated from today's point of view. Both objects and commands have to be remembered or "requested" as they are not constantly visible on the screen. In order to manipulate an object, one has to give a command to an interpreter, which then manipulates the object. The result of such operations is not immediately visible, so one has to constantly invoke a command which prints out the current state<sup>63</sup>.

### 2.4.3 Spatial Objects and Direct Manipulation

At least with the availability of graphical screens, it became possible to constantly display the object of manipulation on the screen. While there are a few examples of user interfaces in which objects were constantly visible, yet manipulations were still made by entering commands into a distinct command area, spatial objects on the screen rise to their full potential when they can also be spatially manipulated. An interface which consists of spatial objects which can be manipulated at their spatial location is called a *direct manipulation* interface. The term direct manipulation was coined by Shneiderman (1983) in order to describe the operating mode of systems which instead of providing a command line interface rely on simulated objects on the screen which could be manipulated by physical action. Shneiderman described two technical requirements<sup>64</sup>, which are quite straightforward:

- Continuous representation of the object of interest.
- Physical actions (movement and selection by mouse, joystick and touch screen, etc.) or labeled button press instead of complex syntax.<sup>65</sup>

While Shneiderman emphasised on physical actions instead of verbal descriptions, Johnson&Roberts (1989) point to the important aspect of the role of space in

---

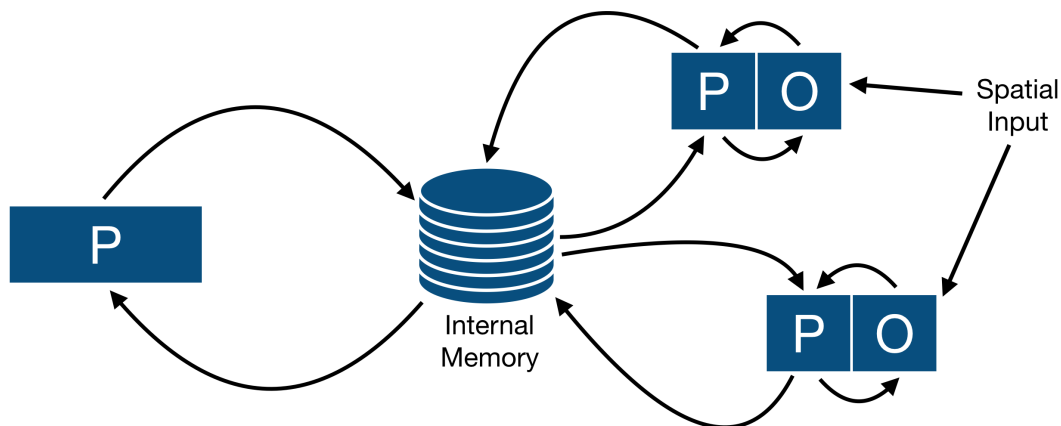
63 These problems are even worse with voice assistants like Amazon's Alexa, which do not even have a display. Users of such systems have to remember the objects they refer to without having them visually present.

64 Shneiderman (1983), page 64

65 In his enumeration, Shneiderman added reversibility as a third requirement, which in my point of view is not a genuine property of direct manipulation interfaces. Of course, reversible operations encourage explorations and help to avoid fatal errors and thus are beneficial, but that is true for all interface styles.

direct manipulation. This aspect is central to the distinctions I make in this chapter, as it makes the virtual object a spatial object, which is not only displayed at a certain location but also can be manipulated by referring to this very location:

“Direct manipulation requires that distinct functions be invoked and controlled in spatially distinct locations, in ways that are specific and appropriate for the particular function being controlled. Data objects should be selected and operated on by simulated physical contact rather than by abstract verbal reference[...] Conventional interfaces are indirect in that there is a single general interface to all functionality (such as a keyboard and a command language or a menu). In other words, there is only one input channel for all kinds of input; different kinds of input are distinguished linguistically rather than spatially.”<sup>66</sup>



*Figure 7: The operating mode of a direct manipulation interface. Input is made directly to the objects on the screen, which are spatially selected and can then be manipulated. Any change in those objects is directly visualised, as objects have their own presentation logic.*

Figure 7 shows a schematic view of this operating mode. Instead of having to direct all manipulation input to a single instance, verbally describing the manipulation of an object and referring to it by using its name or address, objects can be spatially selected at the position they are being displayed. In direct manipulation interfaces, the object of perception is the object of manipulation, so the locus of manipulation also becomes the locus of attention. Each object is, so to

---

<sup>66</sup> Johnson&Roberts (1989), page 14

speak, its own input and output device and can be manipulated in place. For that to be possible, spatial input devices are needed which either allow a direct spatial selection of an object by using some kind of direct pointing technology (e.g. a pen or a touch interface), or which provide a distinct pointing object which can be positioned using a joystick or a mouse.



*Figure 8: A light gun in use at a terminal of the SAGE Computer system (Photo from Astrahan&Jacobs 1983)*

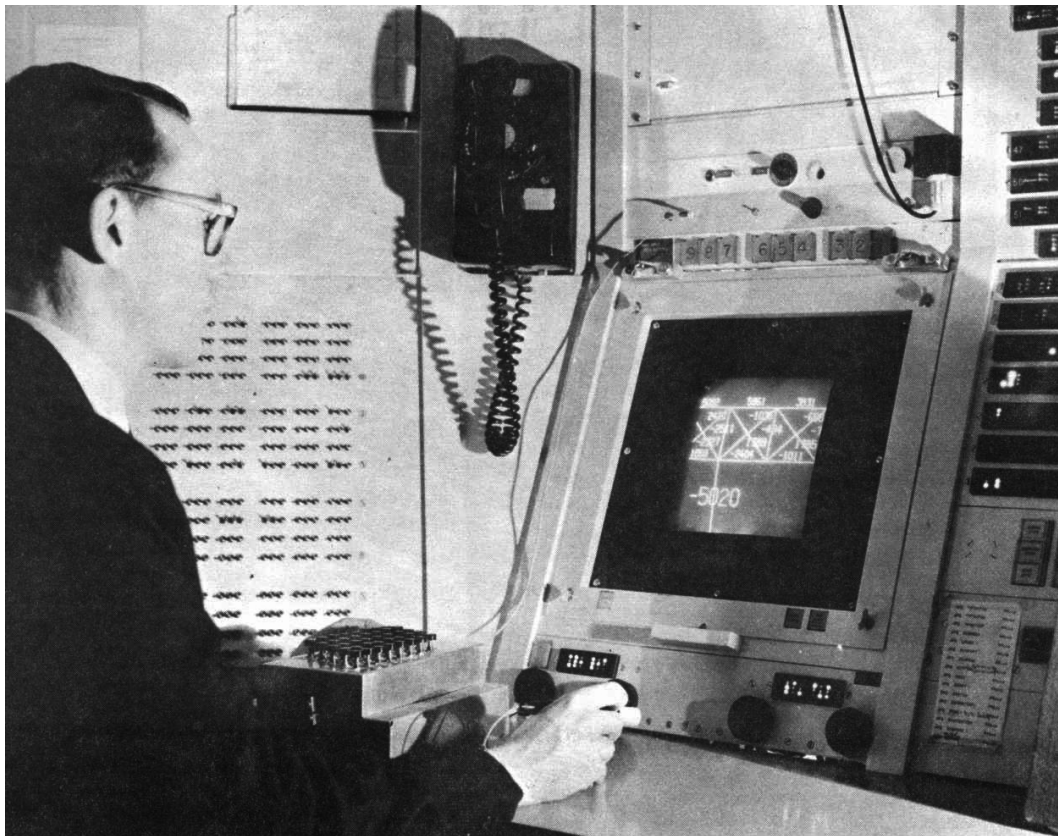
A look into computing history shows that the idea of selecting an object on the screen in order to direct commands to it is older than one might think and even predates the time-sharing ideas. The earliest occurrence I came up with is the semi-automatic system for air-defence called SAGE<sup>67</sup> which was developed in the 1950s and 1960s. It incorporated a viewing device which in form and function resembled a radar screen. This screen presented filtered information about the air traffic situation showing only those planes or flying objects the system could not

<sup>67</sup> Semi-Automatic Ground Environment (also called the Cape Cod System), see Astrahan&Jacobs (1983) and Wieser (1983)

## 2 Interactivity and the Interface

---

bring in correspondence with saved data about planned air traffic. As can be seen in figure 8, operators could use a so-called light gun to select an object on the viewscreen. This triggered an additional text screen to show information about the selected object and allowed the operator to invoke further commands which ranged from displaying a track projection to suggesting the launch of an intercept missile.



*Figure 9: “Sketchpad in use. On the display can be seen part of a bridge. [...] The author is holding the light pen. The push buttons draw, move, etc., are on the box in front of the author.” (Photo and description from Sutherland 1964)*

An early civil example of a direct manipulation interface was the Sketchpad project by Sutherland (1964) which was a predecessor of what today would be called a computer-aided design (CAD) system (see figure 9). The Sketchpad system introduced many interesting concepts which are still used in today’s user interfaces and graphics systems. For example, users were able to create and



manipulate drawings using a light pen<sup>68</sup>, which regarding functionality was a device quite similar to the light gun of the SAGE system. Parts of the drawing could be selected and manipulated, e.g. by dragging the start or end points of lines into other positions. During this operation, the line was constantly repainted, resulting in the impression of actually manipulating the line on the screen.

Computer systems which implement direct manipulation wait for user inputs, apply them to the currently selected object and constantly and rapidly refresh the objects affected by it. If all evaluation and repainting operations occur rapidly without any noticeable delay and if the programming is done in a way that manipulations and transitions appear to be smooth without any disruptions like flickering or faltering, the user gets the illusion of directly manipulating the objects on the screen.

This potential of manipulable objects on the screen, which from today's point of view appears to be almost ordinary because it is used every day when dragging around file-icons, arranging windows on a screen, or using a modern word processor, from a media point of view is nothing but revolutionary. It is a characteristic of classical inscription media, as the name says, that something is inscribed into them, which means it is inscribed into the carrier which makes the symbols very persistent but at the same time limits the possibilities of manipulations. While it is possible to add new symbols to an existing writing and partially remove symbols using mechanical and chemical means by wiping them out, scratching them off, or using chemical ink removers, it is not possible to change and arrange symbols which have already been written. A circle painted on a blackboard cannot be moved around, it can only be wiped out and repainted at another position. If one wanted to change the order of two paragraphs in a text, like I did quite a few times while creating this thesis, one is inevitably confronted with media discontinuities, which means one either had to cut the typed text into pieces, arrange them so that they are in the desired order, and agglutinate them, or one had to rewrite the text altogether. The term *media discontinuities* in the sense I use here has been described in detail in Keil et al. (2006) and by Hampel&Keil-Slawik (2001). Simply put, such media discontinuities occur when instead of

---

68 The Light Pen was presented by Stotz (1964), who also describes the development of input-output systems, which are small computer systems which allow character or line display on a screen without requiring the main processor of a computer to calculate the output images.

being able to focus on the symbols one wants to manipulate, one has to deal with the media carrier in order to perform the necessary manipulations.

In analogue media, symbols can never be moved around on their carriers. If one needs the flexibility of such arrangements, this is only possible by slicing the carriers into many small parts or using small carriers from the beginning. This way the arrangement is at least partially flexible, but the advantage of persistence is diminished as the arrangement is not inscribed, so using analogue inscription media, one always has to decide between the flexibility of an arrangement of media carriers and the persistence of inscriptions into a carrier. If both are needed, additional effort has to be made<sup>69</sup> resulting in media discontinuities. All of this changes when using digital media. As a consequence of the technical potentials of digital interfaces, in digital media objects can be both persistent as well as manipulable. When in a digital manipulation system an object is moved, from the perspective of the user the object itself is manipulated. When the manipulation is over, the object becomes a stable, persistent object without any need for glueing or another form of manual fixation. Looking back at the initial reasons for introducing interactive systems, improving programming by shortening the response times can only be seen as the first step. At least as important is the possibility of being able to manipulate the source code at the position it appears on the screen. Productive source code editing, in this sense, needs more than just a command line interface but relies on a cursor and the direct manipulation of text it provides.

Please notice that the way I introduced direct manipulation in this thesis makes the (virtual) object on the screen the centre of all considerations. In this interpretation of direct manipulation, what the users manipulate are the objects of interest. This is a more technical and more artefact-based interpretation of the concept than the one given by the protagonists of direct manipulation interfaces, who emphasised on the possibility of an object to refer to objects or conditions in the target domain of an application. Shneiderman spoke about “comprehensible action in the problem domain” while Hutchins et al. (1985), who used the term “direct engagement” instead of direct manipulation, spoke about being “engaged [...] with semantic objects of our goals and intentions”<sup>70</sup>. According to Shneiderman basing

---

69 One could, for example, take a photo of an arrangement, collect the parts and later reconstruct the arrangement using the photo.

70 Hutchins et al. (1985), page 318

the object on something outside the computer system even seems to be of the utmost importance: “The trick in creating a direct manipulation system is to come up with an appropriate representation or a model of reality”.<sup>71</sup>

While thinking about models of reality and thus finding objects and conditions in the target domain of an application is a good suggestion for creating concrete, direct manipulation interfaces, for my purpose of determining technical potentials of interactive user interfaces such a reference to a “reality” outside the computer is less important. Of course, virtual objects *may* correspond to something in the target domain and thus exhibit behaviour appropriate to the target domain, but neither is this new in respect to direct manipulation interfaces<sup>72</sup> nor is such a correspondence necessary in every case. For the technical description of the interface I put forward in this thesis, it does not matter whether or not the objects refer to something. A circle may just be a circle and can still be subject to direct manipulation.

#### **2.4.4 Responsive Manipulation**

When Wegner (1997) formulated advantages of interactive programs in contrast to mere algorithmic programs, he compared virtual objects with robots and stated that “objects and robots have similar interactive models of computation; robots differ from objects only in that their sensors and effectors have physical rather than logical effects”.<sup>73</sup> While Wegner wanted to describe the advantages of a system which allows user intervention in contrast to a classical purely algorithm-based computing approach, his thoughts about objects behaving like robots, having sensors and actors, very well describe what is possible through responsive manipulations. Objects, like robots, can be programmed to sense their environment and react accordingly. Users of a system with such objects find themselves in a world of active objects which exhibit a kind of ‘behaviour’.

In chapter 2.3, I described formal symbol arrangements as the theoretical and practical foundation on which computer operations are based. Direct manipulation allows users to perceive and manipulate objects spatially, thereby manipulating

---

71 Shneiderman (1983), page 66

72 Imagine a command line appointment calendar. Objects of this calendar would, of course, be appointments, so they could be called an “object of the target domain” and thus would “behave” accordingly, e.g. by “knowing” about bank holidays.

73 Wegner (1997), page 83

## 2 Interactivity and the Interface

their arrangements. With this possibility, we come full circle as this, of course, means that the spatial arrangement created through direct manipulation can, in turn, be evaluated by the computer system. As I described in chapter 2.4, the main reason for the development of interactive user interfaces was the improvement of the programming process by providing the programmer with the means of manipulating the object arrangements which constitute a program using the same machine the program later runs on. While in the early days, these symbol or object arrangements mostly were texts, the processing of objects is in no way restricted to letters, so arrangements can well consist of graphical elements. The Scratch graphical programming language<sup>74</sup> (see figure 10), for example, which was developed at the MIT with the explicit purpose of providing an easy introduction to programming, provides a number of graphical building blocks which can be used to literally piece together a program.

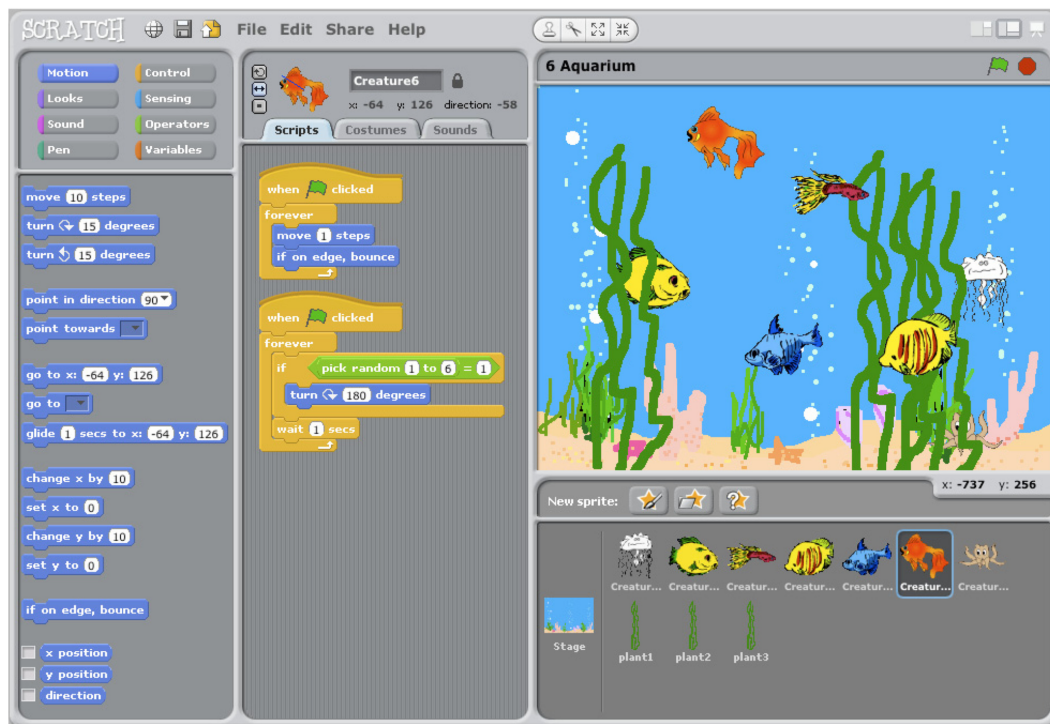


Figure 10: The Scratch user interface (image from Maloney et al. 2010)

Object arrangements which are processed by a computer do not necessarily have to be programs in the typical sense of the word. The spelling correction function

<sup>74</sup> see Maloney et al. (2010)

of a word processor, for example, uses written text as an input and comes up with suggestions according to it, yet one would not call the text of a letter a program while, strictly speaking, this text, of course, controls the behaviour of the spell checker quite as in Scratch a graphical program controls the behaviour of an interpreter. There are, of course, differences, the biggest one being that the spell checking function of a word processor evaluates the arrangement of the letters and as a result manipulates these very letters, e.g. by underlining them. In programming, on the other hand, the execution of a source code typically does not manipulate this very source code in any way.

	<b>non-reflective</b>	<b>referring</b>	<b>reflective</b>
<b>explicit</b>	Execution	Evaluation	Transformation
<b>implicit</b>		Responsive Evaluation	Responsive Manipulation

*Table 1: Object processing depending on degree of reflexivity and implicitness*

Table 1 shows a suggestion for different terms for the processing of object arrangements depending on implicitness and reflectiveness. By providing them, I do not want to introduce a definitive vocabulary but rather want to point to the subtle differences between the way processing is triggered as well to what extent the initial objects are referred to during processing. These differences have a strong influence on whether or not, from a user's perspective, an application seems to be active or passive and whether or not reactions on manipulations are attributed to a program running on the computer or to the object arrangement itself.

When executing a program, a set of symbols is processed. This act of processing happens explicitly by a user who “runs” a program. While the program runs, a multitude of things might happen, yet in most cases, the running program does not manipulate those objects which are the basis for its current operation. It does not change its own programming. In my nomenclature thus *executing* a program is an explicit, non-reflective kind of processing. This is, at least in the sense of this thesis, the least interesting kind of processing as the interactive capabilities lie in

the other modes of symbol processing which I explain using the examples of both editing normal text as well as using an integrated programming environment.

When editing text, a user takes advantage of the object manipulation capabilities of a digital system. Imagine editing a text by creating a new set of characters having the cursor placed somewhere in the middle of a line. The newly created characters do not overlap the existing ones but drag them to the right. If due to this movement the line now grows too long, one or more words automatically move into the next line. While technically the act of typing in the new word consists of a number of manipulation commands which are processed together with the existing text and result in an updated object arrangement, the potentials of digital media are combined in such a clever way that, from a user's point of view, the movements of the letters and words seem to be a property of the letters, words, and lines themselves rather than of a process running in the editor. The application, which is constantly evaluating the attribute arrangements and the input, virtually disappears, so one would not attribute the described behaviour to the program but to the objects themselves. This is what, according to the nomenclature in table 1, I call *responsive manipulation*, where processing is not explicitly started but is rather triggered implicitly and where the process reflectively results in a manipulation of the object ensemble it used as its input<sup>75</sup>.

Responsive manipulations are, of course, a reflective kind of processing. The object arrangement is processed, which results in a manipulation of this very arrangement. If such a reflective process is started explicitly, I speak of a *transformation* of the object arrangement. A pretty printing operation<sup>76</sup> in a programming editor would be such a transformation. It is not triggered by some input event but rather has to be explicitly called. It takes the existing source code

---

75 Nowaczyk (2005), referring to Keil's concept of the typographic machine, described "Responsive Typography" in which "every manipulation of a symbol in the arrangement is a control command for the typographic machine" (page 61). Making every *manipulation* an input is, what enables object evaluations to be triggered implicitly and thus enables responsive behaviour.

76 Rubin (1983) describes a pretty printer as "a software tool used to format programs to make them easier to read and understand. A pretty printer takes character strings, called tokens, from a source, usually a text file or a parse tree, and prints them with aesthetically pleasing spacing and line breaks. The primary functions of a pretty printer are to insert spaces and linefeeds between tokens and to decide where and how to break lines that are too long to fit on the output medium."

as its input and transforms it into a well-formed structure, e.g. by inserting blanks and indents.

Assume a programmer has finished editing a program. He clicks a run button which triggers the execution of the program. In some cases attempting to execute the program this way will result in an error message revealing a syntactical error in the source code. This error message is the result of an *evaluation* of the source code. The process which created the error message has been explicitly started. It took the source code as its input, and instead of executing it or changing it, it created a new set of symbols, the error message, which refers to the input symbols by providing the line in which the error is detected. The frustrated programmer now has to re-enter the editing phase where he has to find the position of the error and correct it according to the error message.

While sometimes syntax errors are revealed this way, modern integrated programming environments have means of giving the programmers feedback about their source code, which would allow them to discover such an error before trying to run the program. An on-line error checking function, for example, constantly checks the source code for syntactical correctness and points to errors by writing them into a visible log area within the programming editor. Such an error checker does not have to be started explicitly but is an implicit process which refers to the input objects. I call such a process a *responsive evaluation*. If an error checker highlighted an error directly at the position it occurs, e.g. by displaying a red exclamation mark in front of the faulty line of code, this could be interpreted as a manipulation of the object arrangement which would make it a kind of direct manipulation. As those markings are not a part of the source code though, they are not subject to interactive manipulations, because they cannot be manipulated together with the source code, so I would rather consider this to be a spatially coupled responsive evaluation than a genuine manipulation of the input objects. The same is true for syntax highlighting which marks up keywords, strings or statements. It is not a function that is called explicitly at a specific time, but it is running all the time, so the programmer has constant feedback about the syntax of his program. While the syntax highlighting changes the source code symbols by colourising them, I still consider this to be a responsive evaluation rather than a responsive manipulation because the syntax highlighting is not a

subject of the interaction. One cannot change the highlighting but can only change the reasons which lead to the colourisations.

In this chapter, I elaborated on technical properties of the user interface which also are technical dimensions of interactivity. Making use of these technical properties allows for user interfaces in which object arrangements can be constantly evaluated by a computer system and in which the result of this evaluation, in turn, becomes perceivable in a visible change within these objects. It is quite obvious that space plays an important role in such interactive user interfaces. On the other hand, considering the initial claim of creating a user interface technique based on positioning objects in relation to structured space, one has to conclude that the potentials of user interfaces identified in this chapter support such an interface technique, yet typical classic interfaces do not make use of this potential.

In the following chapter, I will show, how space is used in knowledge work scenarios. It will become clear that in these scenarios, space is purposefully structured in order to make it a means of differential experience. Knowledge workers create structured spaces in order to visualise object properties which otherwise would not be perceivable. Positioning objects in relation to such structures allows object properties to be visually grasped and, depending on which property is depicted, to be spatially manipulated. I will argue that this characteristic of structured spaces, being able to reveal access to non-visible object properties, can be transferred to the area of user interfaces which in many scenarios serve a similar task of providing access to otherwise non-visible properties of virtual objects.



### 3 Differential Experience and Meaningful Use of Space

Modern graphical user interfaces such as the so-called desktop user interface make use of potentials of space which are well known in areas outside a computer. On a digital desktop, space is mainly used to discriminate objects. This discrimination allows users to select single objects and perform operations on them. A similar technique is used when, for example, spreading out photos on a table in order to simplify the process of finding a certain photo and selecting it, e.g. in order to put it into a photo album. Elements in a desktop user interface can be dragged around and, as a consequence, can be organised by putting them into different containers such as folders or hard disk drives. Objects can also be put onto other objects in order to do something with them. They are, for example, deleted by dragging them onto a trash bin icon. Such an operation can be compared to putting a document into a real-world trash bin or a shredding machine. These examples show that in modern user interfaces, such as in said desktop user interface, objects can be accessed and modified spatially. Yet, when comparing the use of space in user interfaces with other areas where space plays an important role, the computer falls behind, partially because the amount of space on a computer screen is a limiting factor, but mainly, and most importantly in the context of this thesis, because current user interfaces do not make use of the potential of space being structured.

In this chapter, I examine the role of space in knowledge work. I understand knowledge work as that kind of work which knowledge workers<sup>77</sup> are performing when handling the artefacts they use for actions like doing research, specifying things, teaching, or managing a complex process. Knowledge work, in the context

---

<sup>77</sup> “Knowledge worker” can be understood as an umbrella term, so I do not have to distinguish between, e.g., a researcher, a teacher, a student or an engineer. All of these people are knowledge workers, people who work with their knowledge instead of getting something built or fixed using muscle power like, for example, a bricklayer. The term is based on Drucker (1959) who defined knowledge workers as “not ‘workers’ in the traditional usage of the word but who labor productively as technicians, professionals, managers” (page 67). In Drucker (1973) he substantiated this definition by describing a knowledge worker as someone “who puts to work what he has learned in systematic education, that is, concepts, ideas and theories, rather than the man who puts to work manual skill or muscle” (page 28).

of this thesis, is not considered regarding the associated cognitive processes but is interpreted from a media perspective. In knowledge work, space is used for many purposes from being a mere holding area to providing explicitly created structures which can be used as a means of gaining new insights into the properties of knowledge artefacts (such as documents, graphics or post-it notes) as well to depict and develop connections between them. Knowledge workers need to actively structure their spatial environment in order to keep track of things or to learn something new about them. This chapter examines to what extent spatial structures are used in different areas of knowledge work. As a theoretical foundation, I first define the notion of differential experience. Without such a differential experience, knowledge work is not possible as it relies on being able to perceive properties of an object or process and to compare those perceptions with expectations about them.

## 3.1 Differential Experience

Conceptually, the term differential experience (“Differenzerfahrung”), as explained in Keil (2010), can be traced back to Keil-Slawik (1990). Referring to Gibson (1986)<sup>78</sup>, who explains one cannot discover new properties of something one only imagines, Keil-Slawik points out that thinking does not take place within the head but by using the head<sup>79</sup>. Being able to perceive the outside world is a necessary requirement for all thought processes because thoughts need to relate to the outside world. A thinking process needs to be based on something in a perceivable environment or, as Arnheim (1969)<sup>80</sup> puts it: “unless the stuff of the senses remains present the mind has nothing to think with”. Of course, there can be isolated thought processes which do not immediately refer to the outside world but eventually, thinking has to refer to something outside one’s own mind. Wittgenstein (1953/1958) formulated similar conditions as part of his “private language argument”. He described the problems which arise when trying to justify something which only exists in imagination and which cannot be compared to anything outside one’s own mind. He states that “justification consists in appealing to something independent” so, on the question whether the correctness

---

78 Gibson 1986, page 257

79 There is a pun in Keil’s German catchphrase “Das Denken findet nicht *im* Kopf sondern *mit* dem Kopf statt!” which gets lost in translation. A literal translation would be “Thinking does not take place *in* the head but *with* the head.”

80 Arnheim 1968, page 1

of the departure time of a train could be checked just by remembering the time-table in one's mind, he contradicts that...

...“this process has got to produce a memory which is actually correct. If the mental image of the time-table could not itself be tested for correctness, how could it confirm the correctness of the first memory?”<sup>81</sup>

According to Wittgenstein, Keil-Slawik, Gibson, and Arnheim, in order to justify what is in one's mind, one needs to refer it to something outside of the mind, “to appeal to something independent” of one's assumptions, opinions and prejudices. One has to be able to compare one's assumptions about a condition with something in the perceivable environment. Seeking differential experience means engaging in perceptions which could not only confirm one's assumption but rather can be surprising, thus contradicting one's expectations<sup>82</sup>. One cannot be surprised by something one is just imagining because what it just imagined, by definition, can never be independent of the person who imagines it, so as the basic prerequisite for gaining differential experience...

1. ... one has to be able to perceive something outside one's own mind to be able to be surprised by it so, in order to gain differential experience, it is necessary to use one's senses.

According to Hollan et al. (2000), humans are “spatially located creatures”<sup>83</sup>, who “must always be facing some direction, have only certain objects in view” and are “within reach of others”. Any form of organised differential experience, therefore, relies on the possibilities of being able to move oneself in the environment or on manipulating the objects which happen to be in that environment, thus to spatially organise one's environment, so...

2. ... one has to be able to change the perspective on what one can see. This either can mean moving oneself, or it can refer to a spatial manipulation of

---

81 Wittgenstein (1953/1958) number 265, page 93

82 In his definitions of differential experience, Keil speaks about verifying or falsifying hypotheses. As not every perception can be directly associated with the formulation of a hypothesis, I chose to rather speak about the possibility of being surprised, a notion which I borrowed from Esposito (1993).

83 see Hollan et al. (2000), page 190

### 3 Differential Experience and Meaningful Use of Space

---

the perceived object, e.g. by turning it upside down in order to see parts of it which were not visible before.

Being able to manipulate the environment allows further methods of gaining differential experience, which go beyond the aforementioned change of perspective. In order to gain insights about an object, it is often necessary to compare it with another object such as, for example, if one wants to find out if one object is bigger than the other. Getting this differential experience is only possible by either bringing both objects into close proximity, so the sizes can be visually compared, or indirectly by comparing both of them with a common reference object. In both cases,...

3. ... one has to be able to arrange objects in one's environment in order to bring them into the perception space and thus to allow comparisons between them.

Without technical aids, getting differential experience is limited, e.g. because something is too small or too big, happens too rapidly or too slowly, is too far away to be perceived or because we lack the necessary sensory organ altogether, so...

4. ... one has to be able to create and use instruments like a magnifying glass or a Geiger counter which make those aspects of the world which are beyond human perceptual capabilities perceivable nonetheless.

Even by using instruments, it often remains difficult to make out certain aspects of an object or a process of interest because in normal situations many influences overlap. In order to get differential experience about these properties of the environment,...

5. ... one has to be able to create experimental constructions, which serve the purpose of singling out properties of things and processes.

A famous example of an experimental construction is Galileo's experiment on free fall<sup>84</sup>. How objects accelerate while falling was neither perceivable nor

---

84 More on this experiment can be found in Naylor (1974). In this work, Naylor expresses doubts about whether Galilei's experiment could have produced the results he described or if Galileo rather described how an idealised experiment should be conducted. I disregard such questions here, as they have nothing to do with the general idea of using constructions in physical investigations.

measurable in his time, so Galileo created a construction in which instead of letting something fall down, he let a bronze ball roll down a channel on an inclined plane. In order to measure the time this took, he used a water clock.

“For the measurement of time, we employed a large vessel of water placed in an elevated position; to the bottom of this vessel was soldered a pipe of small diameter giving a thin jet of water, which we collected in a small glass during the time of each descent, whether for the whole length of the channel or for a part of its length; the water thus collected was weighed, after each descent, on a very accurate balance; the differences and ratios of these weights gave us the differences and ratios of the times, and this with such accuracy that although the operation was repeated many, many times, there was no appreciable discrepancy in the results.”<sup>85</sup>

Galileo let the ball roll down different lengths of the channel, measured the time the ball took, repeated this experiment “a full hundred times” and “always found that the spaces traversed were to each other as the squares of the times, and this was true for all inclinations of the plane, i.e. of the channel, along which [he] rolled the ball”<sup>86</sup>. For his experiment, Galileo used two constructions in order to measure something which by itself was not measurable. He used the inclined plane in order to slow down the process and instead of measuring time, he used a construction using which he could measure the weight of water in a vessel, instead.

Notice that the way differential experience is defined, it only describes a difference between assumptions and perceptions. This does not mean that what is perceived actually is what an observer thinks it is. I do not want to go into the philosophical discussion whether reality can be perceived at all but would like to point to the fact that using instruments or experimental constructions does not guarantee that what is perceived actually is what is there or what is happening. It is possible to use an instrument and observe something one thinks is a certain property of an object but which in fact either is another property or the result of a still unknown or at least unconsidered interaction<sup>87</sup>. Using luminol as a chemical

---

<sup>85</sup> Galilei (1638/1914)

<sup>86</sup> Ibid.

<sup>87</sup> Such problems are targeted by the “holy trinity” of quantitative research which consists of objectivity, reliability, and validity. Even though according to Spencer et al. (2003), there are

means of finding blood traces, for example, can result in false positives, as the substance does not only indicate blood but also reacts to other substances such as urine and bleach. The pre-Copernican models for planetary motions are a famous example in which perceptions were wrongfully attributed to properties of the planets. Before the recognition of the sun as the centre of the solar system, the planets' trajectories were explained as epicyclic motions. This model for explaining planetary motions very well corresponded to the perceptions made by observers on earth. It needed the Copernican revolution to understand that the properties of the perception were not actually properties of the planets' motions but properties of the interrelations of the planetary motions and the motion of oneself being on the surface of the now-moving earth<sup>88</sup>.

When differential experience is made using constructions which involve artefacts which are not themselves the object of interest and which do not themselves carry the property of interest, like when using instruments and experimental constructions, I speak about using *means of differential experience* for which I define three conditions.

Means of differential experience must be **expectation-independent**: While someone seeking differential experience may, of course, have an expectation about the outcome, this expectation must not influence the process of gaining the differential experience itself. In particular, it must not influence the outcome of this process. A construction or technique which can only find out *that* an object or a process has a certain property cannot be a means of differential experience as it could not refute but only confirm one's expectations. Means of differential experience always find out *whether* something has a certain property, so there must always be the possibility of being surprised by the outcome.

One could argue that this condition would already suffice in order to describe something as a means of differential experience, as expectation-independence is the fundamental requirement for differential experience as described before. I will nevertheless introduce two more conditions which eliminate constructions of which one can know in advance or where the chances are high that the differential experience one can get from them does not have anything to do with the property

---

a "myriad [of] ways in which these concepts have been conceptualised and redefined" (page 59), their focus always lies on criteria for proper studies (or experiments) with the purpose of ruling out unwanted interfering factors.

88 This paradigm shift has been explained in great detail in Kuhn (1957).

or effect one wants to observe. These conditions, as the example of planet trajectories has shown, one can never be ultimately sure about, as one can never know if there are unknown factors of influence. Because of that, instead of demanding that these conditions *must* be fulfilled, I suggest that they *should* be fulfilled in the sense of that means of differential experience should be thoroughly checked for known problems and that perceptions gained using means of differential experience should always be cross-checked with other means in this respect.

Means of differential experience should be **ghost-free**<sup>89</sup>: What one perceives when trying to determine a property of an object or a process should not be a property of the device or technique used. Many supposed sightings of UFOs on photographs or home videos, for example, are due to reflections within the lens systems of the cameras used, so people assumed the existence of objects where there actually were none. The “ghost” appears through the fact that using a lens system one not only observes the outside reality but also perceives properties of the lenses themselves. In many practical applications of constructions and instruments, it is likely one cannot guarantee that they are completely ghost-free. In those cases, a differential experience should be questioned and cross-checked using other means.

Means of differential experience should be **non-interfering**: The means using which one wants to gain differential experience should not themselves influence the very property they are to observe. This is a common problem in social sciences where the mere fact of someone being the observer changes the behaviour of the people involved. Situations where a measurement changes the measured factor are also known in physics and chemistry. Non-interference in this sense does not necessarily mean non-destructiveness. A test of the durability of material can, of course, destroy it in the process, yet the material should not be (or appear to be) more or less durable because of the way the durability is tested.

---

<sup>89</sup> I chose the word ghost following the notion of sensor ghosts, where sensors appear to pick up something external but actually pick up a part of themselves or register disturbances originating from their own operation.

## 3.2 Using Media for Differential Experience

Media, in the sense of manipulating the surrounding world in a way that has a meaning to people, provides further possibilities of differential experience which go beyond just comparing objects. According to Kirsh (1995), space and the arrangements of objects in space can be used to “simplify our cognitive and physical tasks [...] because of the way we are embedded in the world.” Among many other everyday tasks, he observed to what extent people structure their environment while cooking, which included the purposeful orchestration of ingredients within the peoples’ visual fields as well as using cutlery as flexible progress indicators. In these cases, objects of the environment, the ingredients and kitchen tools, fulfil the role of media objects used as a reminder or for visualizing a status.

Media in the usual sense of the word, especially persistent media, very much relies on symbolic inscriptions into a material carrier. Leroi-Gourhan (1964) links the evolution of the human race in the last millennia directly to the development of its means of expression<sup>90</sup>, namely to “its ability to express thought in material symbols”<sup>91</sup>. The use of symbols inscribed into a carrier came up with the need for bookkeeping in Mesopotamia in the 15<sup>th</sup> century BC<sup>92</sup>. Ifrah (1981/1985) explains the ideas behind early symbolic inscriptions by referring to an imaginary shepherd with an everyday problem:

“Let us imagine a shepherd, unable to count, who has a flock of sheep that he keeps in a cave at night. There are fifty-five of them, but he has no understanding of what “the number 55” means. He knows only that he has “many” sheep. But he would like to be sure that all of them come back every evening. One day he has an idea. He sits down at the entrance of his cave and has the sheep go into it one by one. Each time one of them passes in front of him, he makes a notch in a bone. When all the sheep have passed, he has made exactly fifty-five notches, without knowing the arithmetical meaning of that number. From now on, he has his sheep go into the cave one by one every evening. Each time one of them passes he puts his finger on a notch, starting at one

---

<sup>90</sup> Leroi-Gourhan (1964), page 210

<sup>91</sup> Ibid., page 187

<sup>92</sup> See Ifrah (1981/1985) pages 151ff



end of the bone, and if his finger reaches the last notch he is reassured because he knows that his whole flock has safely returned.”<sup>93</sup>

In this imaginary scenario, the shepherd uses a medium. This medium, the bone with notches in it, is a means of differential experience according to the criteria I specified above: The outcome of his “counting” is expectation-independent because it is independent of whether or not the shepherd thinks all sheep are there. It also is ghost-free as the result is only depending on the presence of the sheep, not on the counting process itself and finally, the fact that the sheep are counted does not influence whether or not they are all present, so the counting process is non-interfering in respect of its result.

Making notches in a bone is a very good example of the advantages and disadvantages of inscription media as described in chapter 2.4.3. In inscription media, a carrier is manipulated mechanically, magnetically, or chemically. This way of manipulating a carrier creates a medium, which is both persistent and, at least in most cases, mobile<sup>94</sup>, so it allows its readers to have differential experience independent of time and space<sup>95</sup>: The shepherd in Ifrah’s example becomes independent of time, as the bone allows him to compare the size of his flock with the size of the flock in the past. The bone-medium also is independent of space because it can be carried around and thus allows the sheep to be counted not only in the cave the shepherd originally created the notches but in every location where the sheep can be isolated from each other. The imaginary shepherd uses a bone as a medium in order to count the sheep without having to remember their number and without even having to have counting skills. Persistent notes, such as the bone in this case, not only help to keep track of things or to remember facts but also allow for mediated differential experience, like when someone is looking something up in an encyclopedia.

The technique of writing things down using symbols is the basis for formal operations, as described in chapter 2.3.1. Such operations can by themselves be a means of differential experience as they can meet the given criteria. Take, as an

---

93 Ibid., page 9

94 There are forms of media, like blackboards in a school, which are fixed to a location and cannot be moved.

95 This is often attributed to digital media but actually is a property of inscription media in general. The advantage of a book is that it can be read anywhere and anytime by anyone, thus independent of space and time.

example, a calculation which determines whether one is broke by comparing a list of one's earnings with a list of one's spendings.

- **Expectation-independence:** What the calculation “looks at” is a set of symbols, i.e. a list of spendings and earnings. It only works with these symbols thereby creating another set of symbols. In doing that, the calculation as a formal operation does not depend on what the person who happens to perform the operation thinks about the outcome as, assuming the calculation is performed correctly, this outcome is determined by the form and the arrangement of the initial symbols only. This is true for all calculations.
- **Non-interference:** Every formal operation on symbols is non-interfering. A real-world problem has to be translated into symbols on the basis of which the operation is conducted. Such an operation could, if at all, only manipulate these symbols but cannot directly influence real-world affairs. If formalisms are understood in the way that the initial arrangement of symbols remains untouched (which is the case in written calculations), it would be non-interfering even on this level. Applied to the example, non-interference means that by calculating whether or not one is broke one neither becomes broke nor rich.
- **Ghost-free:** A ghost in the being-broke calculation would mean that the determination of whether one is broke or not depends on a property of the calculation process<sup>96</sup>. For that, the formal operation would have to somehow be dependent on properties of itself. While it might be difficult to generally rule that out for all calculations and formalisms, in the simple case of the being-broke calculation there are not any ghosts to be found.

In accordance with Leroi-Gourhan (1964), who linked the evolution of the human race to its means of expression and thus to technology, Perkins<sup>97</sup> suggests to look

---

<sup>96</sup> The result of the calculation, of course, depends on the definition of being broke. If this definition were different the results of the calculation would consequently be different. In this sense, the result of a calculation depends on properties of itself, yet this is not what is meant here. A calculation would not be ghost-free, if due to properties of the formalism there was the possibility of getting a result which is not consistent with what the formalism is meant to produce as a result.

<sup>97</sup> See Perkins (1993) and Salomon&Perkins (2011)

at what he calls a “person-plus” instead of assessing capabilities of a person deprived of his technical means:

“We could take as our unit of analysis not the student without resources in his or her surround – the person-solo – but the person plus surround, or person-plus for short, in this case the student plus the notebook. We could say that this person-plus system has learned something, and part of what the system has learned resides in the notebook, rather than in the mind of the student.”<sup>98</sup>

These thoughts put media, and with it the possibility of arranging symbols and objects within the action and perception space of a person, into the centre of knowledge work.

### 3.3 Object Arrangements

The use of media, both in the simple sense of arranging things in one’s environment in order to be reminded of something as well as in the sense of handling documents and other media artefacts, depends on the possibility of being able to arrange such objects within a person’s action and perception space, thereby forming a personal working environment.

Arranging artefacts in a spatial environment such as on the surface of a desk allows people to correlate them meaningfully. This not only means being able to have them in sight together but also allows for an explicit expression of object relations by spatially ordering them, grouping them or by putting them in relation to a structured background. To create such arrangements, a relatively small number of basic manipulation operations can be identified. As all of these operations change the perceivable arrangement of objects in the action and perception space, I call them basic *spatial arrangement operations*<sup>99</sup>.

---

98 Perkins (1993), page 89

99 My basic spatial arrangement operations closely resemble what has been described as “personal primary media functions” in Hampel (2001) and Hampel&Keil-Slawik (2001). Hampel and Keil-Slawik wanted to specify operations a computer system needs to provide in order to support basic functions of knowledge work, so their choice of words (create, delete, arrange, link) reflects the operations such a system must provide. Despite differences in the choice of words, there is one more profound difference. “Recomposing” in my case means less than what Hampel meant with linking. Hampel’s definition mixes the level of making an artefact arrangement permanent—thereby creating another artefact—with specifying semantic object relations which are not genuinely spatial in nature.

**Insert:** The most basic necessity of creating a working environment is bringing an artefact which was out of sight before into one's perception space. This could, for example, mean taking a document out of a filing cabinet and putting it onto a desk.

**Create:** While artefacts, on the level of the carriers, are typically not created within a personal working environment<sup>100</sup>, on the level of individual symbols, media artefacts can be created, e.g. by writing down letters onto a piece of paper.

**Rearrange:** Assuming all necessary artefacts are provided, working with them to a great part means rearranging, thus moving, them. Arranging them allows for many semantic processes like comparing or relating, but also allows for a meaningful interpretation of the position itself, which will be the main focus in the remainder of the thesis.

**Recompose:** Being able to rearrange artefacts often is a question of object granularity. Recomposing in this sense means changing the granularity of an object by taking it apart or combining several objects in such a way that they appear to be one single object. This allows, for example, for the creation of a coherent body of text by arranging paragraphs and sections and then making the whole arrangement an object of its own.

**Remove:** When media objects are not needed anymore, it has to be possible to remove them from the action and perception space. Removing an artefact could mean throwing it away altogether or just filing it somewhere so that it is not part of the active working environment anymore.

In the following characterisation of knowledge work scenarios, I describe both analogue and digital techniques. They, of course, differ to a great extent in terms of media discontinuities<sup>101</sup>. Such discontinuities necessarily occur when one of the arrangement operations cannot be carried out, and one has to circumvent the resulting problem, e.g. when the granularity of a document can only be changed by making photocopies of individual pages of a book or by cutting a text into pieces. At this point though, I do not put the focus on the question of to what extent digital potentials can avoid those discontinuities but focus on the characterisation of the way space itself is used in the scenario.

---

<sup>100</sup>Sheets of paper are bought, not "created" in the office.

<sup>101</sup>See chapter 2.4.3.

### **3.4 Space as a Means of Differential Experience**

Space plays an important role in knowledge work. Even for the very simple task of comparing two objects a common space is needed in order to bring the objects together. While, in this case, space is only needed as a medium, which means the positions of the objects being compared have no other purpose than allowing their comparison, spatial aspects like positions or spread can also be used as means of differential experience of their own. In the following, I look into areas in which space is not only used as a medium but where spatial properties of the objects, especially their positions, are used in order to make something perceivable and manipulable which otherwise would not be accessible.

#### **3.4.1 The Use of Space on a Desk**

Desks provide space on which knowledge work can take place. The way desks are structured when performing knowledge work differs greatly from person to person and from task to task. In studies carried out by Malone (1983) and Kidd (1994) in which they interviewed a number of desktop workers and examined their offices, they found knowledge workers whose desktops, from an outside perspective, seemed to be chaotic and unstructured while others had desks which were well structured and tidied up. Based on personal working style and on typical tasks, Malone (1983) distinguishes between filers and pilers. Filers have a clean, “neat” desktop and organise their working materials in a separate filing system. Pilers, on the other hand, tend to have more chaotic or even cluttered desktops. According to Malone<sup>102</sup>, filing and piling have two quite distinct functions. Filing is used for classifying artefacts which have to be revisited at a later time. In jobs, where there are many data artefacts which have to be referred to at a later time and of which one rarely uses more than a few at the same time, filing often is the method of choice. On the other hand, Kidd (1994) characterises those knowledge workers who are pilers rather than filers as relying heavily on “using their desks and floors as a spatial holding pattern for paper-based inputs and ideas”<sup>103</sup>. Pilers use physical space in order to have those items they have not yet ordered or categorised within reach, so Kidd, therefore, concludes that “filing is uncomfortable for them because they cannot reliably say when they want to use a particular piece of information or to which of their future outputs it will relate”.

---

<sup>102</sup> See Malone (1983), pages 105, 106

<sup>103</sup> Kidd (1994) pages 187-189

Their style of knowledge work strongly relies on the ability to dynamically structure the objects in the environment in order to explore them and find connections between them.

The biggest advantages of desks in the context of being a knowledge work environment are its size and its flexibility regarding arrangements and markup. A desk allows for the arrangement of many objects which nonetheless remain perceivable and distinguishable. During ongoing knowledge work, structures can emerge on a desktop, yet these structures often remain vaguely defined and might not even be obvious to the knowledge workers themselves. Examples of possible emerging structures are regions which reflect different degrees of urgency or regions connected with a certain task when, for example, everything related to the doctoral thesis is put to the left side of the desk while all documents used for the preparation of a lecture are put to the right-hand side. A big advantage of a desktop, despite its mere size, is the multitude of possibilities for arrangements and mark-up. Documents can be put onto a tidy pile or onto a messy stack, one or several objects might be twisted, dog-eared, or a certain object can even be diagonally put on top of several piles so it inevitably, and quite literally, stands out. As there is no strict formality, these structures and markups are not enforced by any means and can change at any time.

There is no digital equivalent for knowledge work on a desk in today's digital media. While there is a user interface concept called "desktop", it is a mere metaphor, which, depending on its specific implementation, provides a number of functions of a real-world desktop, but in no way has its spatial qualities. In the classical office world, the desktop is the place where knowledge workers bring together, organise and structure all the artefacts they need for their current work. In this place, the objects reveal themselves and can be identified in a glimpse, often even when they are in a pile. While an office desk is big, computer screens have been, and still are, relatively small in comparison. On a typical laptop screen, maybe one to two pages of classic text can be displayed. On a big office computer screen it might be four or six, not to even mention tablet computers or smart phones. The consequences of this lack of space on the way people work are quite well described in Henderson&Card (1986). In a "gedanken experiment", they explained how knowledge work would change if one could not use the dining

room table or a desktop to organise material used for a paper but would be forced to use a very small table instead:

“Much of the writer’s time must be devoted to thrashing about searching through papers. [...] The ensuing chaos will tend to alter the task itself, pushing the writer toward more formal methods of accessing information, such as file folders and note cards, which have their own overhead, memory for categories, and conceptual ambiguities.”<sup>104</sup>

In the classical office, there was a clear distinction between spaces for current work, mainly the desks, and facilities used for filing, such as filing cabinets or card boxes. When an office worker wanted to work with a file, he took it out of the filing cabinet and put it onto his desk where he had the opportunity to arrange it with other pieces of information which could have been other files but which also could have been objects which were not subject to filing, not suited for filing, or not yet in a state which the office worker held appropriate for filing. In the classical office, the liberties provided by a desk allowed knowledge workers to keep unfiled or unfileable knowledge artefacts available until they either are in a state in which they can be filed or until they are not needed anymore and are discarded. According to what Kidd described in 1994, and what from personal experience I still consider to be true today, typical commercial computer systems do not provide these liberties but mainly are filing systems. Even though knowledge workers are often faced with material they cannot immediately name and file, they are “forced to classify the inherently unclassifiable” by giving it a name and assigning it to a place within the file system.

On the Xerox Star, which was the first system to implement the desktop metaphor<sup>105</sup>, there was a strong emphasis on the objects on the desktop which were clearly separated from filing functionalities. The objects on the desktop represented the users’ “working environment—where [their] current projects and accessible resources reside”<sup>106</sup>. Figure 11 shows a screenshot of the desktop of the Xerox Star having a number of objects placed on it. While most of these objects are presented as icons only, one of those objects, a text document, is presented in

---

104 Henderson&Card (1986), page 217

105 Xerox PARC used the term “office metaphor” for their concept of which the desktop was but one element.

106 Smith et al. (1982), page 255

### 3 Differential Experience and Meaningful Use of Space

an opened state as a window. Notice that the document is presented *as* a window, not *in* a window. Both the icon as well as the window are representations of the same object.

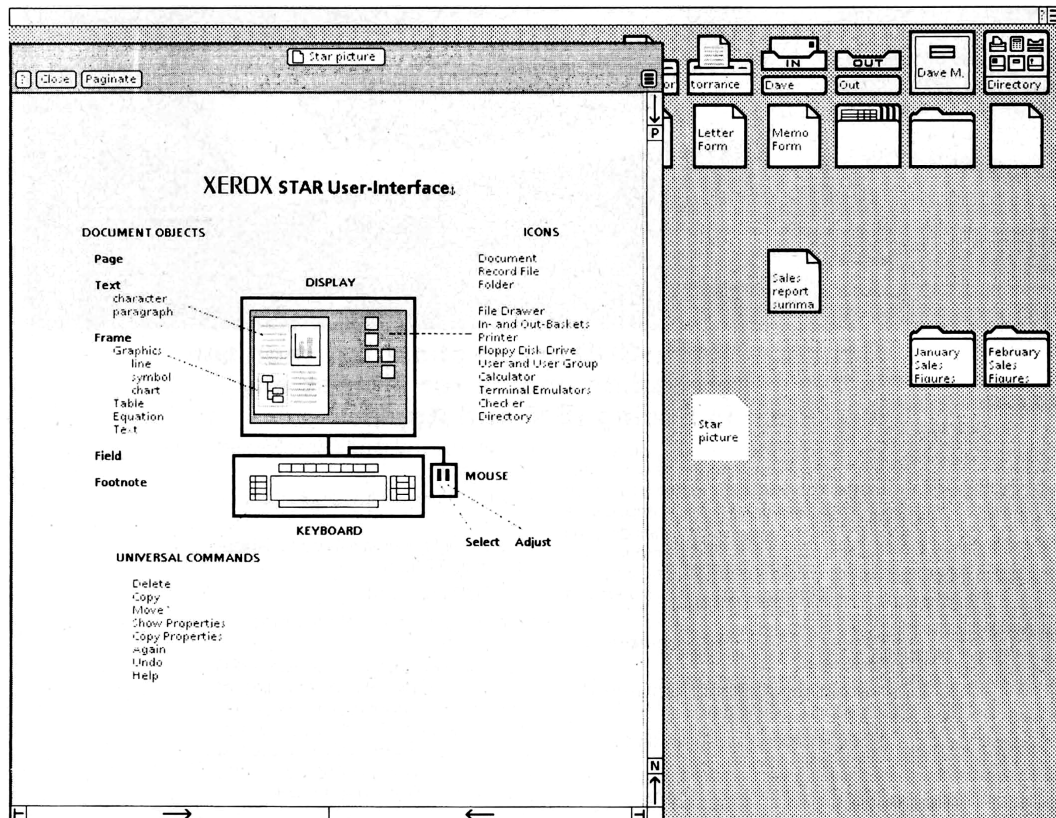


Figure 11: The desktop of the Xerox Star (screenshot taken from Smith et al. 1982)

As explained in chapter 2.2, on the Xerox Star there was no concept of software, so, in contrast to today's computers, a window did not represent an application with a document opened within it, but the window was supposed to be the document itself in its opened state as a window. Having two different modes of presentation was necessary because of the relatively small size of the Star's screen in relation to a real-world desktop. The desktop in figure 11 holds the objects important for current work as well as templates for new documents<sup>107</sup>, inboxes and outboxes used to send those objects to other people, as well as filing cabinets (the object labelled "Dave M." in the figure), which represent fileservers. When objects were put into a filing cabinet, they were removed from the desktop and

<sup>107</sup> On the Xerox Star, new documents were created by copying existing ones instead of by invoking a "Save As" command. I explained this operating style in chapter 2.2.



from that point on were accessible through their entry in a list view which could be sorted by certain criteria such as name or date. In order to work with them again, they had to be put back onto the desktop where they again appeared as spatial objects which could be opened into a window.

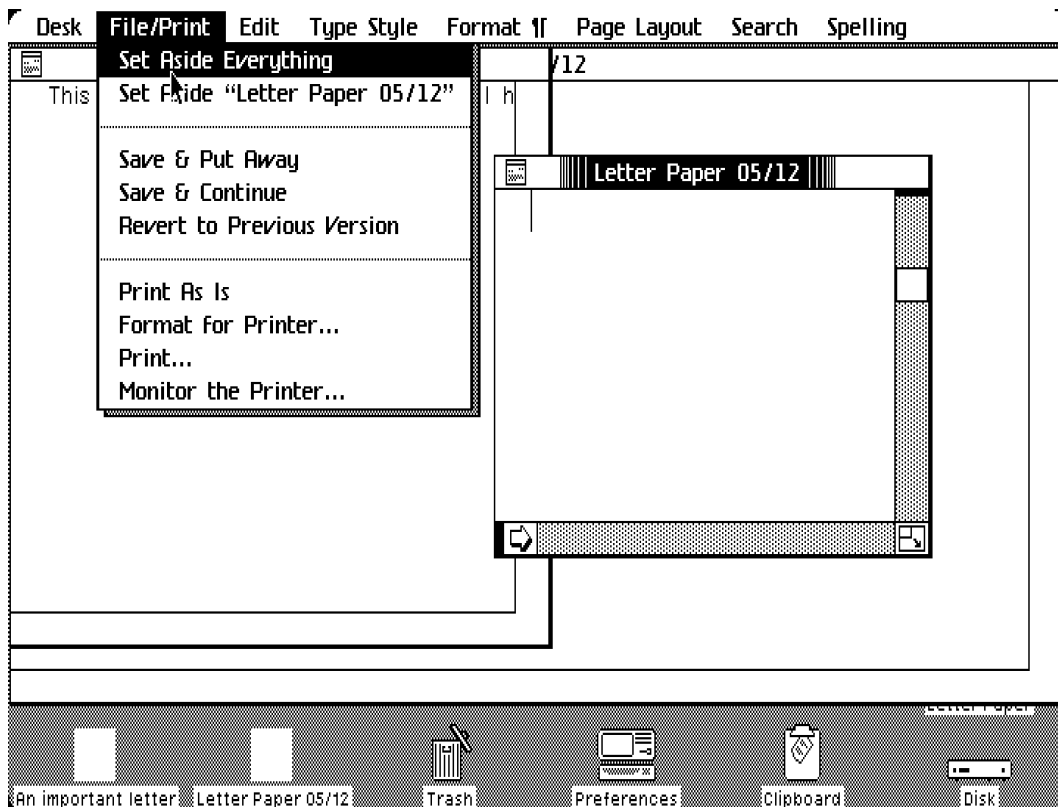


Figure 12: Document windows of the Apple Lisa and the "Set Aside" function about to be invoked.

The desktop of the Apple Lisa (see figures 2 and 12) shared the same basic concept of a desktop being the location for the current working objects. Like on the Star, documents, folders and other objects were visualised as icons or as windows. While on the Star, the decision about objects being on the desktop or being filed was left to the user, the operating system designers at Apple implemented a complex mechanism which determined whether or not an object appeared on the desktop. In contrast to the Xerox Star, on the Apple Lisa every document had a location on a floppy disk or hard disk, so in this sense, every object was filed, but at the same, time every filed object could temporarily be on the desktop. There were a number of ways how filed objects could end up on the

desktop. One could, for example, simply have dragged an object from a folder window onto the desktop. In contrast to modern operating systems, as well as in contrast to the Xerox Star, this would neither have moved nor copied the file to the desktop. The file rather retains its original position within the folder, indicated by a so-called shadow<sup>108</sup> at its filing location (see figure 2), but would additionally be available on the desktop until it is explicitly sent back. Figure 12 shows another possibility of how an object could end up on the desktop. A document is opened at its filing location. As assumed here, it is still needed for the current work so the user decides to “set it aside”. Setting a document aside “reduces an open window to an icon, and then puts the icon on the desktop”.<sup>109</sup> Documents residing on the desktop could be removed from there either by dragging them into a filing location, putting them into the trash bin or “putting them away”, which means sending them back to the locations where their shadows reside, by calling a respective command.

While the desktop concepts of Xerox and Apple indeed show a number of conceptual similarities with classic desktops in some regards, working with objects on a digital desktop can hardly be compared to working on a real-world desktop. In an early article about the desktop metaphor, Smith et al. (1982) explain:

“You can “open” an icon to deal with what it represents. This enables you to read documents, inspect the contents of folders and file drawers, see what mail you have received, etc. When opened, an icon expands into a larger form called a “window,” which displays the icon’s contents. Windows are the principal mechanism for displaying and manipulating information.

The Desktop “surface” is displayed as a distinctive gray pattern. This restful design makes the icons and windows on it stand out crisply, minimizing eyestrain. The surface is organized as an array of one-inch squares, 14 wide by 11 high. An icon can be placed in any square, giving a maximum of 154 icons. Star centers an icon in its square, making it easy to line up icons neatly. The Desktop always occupies the entire display screen; even when windows appear on the screen,

---

108 See Apple (1983), page B-19

109 Ibid., page B-22

the desktop continues to exist beneath them. The Desktop is the principal Star technique for realizing the physical office metaphor. The icons on it are visible, concrete embodiments of the corresponding physical objects. Star users are encouraged to think of the objects on the Desktop in physical terms. Therefore, you can move the icons around to arrange your Desktop as you wish. (Messy Desktops are certainly possible, just as in real life.)”<sup>110</sup>

While Smith et al. (1982) try to emphasise the similarities between their desktop and a real-world desktop, their explanation reveals why a digital desktop cannot substitute a real desktop in terms of its spatial properties.

- On a real desktop objects may be occluded, but if they are not, they reveal themselves as they are. There is no minimised version of anything on a real-world desktop; thus nothing has to be magnified into some sort of window in order to use it.
- 154 objects on a desktop are not too many, especially considering this number only describes the number of minimised objects. The number is especially low considering that when trying to structure the objects on the desktop, which implies that there must be spatial distances between agglomerations of icons, not every position may be filled with an icon. The situation becomes even worse when considering full-sized, “opened” documents where even on a modern computer no more than four to six of them can be reasonably arranged on the screen.
- Making it easy to “line up icons neatly” may be desirable in some cases, but automatically lining them up neatly reduces the possibilities of structuring them according to one’s needs and thus prevent many structures from being created.
- In the real world, a document on a desktop hardly ever occludes the whole desktop in a way it would be necessary to assure that the desktop “continues to exist beneath” them. On a digital desktop, such an occlusion happens frequently. As a consequence of that, in contrast to a real desk, arrangements on the desktop are not continuously visible and thereby can only hardly be subject to the knowledge work process.

---

<sup>110</sup> Smith et al. (1982), page 256

Please note that by pointing out these discrepancies, I do not want to state that digital desktops are not useful in knowledge work, that they did not simplify the user interface significantly, or that digital desktops necessarily should provide additional means of arranging objects in stacks or piles<sup>111</sup>. My remarks only serve the purpose of making clear that, regarding space, digital desktops are behind real-world desks to a degree that the main advantages of desks, to allow for a flexible creation of arrangements, is almost completely lacking.

An area in which the desktop concept of Star and Lisa indeed borrow a central concept of desktop work is the provision of a location where knowledge workers can bring together the objects they need for their current work. On the Star, documents had to be brought onto the desktop before they could even be edited and on the Lisa, documents which were “set aside” were automatically put onto the desktops. This hub characteristic of the desktop has since, in fact quite rapidly, disappeared. Tognazzini (1995)<sup>112</sup> describes a technical reason for its disappearance. The Xerox Star, as well as the Apple Lisa, both were very powerful and thus expensive machines. They both had one MiB of RAM and a hard disk at their disposal. This allowed documents to be persistently and rapidly available and it allowed several applications to be running at the same time. With the introduction of the Macintosh in 1984, which was targeted at the consumer market, those conditions changed. The first Macintosh only had 128 kiB of RAM and instead of a hard disk only provided an internal diskette drive. This had consequences for the operating system. On the Lisa, if one wanted to edit a text file, for example, one could locate it on the desktop or its hard disk filing location. As the application was also saved on the hard disk, opening a file could directly trigger the respective application to be opened. This enabled a document-centred approach. On the Macintosh though, there was no persistent hard disk and only one program could run at a time. The user booted the system using an operating system disk. He then had to eject this disk and had to insert the floppy disk containing the word processor which he could then start. Due to the small disk sizes it is likely that the text was not saved on the same volume, so in order to

---

111 While a number of researchers over the years suggested virtual piles on the desktop, none of their suggestions made it into current operating systems. See, for example, Malone (1983), Mander et al. (1992), Robertson et al. (1998) and Agarawala (2006). Lansdale (1988) criticised the idea of implementing stacks in digital desktop systems by stating that solutions in one technology do not necessarily have to be good solutions in another technology.

112 Tognazzini (1995), page 129

open the file, the user had to open a dialogue in which he could eject the program disk and, after inserting the floppy disk containing the text file, select and open it. Said technical restrictions have since disappeared, but many consequences of this shortage of resources, explicit saving commands for example<sup>113</sup>, have survived till the present day.

#### **3.4.2 Spatial Arrangements in “Thought Structuring”**

In contrast to desktop work, which in large parts is informal, even the most chaotic knowledge workers often purposefully create spatial structures in order to create persistent traces of their thought processes. These traces remain within their perception spaces and thereby constantly reflect back to them during ongoing knowledge work. Techniques in this area are often called “thought structuring”. This term is somewhat imprecise or even misleading as it suggests one would actually structure one’s thoughts, get them out of one’s brain onto a piece of paper or put them back into the brain. From the artefact-based perspective on knowledge structuring put forward in this thesis, the argumentation can less be about directly structuring one’s thoughts but must focus on the artefacts which are created while performing knowledge work. One likely cannot directly control what someone thinks or learns but one can influence the environment in which such processes take place. In terms of this thesis, this means that one can control the circumstances of differential experience and provide opportunities for making such experiences. Making differential experiences during the process of arranging and relating traces of one’s own thoughts is active knowledge work. By building structures which stay persistent in the visual field of the knowledge worker, they can channel their thoughts and thereby find out more about their own understanding of things. Many spatial arrangement techniques to structure one’s thoughts are based on graph structures. Graphs, allowing for visual connections, have advantages in relation to linear notes when it comes to finding or depicting relations between things. Well known examples of graph-based thought structuring techniques are mind mapping and concept mapping.

---

113 Explicit saving as well as the “save as” command were necessary due to slow writing speeds of the floppy disk drives, noisy drives, and due to the fact that the file edited in an application likely was saved on a floppy disk which is not inside the disk drive at the time of the editing.

## Mind Mapping

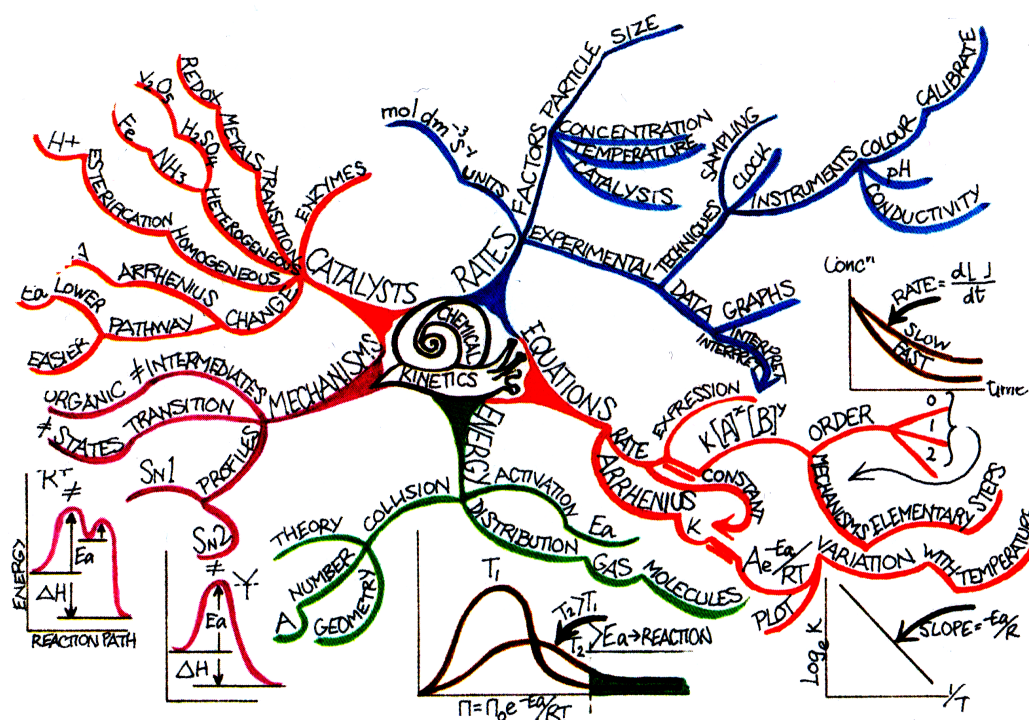


Figure 13: Mind Map on Chemical Kinetics by Graham Wheeler covering an entire section of a chemistry course (Mind Map and description taken from Buzan&Buzan (2003), plate XVI, page 185)

Figure 13 shows a typical mind map. From a graph point of view, mind maps are tree structures. The root of this tree is positioned in the middle of the mind map. It typically consists of a central word or an iconic image representing the general topic of the map. The map in the figure divides into five branches, which spread out from the centre image. They represent the map's main categories. Each of these categories then branches into further subcategories and elements. Mind maps, which are used in business processes and educational contexts, have been made popular by Tony Buzan who, in bold words, calls them “the next evolution in human thought” and explains

“The Mind Map is an expression of Radiant Thinking and is therefore a natural function of the human mind. It is a powerful graphic

technique which provides a universal key to unlocking the potential of the brain.”<sup>114</sup>

This suggested similarity between structures within the human brain and the structure of a mind map on paper is explicitly described in Buzan (2002):

“A Mind Map is the easiest way to put information *into* your brain and to take information *out* of your brain—it’s a creative and effective means of note-taking that literally “maps out” your thoughts.”<sup>115</sup>

Buzan&Buzan (2003) highlight the associative nature of mind mapping. A whole chapter of their book<sup>116</sup> is dedicated to brainstorming. Despite this emphasis on associations, they declare hierarchies, categorizations, and order<sup>117</sup> to be the key factors for good mind maps. While this is quite a contradiction to the associativity they claimed before, building hierarchies, categorizing things and bringing them into an order are active acts of knowledge work, the purposeful struggle with mere chaotic associations. In a mind map, a new “thought” cannot be just put somewhere as documents could on a desktop but has to be related to what is already there.

The idea of mind mapping, described neutrally without questionable comparisons to brain structures or bold claims of revolutionary techniques, can be summed up as follows:<sup>118</sup>

- Prepare a big sheet of paper and some felt pens.
- Write a starting term or draw a starting picture into the centre of the sheet.
- Think about the main categories associated with the central term.
- Create branches for each category<sup>119</sup> and label the branches with one or two words.

---

114 Buzan&Buzan (2003), page 55

115 Buzan (2002), page 6

116 Buzan&Buzan (2003), pages 58-75

117 Ibid., pages 76-81

118 based on Budd (2004) and Buzan&Buzan (2003), page 55

119 Buzan finds labelled curving branches to be more natural than labelled nodes connected with straight lines. From a visual perspective, they appear to be trees with unlabelled nodes, yet from a graph-theoretical point of view they more resemble a tree with labelled nodes but unlabelled and untyped, yet directed edges.

### 3 Differential Experience and Meaningful Use of Space

---

- Continue this process until the elements become elementary or you have reached your desired level of detail<sup>120</sup>.
- Add cross-references and illustrating pictures where necessary.

Mind mapping, as a technique which describes a form of writing down thoughts, still is quite informal, so here are only very few rules. The elements in a mind map do not have to be of the same kind throughout the map; thus a map can be a mix of concepts, ideas, examples or categories. The meaning of a connection or of a cross-connection is far from being clearly defined apart from meaning “is somewhat related to”. This under-definedness has the advantage of not impeding the mind-mappers too much while still providing at least some structure. Davies (2011) thus concludes that mind maps are basically “association maps” and elaborates:

“There are no limits on the ideas and links that can be made, and there is no necessity to retain an ideal structure or format. Mind mapping thus promotes creative thinking and encourages ‘brainstorming’”<sup>121</sup>

While Davis, Buzan and others emphasise on the associative aspect of mind mapping, Budd (2004) suggests a distinct brainstorming phase before the actual mind mapping begins.

“Because brainstorming is a critical component of creating a Mind Map, the groups are instructed to brainstorm as their first step. Small post-it notes are quite useful at this stage because the groups can write ideas on the post-it notes and then shuffle them as many times as necessary to create effective categorizations.”<sup>122</sup>

Buzan&Buzan (2003) also suggest a distinct brainstorming phase, but instead of relying on techniques which allow dynamic arrangements and manipulations, as suggested by Budd, they propose an initial “quick-fire mind map burst” followed by a reconstruction and a revision phase<sup>123</sup>. Regardless of the existence of an initial separate brainstorming phase, reconstructions and incremental revisions are

---

120 According to Buzan, the process could theoretically be continued indefinitely.

121 Davies (2011), page 282

122 Budd (2004), page 39

123 Buzan&Buzan (2003), page 135



suggested both by Buzan&Buzan as well as in many practical guidelines, like, for example, in Mattes (2002) who gave guidelines for using mind maps in school:

“The mind map should be worked over in the end. Have the main and subbranches been assigned correctly? Is there another branch which should be added? Are there connections between main and subbranches?”<sup>124</sup>

The reason why incremental revisions (or at least a final revision) are necessary is due to an unavoidable problem which is typical in the distinction between artefact and process. While the mind map as a product is a hierarchical structure, the process of mind mapping is not necessarily, and the aspect of brainstorming surely is not<sup>125</sup>.

#### **Concept Mapping**

Concept maps<sup>126</sup> are more formal and more structured than mind maps. The main structure of a concept map is a rooted tree<sup>127</sup> (see figure 14). In contrast to mind maps, in a concept map all nodes are of the same kind, they all represent concepts, and all edges describe a typed relation between two concepts. Like mind mapping, the creation of concept maps begins with a brainstorming phase which results in a “parking lot” of concepts, which Novak&Cañas (2008) describe as a kind of holding area:

“We refer to the list of concepts as a parking lot, since we will move these concepts into the concept map as we determine where they fit in. Some concepts may remain in the parking lot as the map is completed if the map worker sees no good connection of these with other concepts in the map.”<sup>128</sup>

---

124 Mattes (2002) Page 117: “Am Ende sollte die Mind-Map überarbeitet werden. Sind zum Beispiel Haupt- und Nebenstränge richtig zugeordnet? Sollte noch ein weiterer Strang hinzugefügt werden? Gibt es Verbindungslinien zwischen Haupt- und Nebensträngen?”

125 More on the difference between artefacts and their genesis in chapter 3.5.

126 e.g. described in Davies (2011), pages 282-286, and Novak&Cañas (2008)

127 The structure of a mind map also is a tree which, due to the unusual way of painting a mind map, is less obvious than in the case of concept maps.

128 Novak&Cañas (2008), page 12

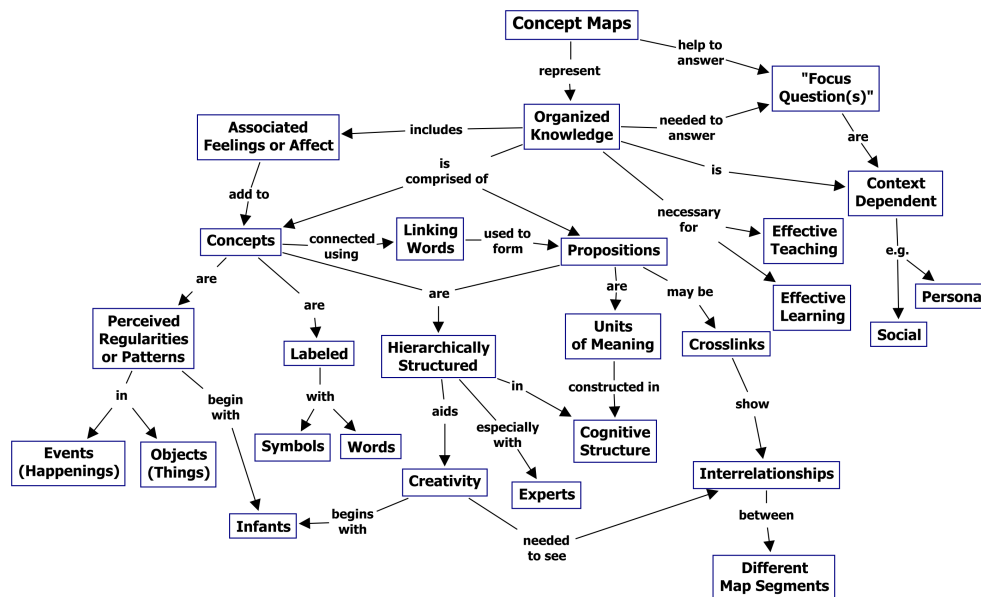


Figure 14: A concept map about Concept Maps (taken from Novak&Cañas 2008)

During the process of creating the map, many revisions are necessary. Arrangements have to be tested and overthrown constantly. Because this process is connected with the necessity of constantly repainting the map, Novak&Cañas (2008) suggest the use of post-its for non-digital concept maps:

“Post-its allow a group to work on a whiteboard or butcher paper and to move new concepts around easily. This is necessary as one begins to struggle with the process of building a good hierarchical organization. [...]”<sup>129</sup>

Finding the initial concepts, of course, is not the only area where effort has to be spent in order to structure the various concepts. According to Novak&Cañas, the most problematic field is the aspect of connecting the concepts and subconcepts with meaningful named relations:

“Students often first do not know how to name the connections and later, when they recognize that every concept can be connected with every other one, they struggle to select the important connections.”<sup>130</sup>

<sup>129</sup> Novak&Cañas (2008)

<sup>130</sup> Ibid.

Quite like in the case of mind mapping, the process of concept mapping is described as an incremental one, which means that in order to improve the structure of an existing map, it has to be repainted repeatedly.

#### **The Metaplan Technique**

Purvis (1983) describes a collaborative thought-structuring approach called metaplan technique or card technique. The goal of this technique is the support of collaborative decision-making in a group process. An initial problem statement or question is put onto a big central pinboard; participants of the decision-making process, in Purvis' case the pupils in a class or course, are given a bunch of cards they write their arguments or answers on. The cards are then collected and clustered in a common group process, meaning cards belonging to a common aspect are spatially grouped by moving them closely together. During this process, a number of problems can occur. Two cards might be almost similar, the content on the cards might be ambiguous, or the clustering itself might be controversial. Such problems have to be solved in a group discussion process. After the clustering is finished, the individual clusters are named and labelled. During this step, Purvis argues, often cards are found which still are grouped in the wrong cluster, so corrections to the spatial clustering have to be made dynamically.

During the metaplan process, the pinboard the cards are put on becomes spatially structured. When the initial clustering takes place, space is still completely unstructured. All positions are equal, so initially, every card can be placed at every position of the pinboard. Later, during the clustering, the positions of cards are determined by comparing their contents with the contents of other cards and, depending on the results of these comparisons, by positioning them in close relation or in a notable distance to each other. The need for these comparisons decreases during the clustering process. At least by the time the clusters get their labels, thereby creating a named spatial region, all further operations can be conducted in relation to these regions rather than to the individual cards. Figure 15 shows the card technique in a state where there are distinct spatial regions to which card clusters are assigned<sup>131</sup>. When new cards are added or moved in this late stage of the process, they are positioned according to these regions without any need for card-to-card comparisons.

---

<sup>131</sup> The structure in figure 15 slightly differs from the rules described here, as it lacks an explicitly written down thesis or question.

### 3 Differential Experience and Meaningful Use of Space



Figure 15: An example of card clustering and labelling using the metaplan technique. (Map by SPD Schleswig-Holstein, published on flickr)

The idea of card clustering like in the metaplan technique has frequently been adapted in digital knowledge work tools. The tools take the advantage of the flexibility of post-it notes or cards as chunks of information which can easily be arranged and combined. Looi et al. (2008) and DiGiano et al. (2006), for example, describe a collaborative card-based solution called group scribbles. Users have at least two separate planes on which they can arrange virtual post-its. In the personal plane, which is private to each individual user, users can create their post-its and maybe pre-structure them according to their own understanding. Common structures can be created by dragging virtual post-its from the personal plane into the public plane. The content of this plane is synchronised between all

participants which effectively makes it a common area in which the post-its can be arranged. Prante et al. (2004) implemented a card metaphor based solution for a big touchscreen wall, which is mainly interesting because of its user interface solution for clustering which is based on a metaphor of magnetic elements which attract or repel each other. Cards, which they call MagNets, which are not part of a cluster repel each other. This prevents them from overlapping and thus guarantees the visibility of every card because cards which are accidentally placed on top of each other automatically move away from each other until they do not overlap anymore. In order to create a cluster, a distinct title card is created. A title card is “attracting” for every card which is put on top of it. Any card which is put on top of such a title card becomes attracting as well, so as a consequence when a title card is present, cards can be clustered by dragging them on top of each other.

#### **Semantic Spaces**

While the digital thought structuring tools mentioned above are digitalised versions of analogue techniques, Klemme et al. (1998) developed a digital knowledge structuring solution which has no direct analogue counterpart and which in its potentials and use cases goes beyond the concepts described above. They describe a scenario, in which a big body of knowledge is available in a hypertext system. Hypertexts by definition build a network of linked text documents. Such hypertext networks can, of course, be visualised as a graph which can be created by systematically following the links between hypertext nodes, thereby creating a map. In some cases, these link structures do not reflect the semantic relations of the text but rather are created according to mere organizational properties, yet even when a hypertext does contain links according to semantic relations, these links have been put into it by the authors and thus can, of course, only represent *their* semantic connections and thus can hardly reflect a *learner's* understanding of a specific matter.

Klemme et al. proposed *Semantic Spaces* as “a tool for the active exploration of hypermedia systems”. A semantic space like the one in figure 16 allows documents and anchors from a hypertext system to be directly put onto a two-dimensional surface on which they can subsequently be arranged to one's liking, thereby creating a personal map of one's understanding of the contents. Links into the hypertext which are created this way are bi-directional, which means whenever a user navigates through the hypertext and reaches a node for which a

### 3 Differential Experience and Meaningful Use of Space

link exists in a semantic space, this link is highlighted, making semantic maps personalised navigational maps for the hypertext system.

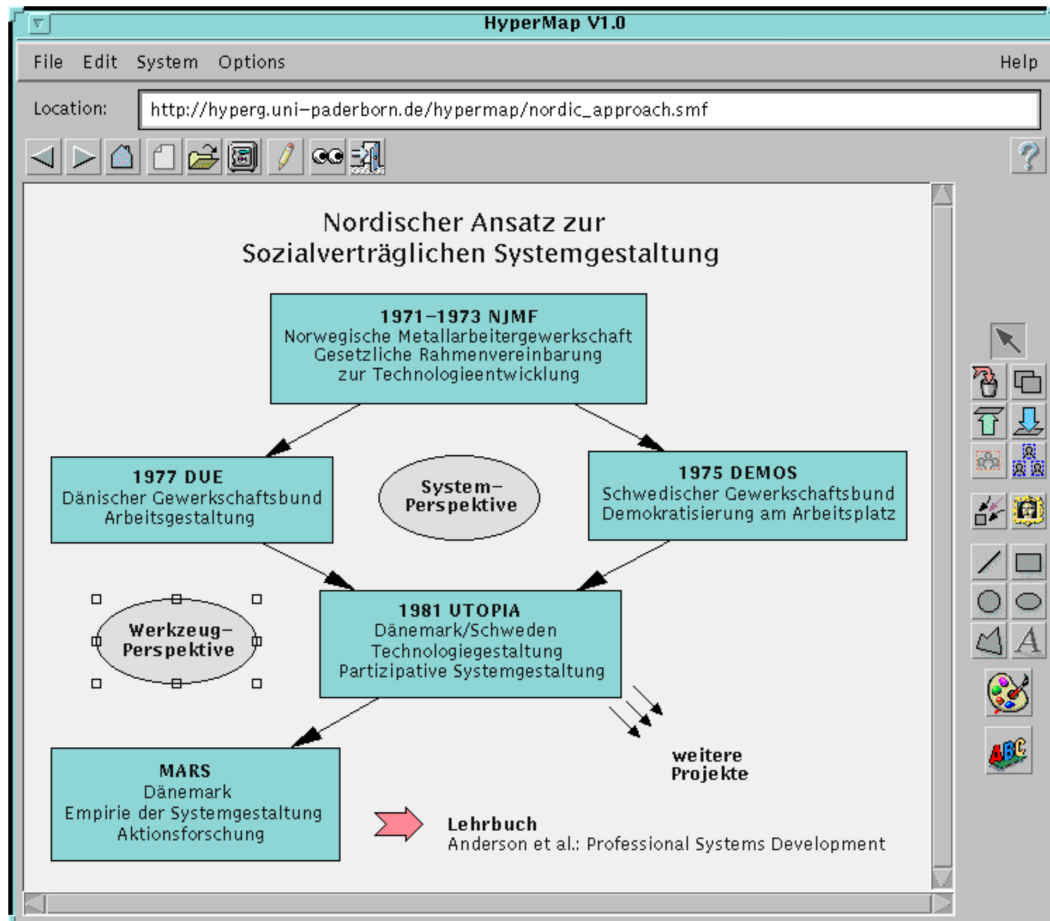


Figure 16: A semantic space visualizing a personal arrangement of interlinked documents provided by the lecturers of a computer science course. (Screenshot taken from Klemme et al. 1998)

Links can be arranged freely on the surface, so, as Klemme et al. put it, “information can be sorted according to metrics that the users feel to be meaningful for their purpose”. In order to indicate those metrics, additional elements such as texts, graphical elements or even images can be added to the semantic space. All these elements, including the links, retain their character as digital objects, which means they can be rearranged and manipulated in order to reflect changes in the understanding of the subject the space is about.

### 3.4.3 Structured Space in Knowledge Work

Summing up the findings of the investigation of the use of space in knowledge work contexts so far, it can be stated that on the desktop, one of the biggest advantages of space lies in its size and thereby in its potential of allowing many objects to be brought together and to be accessible. As far as the surface of a desktop becomes structured, this structure remains rather vague. In “thought structuring” scenarios, on the other hand, space is used in order to purposefully arrange and structure knowledge artefacts. Due to the iterative nature of many of these thought structuring processes, said techniques allow the depiction and the incremental development of connections between objects. In the thought structuring examples, with very few exceptions, objects are related to each other by explicitly comparing them, which, on the other hand, means that the space which is used for the knowledge structuring purposes is not structured by itself.

The background a mind map is created on, for example, is just unstructured space which remains unstructured during the whole process. Simple examples of a structured space can be found in the metaplan scenario where during the process of working with the cards, the pinboard gets structured by dividing it into spatially visible regions. In the semantic space scenario whether or not the resulting map is based only on the interrelations between the objects or if there is a spatial structure beneath those relations depends on the way it is used. In this respect, creating a semantic space is like working on a desktop where the surface can be explicitly structured, but other kinds of structures and even pure chaos is also possible. In contrast to these techniques based on unstructured or only sparsely structured space, the techniques I present next do not only use space as their medium, as an unstructured plane on which objects can be arranged but they rather rely on space itself to be structured. I will argue that, by doing so, space itself can become a means of differential experience.

Figure 17 shows a map of the district of Paderborn. Such a map can be used as a means of differential experience for a number of scenarios. Assume, for example, that the map is used as a background for objects which are labelled with the names of the mayors of all towns and municipalities within the district. Every mayor-object is positioned according to the area of the town on the map. The result of these placements alone can be the source for differential experiences, as it allows knowledge workers to “look up” the mayor of a town or municipality by referring

### 3 Differential Experience and Meaningful Use of Space

to the map. Manipulations of the spatial positions of the objects, the removal of objects or the insertion of a new object, have semantic meaning as they correspond to new information about the political situation or, more likely, a change in this situation, e.g. when, for whatever reason, there is a change within a local administration. The map and the distribution of objects in relation to it allows for further differential experience when additional information is made perceivable through properties of the objects. In figure 17, the objects are colour-coded according to the party affiliation of the mayors. This colouring reveals a distribution which can be the foundation for further differential experience as it can, for example, encourage the formulation of hypotheses about voter rationale.



Figure 17: A map of the district of Paderborn as a structured background for objects representing the mayors of towns and municipalities in that district. The objects are coloured according to party affiliation. Red indicates a mayor is a member of the SPD, black is for the CDU, and grey-coloured objects refer to mayors without any party affiliation. (Background map by Hagar66, published in the Wikimedia Commons)



The district map is a means of differential experience, which can be confirmed by checking the requirements for differential experience.

It is *expectation-independent*: The placement of the objects is independent of what one thinks about it and relies only on the data about a property of what the object stands for, in this case on publicly available data of election results.

It is *ghost-free*: The placement of the mayor-objects only depends on the given data, and there is nothing in the placement process which could cause such a placement apart from these data.

It is *non-interfering*: In the example, objects which represent people are places on a map according to data about the mayors of the towns and municipalities within the district. These data are completely decoupled from the placement, which means by placing the objects, no mayoralty will change<sup>132</sup>.

Arranging objects in relation to a map, of course, is quite a simple example of a spatial structure because the structure itself is given through the topology of the environment. Creating a meaningful structured space for more complex and more abstract purposes is not an easy task. The challenge knowledge workers are faced with when creating a proper spatial structure is somewhat comparable to finding the right way of solving a mathematical problem. Knowing what information you have and knowing what information you want to know is, of course, necessary in order to perform a calculation, yet it needs quite an amount of work and knowledge and maybe a lot of muddling around to figure out which kind of calculation can solve the given problem. As you cannot just take numbers, perform random calculations and expect a meaningful result, of course, you cannot place objects in relation to randomly created structures and expect meaningful results.

---

<sup>132</sup>Even if it were possible to “manipulate” the mayoralty of a town or municipality by manipulating the objects on this structure, it would still be non-interfering as the process of placing the objects according to who is mayor in which town or municipality does not already change these facts.

### 3 Differential Experience and Meaningful Use of Space



Figure 18: One of the boards in the room management office of Paderborn University

	EIM-E	EIM-E	EIM-E	MB	MB	MB	MB				EIM-E	MB	MB	MB	EIM-E	EIM-E	MB	MB	EIM-E
	168	258	88	128	179	262	105	80	80	46		40	28	70	50	80	80	50	
	P 7203	P 7201	P 6203	P 6201	P 5203	P 5201	P 1417	P 1101	P 1102	P 1.2.21	P 1401	P 1408	P 1408.1	P 1418	P 1508.2	P 1509	P 1510	P 1611	
7 - 8	Montag	Montag	Montag	Montag	Montag	Montag	Montag	Montag	Montag	Montag	Montag	Montag	Montag	Montag	Montag	Montag	Montag	Montag	Montag
8 - 9																			
9 - 10																			
10 - 11																			
11 - 12																			
12 - 13																			
13 - 14																			
14 - 15																			
15 - 16																			
16 - 17																			
17 - 18																			
18 - 19																			
19 - 20																			
7 - 8	Montag	Montag	Montag	Montag	Montag	Montag	Montag	Montag	Montag	Montag	Montag	Montag	Montag	Montag	Montag	Montag	Montag	Montag	Montag
8 - 9																			
9 - 10																			
10 - 11																			
11 - 12																			
12 - 13																			
13 - 14																			
14 - 15																			
15 - 16																			
16 - 17																			
17 - 18																			
18 - 19																			
19 - 20																			

Figure 19: A detailed view of one of the boards in the room management office at Paderborn University

More complex and real-world examples of spatial structures which have been well designed are the structures which can be found attached to the walls of the room management office at Paderborn University. These structures, which are shown in figures 18 and 19 basically are big timetables in which every column represents a room, and every row represents a time slot. In figure 18 you can easily make out the five working days of a week as five groups of rows.

Figure 19 shows a detailed excerpt of one of the structures. Little yellow and red cards are attached to the timetable-matrix which forms the background. Yellow cards represent reoccurring assignments while red cards symbolise individual assignments which are valid for certain dates only. The structure in figures 18 and 19 constitutes what in this thesis I call a *structured space*. Positions in space correspond to properties objects need to have at this position. There is a correspondence between the room assignment of a lecture and the position of its respective card on the wall. One direction of the correspondence, the mapping of room assignments to positions of the cards, easily reveals which rooms are still available and which are not and would, if someone was not careful, also indicate a “double booking” as it would result in two cards in the same cell of the matrix. In this respect, the structure is a means of differential experience, yet the structures of the room assignment office are not only means of differential experience but also are a means of manipulation. This means that the other direction of the correspondence, the mapping of the positions of the cards to room assignments, is also defined. This provides meaning for the insertion of new cards or the movements of existing cards from one slot to another slot. Inserting a new card into the ensemble creates a new room assignment, whereas moving a card from one position into another changes the room for the lecture accordingly.

#### Structures in Space vs Structured Space

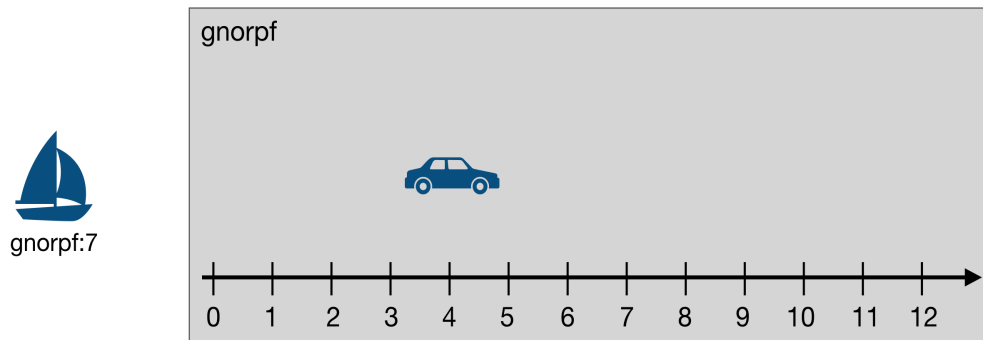
The timetables on the walls in the room management office are examples of structured spaces. I distinguish these structured spaces from other spatial structures in which an object arrangement relies on the comparison of the objects it is composed of. If for example, one is spatially ordering the names of one's relatives by birth date, one creates a structure which extends in space and which has a semantic order that has been translated into a spatial order by following formal rules. As the surface this list of names is created on is not itself structured, the person creating the list rather maps the semantic relations the objects have to each other, whether or not one person is born earlier or later than the other, to spatial positions between the objects. When a new person is to be inserted, one cannot refer to the surface in order to know where to put it but has to understand the structure and then has to consider the "values" and the positions of the other objects in order to know where to put the new one. I call such a structure which extends in space but which does not refer to an inherent structure of this space, a *structure in space*. When instead of creating a list, the same names are put onto a timeline they are structured in relation to a *structured space*. In order to know where to put a person's name on the timeline, one cannot refer to the other objects but has to refer to the surface structure which contains the timeline and thereby contains a perceivable hint on how the object structuring works and thus where to put the object<sup>133</sup>.

An interesting aspect of structured spaces, which distinguishes them from structures in space is the fact that its structures are themselves spatially perceivable. While, of course, the process behind the creation of a graph or a tree structure can be a means of differential experience, its inner relations, the rules according to which objects are positioned on the surface, are not spatial by themselves, meaning these kinds of structures do not explain themselves spatially.

---

133 A possible objection against this argumentation would be that the space on which the ordered list of birth dates is created does indeed have an inner structure in the way that in such an "ordered list space" it is clear that those spatial areas further down refer to later birthdates than those at the top. This argumentation, though, does not contradict the fact that in order to find out the placement of an individual object in the list, there is no spatial property which would provide the necessary information. Additionally, the information which led to the hypothesis that places further down refer to later birth dates than those at the top can only be derived from examining the existing entries in the list. The spatial surface itself cannot provide this information as it is just a blank, neutral plane.

Using structured space, e.g. by positioning objects in relation to a matrix or spatial regions, on the other hand, the structuring mechanism is itself spatial and thereby perceivable. Even when there is not a single object positioned in relation to the structured space, the structure itself is perceivable and reveals how it can be used.



*Figure 20: Though purpose and meaning are unknown, this spatial structure provides signifiers which suggest how it can be used.*

This, of course, does not mean that a spatial structure is completely self-explaining but in contrast to structures in space, where the construction mechanism is hidden, in spatial structures it is at least perceivable. While following the terminology introduced by Norman (2013), every surface affords the placement of objects on it, structured space also provides signifiers<sup>134</sup>, hints if you will, which tell a knowledge worker where to place his object according to which criteria, so structured space to some extent is self-revealing with respect to its inner workings. Figure 20 shows a spatial structure whose purpose and meaning remain unclear, yet the way it has been created gives some hints on how it may work. Someone who has already seen a number ray can recognise the inner workings of this structure. The background structure consists of a number ray which spans the numbers from 0 to 12 and is labeled with gnorpf, so even not knowing at all what it is for and what gnorpf means, one can tell that there is one object with a gnorpf of 4. Similarly, one can imagine where the sailing boat object depicted on the left-hand side, which happens to have a gnorpf of 7, has to be placed in relation to the depicted axis.

<sup>134</sup> Norman (2013), pages 10-20

### 3.5 Of Artefacts and their Genesis

While in this thesis, especially for the semantics I describe in chapters 4 and 5, I focus very much on structured spaces, I would like to stress that this does not mean that spatial structures are less important than these. One reason I can focus on structured space and describe its semantics is that in order to create structured spaces a great part of the thinking process which is necessary to structure a knowledge work environment has been “pulled upstream”. Thought has been put into the creation of the structure, so, later on, it can be spared to a degree that the remaining evaluations can even be computerised. To create a meaningful structured space, a complex thought process is necessary, and within this thought processes, structures in space may play an important role. Structured spaces and the way they are created are an example of to what extent products and their genesis often have different characteristics.

It may be a common mistake to assume that products such as a software system or, for example, this thesis in form and characteristics are similar to the circumstances of their creation or to interim products which were needed during their creation. The example of a mathematical proof can illustrate this. A proof, in essence, is a written down form of argumentation. The validity of a claim, often expressed as a mathematical formula, is shown using a number of argumentation steps which can be logically combined to a conclusion that proves the claim. While these steps can be thought through one after the other, their order in the proof does in no way indicate that the mathematician who came up with the proof had exactly these thoughts in exactly this order. Not only does a proof not include dead ends the mathematician might have run into, but the order of thoughts in general may have been completely different than the order of the steps of the final argumentation. Interim versions of the proof may have included detours or imperfections in the argumentation. These traces of the thought process of a mathematician are wiped out when writing down the final proof. The same difference can be observed when it comes to programming. Dijkstra (1970) compared the design of programs to a necklace where

“the top pearl describes the program in its most abstract form, in all lower pearls one or more concepts used above are explained (refined) regarding concepts to be explained (refined) in pearls below it.”<sup>135</sup>

---

135 Dijkstra (1969), page 87

This resembles the popular *top-down* structure. While many programming scholars interpreted this structural paradigm as a guideline for programming in a top-down way and even encouraged that programming should be taught that way, Dijkstra clearly stated:

“Pearls in a necklace have a strict logical order, say “from top to bottom”. I would like to stress that this order may be radically different from the order (in time) in which they are designed.”<sup>136</sup>

So while top-down may be a good structure for software system layers or even a good structure for the source code of a program, the order of program development as well as the order of teaching programming principles might be, or even should be, very different.

At the end of the 1970s, Christiane Floyd criticised the predominant phase-oriented approach to software development<sup>137</sup> which included the creation of requirements definitions, functional specifications, and design specifications early in the software development process. These specifications were the main references which software programmers developed and checked their programming against, leaving the presumed users out of the loop. Creating software this way often led to computer systems which, while fulfilling the requirements described in the documents, failed to meet the demands of everyday use. Instead of creating fixed specifications early on, Floyd suggested “evolving documents” which are to be adapted frequently during a process-based software development process. This is necessary because, as she explained in Floyd (1992), users of computer systems “do not analyse requirements [but] construct them from [their] own perspective”. Those requirements “reflect differences in perspective and are subject to temporal change”. Keil<sup>138</sup> expanded on what Floyd (1987) described as complementary product-oriented and process-oriented perspectives on software engineering by proclaiming a complementarity between artefacts and their genesis<sup>139</sup> both in learning as well as in design processes.

---

136 Ibid.

137 See, for example, Floyd (1981), Floyd (1987) and Floyd (1992)

138 See Keil (2007) and Keil (2011)

139 Keil initially spoke about a complementarity between product and process (“Produkt-Prozess-Komplementarität”) but later decided to speak about genesis instead of process as in today’s discourse the word process is often used in the sense of an inflexibly modelled chain of events which is precisely not what Keil and Floyd were thinking about when speaking about processes and their characteristics.



The complementarity between an artefact and how it came into existence, its genesis, can be explained by revisiting the aforementioned process of creating mind maps. While a final mind map is a tree structure which thereby represents hierarchical relations between the elements, interim stages of the process included a brainstorming phase which by definition is not hierarchical. Later stages include interim maps in which aspects may appear twice or in which design decisions might contradict each other. All of these interim products, while not meeting the criteria a mind map must fulfil, are necessary to create a good mind map.

As indicated above, the described complementarity between the characteristics of a product and those of its genesis give an interesting perspective on the creation of structured space. While, as a final product, structured space must meet certain criteria, which will be discussed in detail in the next chapter, the creation process of such a structured space may well include arrangements with quite different characteristics. A simple example can illustrate this. Assume you have a bunch of papers you want to sort according to what they are about. You put a single paper to a random position of your desk. Now you examine the next paper, have another look at the already placed paper and decide whether those two belong together. If they belong together, you cluster the two papers by putting them on top of each other, thereby creating a pile. If they do not belong together, you create a new one-element pile by placing the paper at another position of the desk. This process is repeated for every paper. While looking through the different piles during this process, you may need to separate one pile into two or combine two piles into one. During the whole sorting process, these operations are repeated a number of times, so spatial arrangements and the granularity of the piles are quite dynamic at first. After a while though, the clusters themselves become stable, and the distribution task becomes much easier as you do not have to look into each and every pile every time but can just put a paper into the *region* it belongs to. The clustering process which started as a process consisting of many comparisons among the objects which are to be structured resulted in the emergence of structured space which by definition works without such comparisons.

The intellectual clue in this switch from clustering to positioning in relation to structured space is the abstraction from the actual objects of the arrangement by finding those properties which decide whether or not an object belongs to be in one pile or the other and assigning those properties to spatial regions. This way,



the object placement becomes independent of existing objects. When instead of “I put this object where similar objects are” one can say “I put this object where the other objects with the xyz property are”, one can just remove the “other objects” from the sentence and end up with “I put this object where the xyz property is”, thereby assigning the property directly to a region of the surface. For more complex structures, such a process of building a structured space might be a little more complicated and require more thought and time. Nevertheless, in every case, a formal, structured space is created through a thought process which happens according to completely different rules than those of the structured space. When focusing on structured space here, I focus on features of the spatial structure as an artefact. This does in no way mean I despise the process from which these artefacts result, but it is a humble insight that, as an engineer, I can only competently speak about the artefacts and their properties.

## **3.6 The Similarity of Structured Space and the User Interface**

In this chapter, especially in its last section, I explained that structured space can be used as a means of differential experience. Using structured space, one can perceive and manipulate affairs which are not perceivable or manipulable otherwise. Object positions in relation to a structured space are meaningful as they make the object position stand for object properties thereby making them not only accessible for differential experience but also allowing them to be manipulated through a spatial movement.

User interfaces provide users with perceivable and spatially manipulable objects. Aside from purely graphical objects or elements of text for which the manipulation mostly concerns their arrangements on a page or a canvas, many applications provide a number of objects which carry attributes that have to be visualised as well as manipulated using the user interface. A photo management application, for example, allows photo metadata to be manipulated, an application managing the data of pupils is, among many other things, used to assign pupils to classes, a todo application has to have means of setting the status of a task to ‘todo’, ‘doing’, ‘done’ or ‘rejected’ and a music playing application like iTunes not only has to have means of assigning songs to playlists but also manages the status of the songs which either have never been played, are currently playing or have

already been played before. In all of those cases, a user interface must have means of visualising these attributes in order to make them perceivable by the user. In this respect, an interface can be interpreted as a means of differential experience. Like in knowledge work this *means of differential experience* is used to make affairs visible which are not visible by themselves. User interfaces of course are not passive in nature. They do not only visualise an attribute or a status but, in many cases, they also allow its manipulation. In some cases, the purpose of manipulating a status or attribute is served by the attribution itself while in other cases an attribute or status change triggers the application to do something. Considering the description of those user interfaces and the use cases of structured spaces in knowledge work given in this chapter, one can easily see that they both are “tools” for a similar purpose of translating invisible properties into a visible form and by allowing this visible form to be manipulated allowing the manipulation of the otherwise non-accessible properties. Following this argumentation, it stands to reason that structured spaces cannot only be a means of differential experience in knowledge work contexts but can also fulfil this role as a user interface technique. This hypothesis is investigated in the following chapters of this thesis.

## 4 Structured Space and Semantic Positioning

In chapter 3, I have shown that in many knowledge work scenarios space is purposefully structured in order to provide differential experience about object properties or about properties an object stands for. In examples like the room management scenario, where moving a card in relation to the matrix structure implies a change in a room assignment, I have demonstrated that the structures cannot only visualise properties but can also be used to modify these properties. In the closing thoughts of the chapter, I have suggested that these characteristics of making otherwise inaccessible properties both perceivable as well as manipulable resembles the requirements for some user interface scenarios which have to enable the task of assigning and visualising attributes of virtual objects. The consequence of this thought is the hypothesis that spatial positioning techniques which are used in knowledge work scenarios can be adapted to be a new user interface technique.

In a first step to such a user interface technique, in this chapter, I provide a (formal) semantics<sup>140</sup> for structured spaces. This semantics describes how the structures within a structured space make attributes and position correspond to each other. In the second part of the chapter. I define positioning objects in relation to such a structured space as semantic positioning and will argue that for such a semantic positioning a number of different evaluation semantics are possible depending on the context the positioning technique is used for. In chapter 5, Responsive Positioning, which is explicitly tailored to its application in a user interface technique, will be introduced as one of these evaluation semantics.

The clue of the structured spaces I described in chapter 3.4.3 is that space is structured in such a way, one can determine properties of an object by looking at its position and, in turn, can determine a correct position for an object by

---

<sup>140</sup> I am aware of the problematic usage of the term “semantics” in this context. The term, which generally stands for “meaning”, is used in computer science even though computers have no sense of meaning. When a computer gets a computer program as its input, it does not understand its meaning, so it does not understand the program on a semantical level. When computer scientists nevertheless speak about the semantics of a program or the semantics of a programming language or an evaluation function, they mean formal transformation and processing rules which are applied to the program so in the end they refer to what effect a line of source code or a construct of a programming language has within the machine. When speaking about the semantics of structured backgrounds, semantic positioning, or Responsive Positioning, I use the word in this ‘formal’ interpretation.

identifying its properties. A structured space thus defines a relation between what is to be depicted, the object property, and how it is depicted, the object position. Such relationships between what is depicted and how it is depicted are subject to research in the field of information graphics, scientific graphics and diagrams<sup>141</sup>. All these terms are used almost synonymously. Richards (2002) defines a diagram as a graphical structure with “the ability to recognize in it spatial relations which in some way correspond to the relationship represented”.<sup>142</sup> Such a correspondence between ‘spatial relations’ and the ‘semantic relationship represented’ can be realised in many ways. In the context of the positioning techniques described in this thesis, it is the mapping of object properties or object attributes<sup>143</sup> to object positions and vice versa, which means that, in contrast to pure visualisations, in this thesis, I understand correspondence in a bi-directional way.

In chapter 3, structured space was introduced as space in which “meaning”<sup>144</sup> is assigned to spatial positions. Structures in space, on the other hand, were described as structures which merely extend in space without making spatial positions as such meaningful. On a city map, for example, the positions of pictograms of sights can be translated into information about the position of the sights within the city. In a list of important sights of that city, on the other hand, the space the list is written on does not convey any information about the items of the list. In order to determine the rank of an entry in the list, one cannot look at the properties of space but has to compare the entry with the elements present in the list. Though the list extends in space, there is no structure within the background space.

---

141 An overview of the characteristics of diagrams as well as different definition attempts can be found in Wöpking (2010).

142 Richards (2002), page 87

143 The terms “property” and “attribute” mainly have the same meaning. Within this thesis, I reserve the word attribute explicitly for non-graphical attribute assigned to virtual objects. A computer document, for example, can have a topic attribute.

144 Meaning is generally believed to be a social construct which thus cannot be “assigned”. Nevertheless, Engelhardt (2002), for example, uses terms like “meaningful space” when speaking about space in which a certain property can be determined by determining its spatial position relative to it. When speaking about meaning in my contexts, I refer to “having a property which is meaningful in a given context”, so the “meaning” of the position of a card in the room management office is the room assignment data which is meaningful in this context.

A number of authors have made attempts to categorise or systemise basic structures or have investigated techniques according to which objects are placed during knowledge structuring processes. *Richards (2002)*, for example, defined three basic “modes of organization” for diagrams:

- *grouping; where elements imply belonging, e.g. by sharing a common colour, shape or enclosing boundaries*<sup>145</sup>
- *linking; where elements display connectivity, e.g. by a means of a joining line*
- *variation; where elements suggest value, e.g. by changes in size, distance from a baseline, or intensities in colour.*<sup>146</sup>

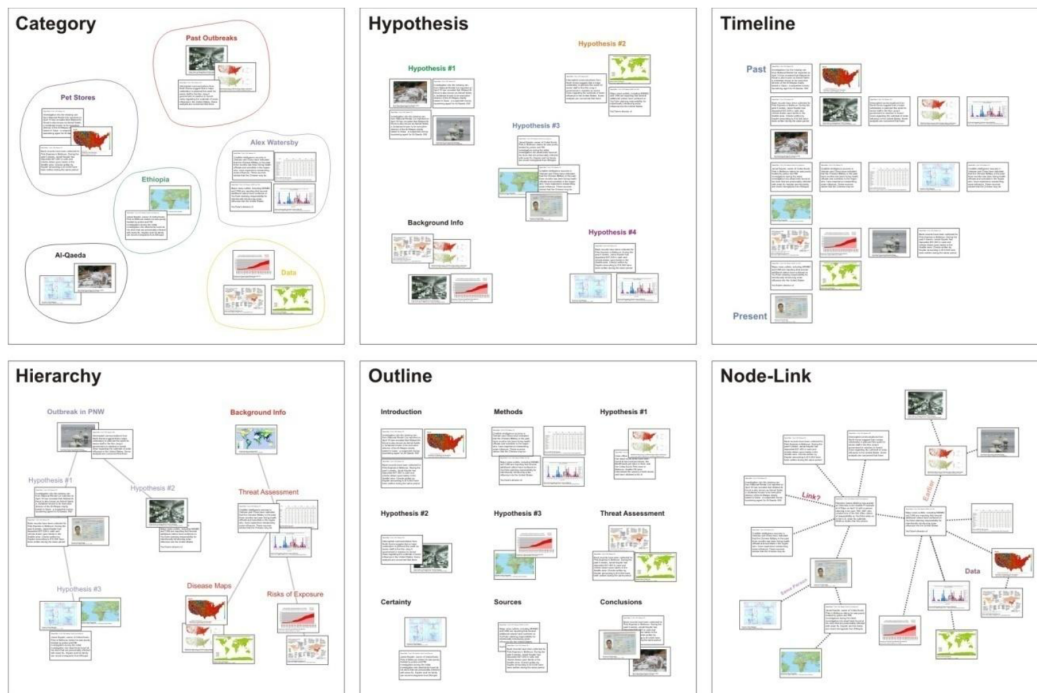


Figure 21: “Synthesis [of] organizational metaphors gathered from interview and experiment evidence” (figure and description from Robinson 2008).

In contrast to Richards, who explicitly wanted to offer a “terminology for discussing diagrams” and thus tackled the problem from a rather theoretical standpoint, Robinson (2008) chose an empirical approach to find out the

<sup>145</sup> Later in the text, Richards also mentions grouping by spatial distance.

<sup>146</sup> Richards (2002), page 91

necessary graphical structures which can be used in support software for geovisualisation projects. Participants of tests he conducted were given a number of heterogeneous little pieces of paper with more or less relevant information on it. Using them they had to solve a task which Robinson considered to be realistic in the field of geovisualisation:

“Participants were instructed that an avian influenza outbreak had occurred in the Pacific Northwest and that their task was to develop hypotheses for the source of the outbreak using the artefacts and tools they had been provided.”<sup>147</sup>

As a result of his experiments, Robinson found a number of “organizational metaphors” which are shown in figure 21. Summing them up, three main “metaphors” can be made out:

- *grouping* according to categories of information, by hypothesis or by the sections of a typical report,
- *sorting* of single artefacts or whole groups of artefacts according to the timely order of events or according to the outline of a typical report, and
- *connecting* in order to define hierarchical groups or in order to visualise semantic relations in a node-link structure

The “organizational metaphors” and “modes of organization” described by Robinson and Richards mostly remain in the realm of *structures in space* (see chapter 3.4.3) and thus are based on relations between the individual objects themselves, yet Robinson gives an indication towards structured space in a side note on observations he made during his test:

“Participants indicated groups by collecting artefacts in close proximity on the workspace, and often chose to draw lines around the objects to develop ‘regions’ to separate groups from one another.”<sup>148</sup>

In this chapter, I base my definition of structured space on the work of Yuri Engelhardt<sup>149</sup>, as his work supports the differentiation between structured space

---

147 Robinson (2008), page 25

148 Robinson (2008), page 100

149 I base my definitions mainly on Engelhardt (2002) but also consider a previous article (Engelhardt (1999)) and additions he made in (Engelhardt (2006)) and (Engelhardt (2007))

and structures in space through his distinction between object-to-object relations on the one hand and object-to-space relations, on the other hand. Engelhardt intended to describe information graphics as a language so, similar to a written language, one is supposed to be able to dissect scientific graphics into parts in order to determine their structures and semantics. Because of this approach, his definitions are more extensive and all-embracing than what I need for my definitions. It is not in my interest to extensively describe structures I happen to find out there, but I rather need to develop rules and concepts which I can use constructively in order to determine the building blocks for the creation of structured space. Thus, in the following subchapter, I explain those parts of Engelhardt's definitions on which I can base my own theoretical framework, which I will describe in detail in chapter 4.2.

## 4.1 Engelhardt's Language of Graphics

Large parts of Engelhardt's thesis (Engelhardt 2002), especially those parts I refer to here, describe a syntax for information graphics. Like the grammar of a language using which one can analyse a sentence and determine its structures and elements, Engelhardt's grammar promises to allow the same for non-interactive information graphics<sup>150</sup>.

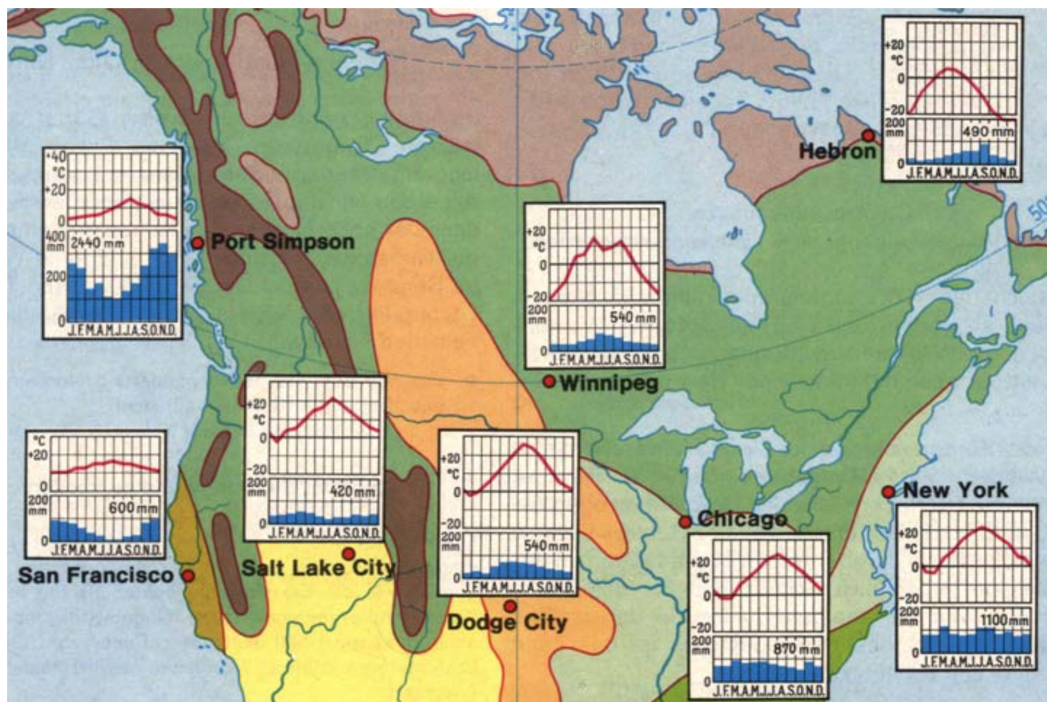


Figure 22: Map fragment from a vegetation map of North America with annual temperature and rainfall charts (Original source: Degn et al. (1973), page 5), taken from (Engelhardt 2006)

<sup>150</sup> Strangely enough, the concept of evaluable object positions, which I present in this thesis, while being highly interactive, more resembles non-interactive information graphics than what is often called “interactive visualisation” as, for example, described in Brodbeck et al. (2009). This is because interactive visualisation in Brodbeck's sense does not mean interacting with the visualisation in the sense of interacting with the presented objects or the data but merely means changing the parameters of the visualisation like, for example, zooming in or imposing a filter. Shneiderman (1996) condensed this approach in the visual information seeking mantra “Overview first, zoom and filter, then details-on-demand”.



As Engelhardt attempts to explain all instances of information graphics, he decided to take a hierarchical or recursive approach, which allows him to explain complex examples like figure 22 were within a meaningful space, the map, there are elements, such as the charts, which themselves create meaningful spaces.

The main element of Engelhardt's concept is the *graphic object* which can either be elementary or composite (see figure 23). Elementary objects are rather simple objects such as dots, text elements, or lines. Elementary graphic objects are part of a composite graphic object but do not provide any graphical space within themselves. In contrast to them, a *composite graphic object* is an object which within its boundaries provides a graphic space of its own and which thus consists of a number of sub-objects.

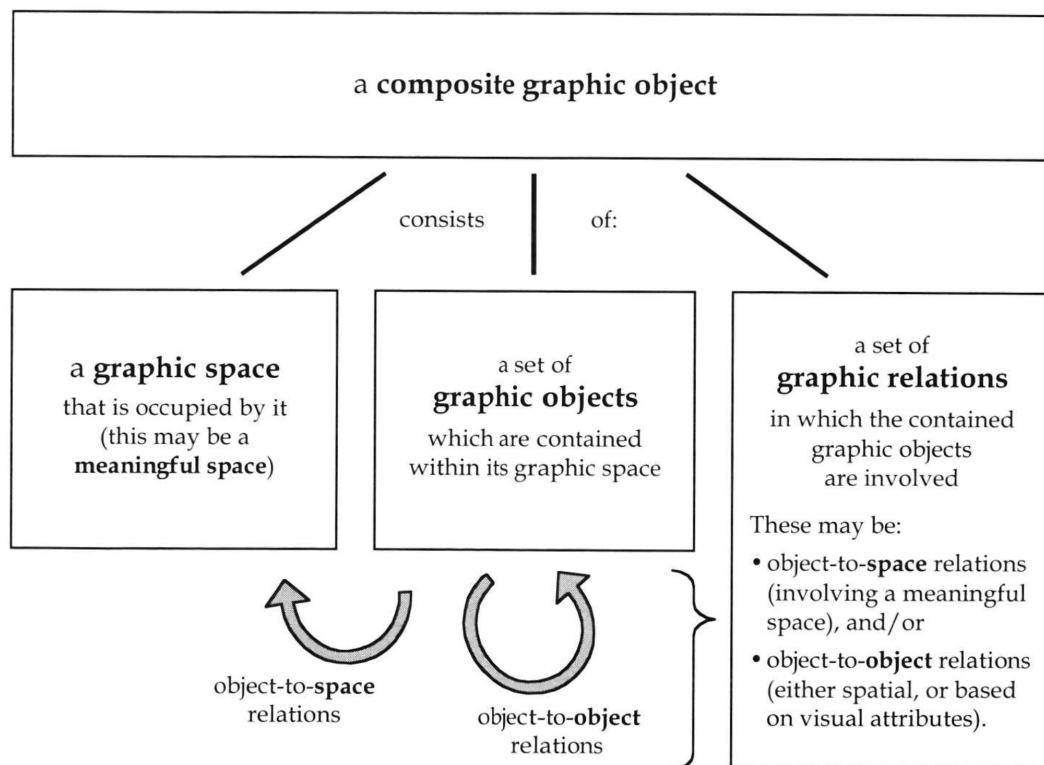


Figure 23: "The proposed syntactic decomposition of graphic representations. A graphic object may itself be a composite graphic object, thus this decomposition can be applied recursively" (figure and description taken from Engelhardt 2002).

In Engelhardt (2006), Engelhardt sums up this hierarchical structure as follows:

“We define a graphic space as the (‘canvas’-)surface that is occupied by a graphic object. Any object may contain a set of (sub-)objects within the space that it occupies. When this principle is repeated recursively, the spatial arrangement of (sub-)objects is, at each level, determined by the specific nature of the containing space at that level. In other words, an object may contain an internal space and (sub-)objects that are arranged within that space [...]. This principle can be repeated recursively: A (sub-)object may again contain an internal space and (sub-)objects that are arranged within that space.”<sup>151</sup>

Figure 22 as a whole, for example, is a composite graphic object which on the highest level provides a meaningful space, the map. Within the space of this composite graphic object, there are a number of graphic (sub-)objects. There is a grid, there are regions indicating climate zones, there is a combination of dots and texts which indicate cities, there are climate charts, etc. The objects within this graphic space are not randomly distributed<sup>152</sup> but have been purposefully positioned according to “a set of graphic relations in which the contained graphic objects are involved”. In the example in figure 22, the dots indicating major cities are positioned in relation to the geographical map in the background and thus relate to the position of the cities on the surface of the earth. This relation is what Engelhardt calls an *object-to-space relation*. A dot gains its meaning through its relation to the map structure in the background. Other objects are positioned in relation to other objects rather than to a spatial property. The climate charts, for example, are bound to the city-indicators. This type of relation is an example of what Engelhardt calls an *object-to-object relation*<sup>153</sup>.

On a detailed level, the elements within figure 22 can be described using the same model. While a lot of elements like the dots or the grid are elementary and cannot be further divided, the climate chart of Winnipeg, for example, while being a sub-object of the whole composite graphic object named “figure 22” is itself a composite graphic object. It consists of two different charts which share a

---

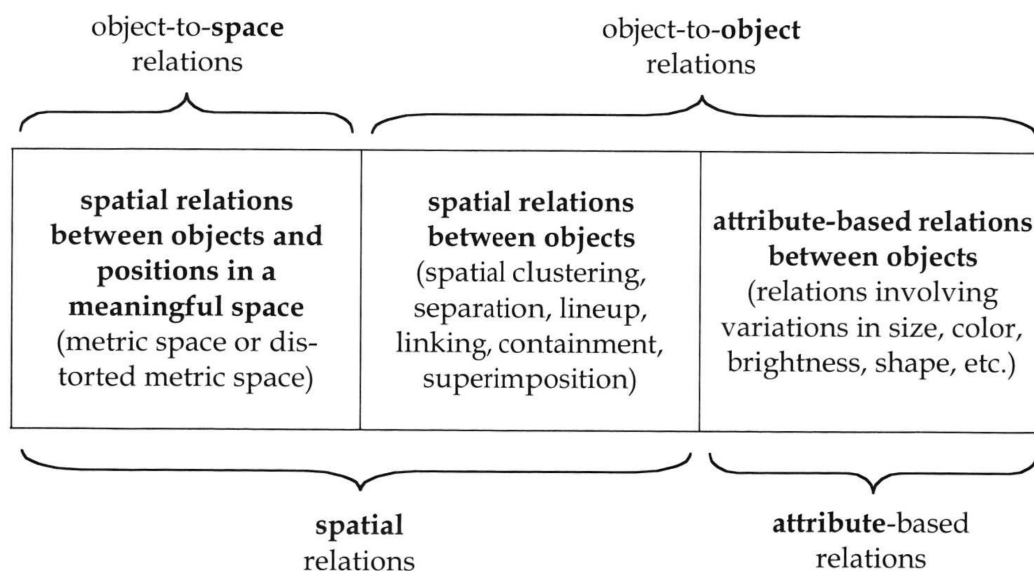
<sup>151</sup> Engelhardt (2006), page 104

<sup>152</sup> Graphical objects can, of course, be subject to no graphic relation at all, see (Engelhardt 2002, page 31): “A spatial structure that involves neither a meaningful space nor meaningful object-to-object relations is an arbitrary spatial structure”.

<sup>153</sup> See Engelhardt 2002, page 30

common time axis. The upper chart represents medium temperatures while the lower chart represents average rainfall per month. Within these structures, the elements, such as the curves, the grids, the labels, or the bars, are sub-objects which again are arranged according to object-to-object and object-to-space relations.

From my point of view, it is Engelhardt's biggest merit to introduce a clear distinction between object-to-object and object-to-space relations to the discussion. It proves to be quite helpful for the definition of semantics for structured space and Responsive Positioning, as this distinction corresponds to the distinction between structured space, where the space the objects are positioned on is itself structured, and structures in space, where space itself is not structured even though a structure is occupying space.



*Figure 24: Engelhardt's distinction between object-to-space and object-to-object relations on the one hand and spatial relations and attribute-based relations on the other hand. (Figure taken from Engelhardt 2002)*

The characteristic feature of an object-to-object relation is its implied necessity of having to compare objects with each other in order to find out their relation. According to figure 24, object-to-object relations include both spatial relations between objects, as well as attribute-based relations between them. The district map scenario in figure 17 is an example where attribute-based relations play a role. The mayor objects in the example were not only spatially positioned in

relation to the map in the background, which is an object-to-space relation, but also relate to each other in the similarity or the difference of their colourisation. As this thesis aims at describing a technique which couples object attributes and object positions, in the remainder of this thesis such attribute-based relations do not play any role and thus are neglected.

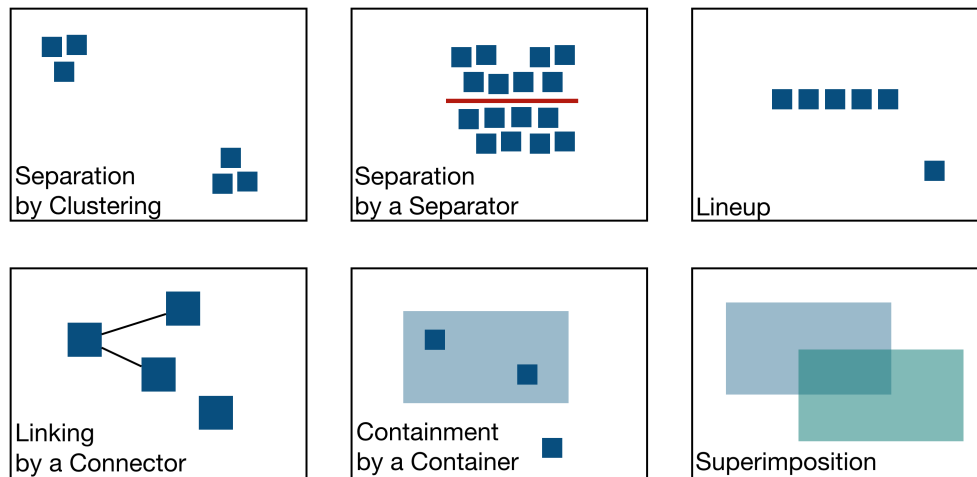


Figure 25: Separation by clustering, separation by a separator, lineup, linking by a connector, containment by a container (all according to Engelhardt 2002)

Engelhardt presented six different types of spatial object-to-object relations<sup>154</sup> which are depicted in figure 25 and can be described as follows:

- *Spatial clustering*<sup>155</sup> means placing one object near a position where other objects already are. Spatial clustering makes use of a “within-group proximity” and a “between-group distance” which results in perceivable clusters of objects.
- *Separation by a separator*<sup>156</sup> is somewhat similar to separation by spatial clustering except that instead of relying on the difference between proximity and distance, an explicit separator object is introduced into the arrangement.

<sup>154</sup> Engelhardt’s selection of object-to-object relations, which he described as “owing their existence to Gestalt principles of visual perception” (page 50), basically is a synopsis of literature research (ibid.) concerning such relations. His naming scheme can be questioned as it contains a number of inconsistencies. (e.g. to be consistent with “separation by a separator”, instead of “spatial clustering” it should be “separation by clustering” or even “separation by spatial distance”).

<sup>155</sup> See Engelhardt (2002), page 32

<sup>156</sup> See ibid., page 34

- In a *Lineup*<sup>157</sup> objects are arranged in a way they build a string of objects, giving every object except those at the ends two neighbours.

For separations and lineups Engelhardt defines both sorted and unsorted interpretations. In sorted separations, the order of the subsets is itself subject to interpretation while in unsorted separations the order is arbitrary. Similarly, in an ordered lineup the order of the lined up elements either is subject to interpretation or it is not.

- *Linking by a connector*<sup>158</sup> can make direct relations between objects visually perceivable. In order to link the objects, a new, distinct object, the connector, has to be created and positioned accordingly. A connector always connects two graphical objects which, at least when speaking about them on a meta level, are often called nodes. Visually a connector mostly is a line or an arrow.
- *Containment by a container*<sup>159</sup> means the use of a container object which includes a number of sub-objects which appear to be *within* the container object. If a container were moved or otherwise manipulated, the objects within the container would move or change with it.
- *Superimposition*<sup>160</sup> means objects are arranged on different virtual layers, so one object appears to be in the foreground while another object in the background is partially occluded by the foreground object.

Characteristic for all kinds of object-to-object relations, as the name says, is its reference to other objects. In order to put an object into its correct position within a sorted lineup, one has to look at the lined-up objects and compare them with the object to be inserted. The same is true for clustering. In order to know into which cluster of objects a new object is to be put the objects in the clusters have to be considered.

Using Engelhardt's object-to-object relations, one can create what I introduced as *structures in space* in chapter 3. In these structures, object positions are determined by object-to-object comparisons. To be able to create *structured*

---

157 See *ibid.*, page 36

158 See *ibid.*, page 40

159 See *ibid.*, page 44

160 See *ibid.*, page 50

*spaces*, which Engelhardt calls “meaningful spaces”, the concept has to be extended to object-to-space relations. Engelhardt describes that in contrast to object-to-object relations, to place an object in meaningful space no other objects have to be put into consideration.

“The graphic space [...] is a meaningful space if spatial positions in it are subject to interpretation regardless of whether or not there are graphic sub-objects present at those positions. To say it differently, a meaningful space is a graphic space that involves an interpretation function from position in space to information.”<sup>161</sup>

Engelhardt substantiates his distinction between object-to-object and object-to-space relations by defining what an object is “anchored to”:

“In object-to-object relations (spatial clustering, separation by separators, lineup, linking by connectors, containment, superimposition), an object is anchored to one or more other objects. For example, a connector is anchored to the nodes that it connects, and a label is anchored to the node that it labels. In object-to-space relations on the other hand, an object is anchored to one or more spatial positions in the involved (distorted) metric space.”<sup>162</sup>

Within this thesis, I consider an object to refer to exactly *one* position in space which means that object sizes or other object properties concerning object shapes are disregarded. As a consequence of this restriction, in the context of this thesis, an object-to-space relation effectively means that an object is anchored to *exactly one* position in a meaningful space. The positioned objects thus are what Engelhardt defines as point locators<sup>163</sup>.

“A point locator is anchored to a specific point in a meaningful space. Examples: a church symbol on a map, a dot in a scatter plot, [or] city-markers in [a] map [...]. Usually the area occupied by a point locator is centered on the specified point in meaningful space. Another possibility is that the point locator has a kind of 'vertex' or 'tip' that is

---

161 Ibid., page 54

162 Ibid., page 54

163 An example of objects which are anchored to more than one position in space are surface locators, like a lake in a map or the vegetation zones in figure 22, or line locators, like a river in a map or the temperature curves in the same figure.

positioned on the specified point in meaningful space[...]. A point locator is basically free in its shape and size.”<sup>164</sup>

Notice that while positioning according to a meaningful space is carried out without caring about the positions or even the existence of other objects, the resulting object distribution, of course, does allow for comparisons of the resulting arrangements. In other words, even though the individual objects are placed without considering other objects, a resulting object distribution can nevertheless provide differential experience regarding object relations which are revealed by this placement. Positioning objects representing the invention date of important computer technologies on a timeline, for example, happens according to object-to-space relations, so each position is determined without having to care about the objects already present, yet as a result of these placements, objects may, for example, occur in clusters or reveal certain patterns. Such a pattern can provide differential experience and thus allows for the formulation of hypotheses as it, for example, uncovers that after the presentation of a certain technology, the rate of development increased significantly.

In meaningful space, there is a defined relation between the “meaning” of an object and its spatial position. Engelhardt describes three kinds of formal relations which he calls *basic syntactic structures*:

- In *metric spaces*<sup>165</sup> metric aspects of object positions in relation to space are meaningful. A very simple form of a metric space is a metric axis, such as a timeline. Engelhardt calls this a “basic metric space” and distinguishes it from composite metric space, which is a combination of two or more basic metric spaces, e.g. a coordinate plane, and “integral metric space” like a topological map “in which all geometric properties of Euclidian space are subject to interpretation”.
- *Distorted metric spaces*<sup>166</sup> in Engelhardt’s sense can be thought of as metric spaces which were printed on a “rubber sheet” and then stretched non-homogeneously preserving both order and approximate directions but

---

<sup>164</sup> Ibid., page 75

<sup>165</sup> See *ibid.*, page 57. The term “metric space” is imprecise from a mathematical point of view as what Engelhardt defines here is not only metric, but also meets the criteria of a *normed* vector space (see Christensen 2010).

<sup>166</sup> See *ibid.*, page 65

not preserving spatial distances. An example given by Engelhardt are London-style tube maps.

- *Partitioned graphic spaces*<sup>167</sup> are spaces which contain regions or segments which are meaningful as a whole. An example of a partitioned graphic space is a matrix, in which for every cell there is a different semantic interpretation. Within one cell though, the semantic interpretation is the same regardless of which position is considered.

It should be noted that in contrast to earlier articles about meaningful space<sup>168</sup> in his 2002 thesis, Engelhardt decided not to include separated graphic spaces into his selection of syntactic structures for meaningful spaces. As Engelhardt introduced his distinction between object-to-space and object-to-object relations which include separations by separators as well as containers he saw a problem of different possible ways of interpreting the same graphical structures and therefore decided:

“‘Partitioned graphic spaces’ [...] can be analyzed as being created through spatial clustering, lineups and separations by separators, and do therefore not appear to be indispensable ingredients of a minimal set of basic syntactic structures.”<sup>169</sup>

I decided not to follow this argumentation, mainly because Engelhardt’s reasons for his decisions do not apply to my lines of thought. Engelhardt pursues a completely different target than I do. He defines his “basic set of syntactic structures” to be able to provide a grammar for *all* existing information graphics. Just as the grammar of a language, this grammar should not contain any ambiguities (though grammars for languages often do), which means using the grammar it must be possible to *deconstruct* a given information graphic in a way which allows for only one interpretation. I use Engelhardt’s concepts differently as I do not want to deconstruct given graphics but rather want to *construct* structured space. For this, I use only a subset of Engelhardt’s definitions and within this subset, I have a priority on object-to-space relations. For the semantics I describe in chapter 4.2, every position is only interpreted as such an object-to-space relation. Object-to-object relations do play a role in the construction of

---

167 See *ibid.*, pages 54,55

168 e.g. Engelhardt (1999)

169 Engelhardt (2002), page 55



structured spaces but never are subject to any formal evaluations. For this reason, a partitioned graphic space cannot mistakenly be interpreted as an object-to-object relation. Apart from that, even Engelhardt himself is not consistent in his own decision. In a later chapter about the “degree to which aspects of space can be meaningful”<sup>170</sup> he describes an “unordered separation” (e.g. a table or matrix) as making “use of the separateness of subspaces”. In this definition, he is referring to space but is not considering the positions of other objects and is neither calling the table or matrix a container nor describing it as a bunch of separators. In Engelhardt (2007) he then provides a “typology of meaningful graphic space” which includes “group space” which he explains as “columns and rows in a table [or] any other meaningful spatial grouping”.

Engelhardt defines that for each meaningful space, there is an “interpretation function from position in space to information”<sup>171</sup> which effectively means one can determine information about an object in a given context by determining the spatial position of the object in relation to its meaningful space according to the syntactic structure of that space. In simple examples, the syntactic structure of a meaningful space would just be one of the basic syntactic structures described above, yet in many information graphics there are more complex structures which are “constructed from two or more basic syntactic structures through simultaneous combination and/or nesting”.<sup>172</sup> Of these two alternatives, for my purposes, nesting can be neglected<sup>173</sup>, so only the *simultaneous combination* is relevant:

“In a simultaneous combination of basic syntactic structures, a set of graphic objects simultaneously participates in two or more basic syntactic structures, at the same syntactic level of object decomposition.”<sup>174</sup>

Figure 26 shows an example of such a combination of syntactic structures. A number of icon-style objects which represent people are positioned in relation to a

---

170 See Engelhardt (2002), page 70

171 Engelhardt 2002, page 54

172 Ibid., page 79

173 Nesting (ibid., page 80) means that composite graphical objects are interpreted as a single graphical object on a higher level. The climate charts in figure 22 are examples for that. While providing meaningful space within their respective spatial areas, from the perspective of the map, they are single objects which relate to the map or the objects labelling cities on the map.

174 Ibid., page 79

meaningful space which provides two simultaneous graphic relations, so each object participates in a metric space which relates the object position to information about the age while at the same time it participates in a partitioned graphic space which relates it to information about gender.

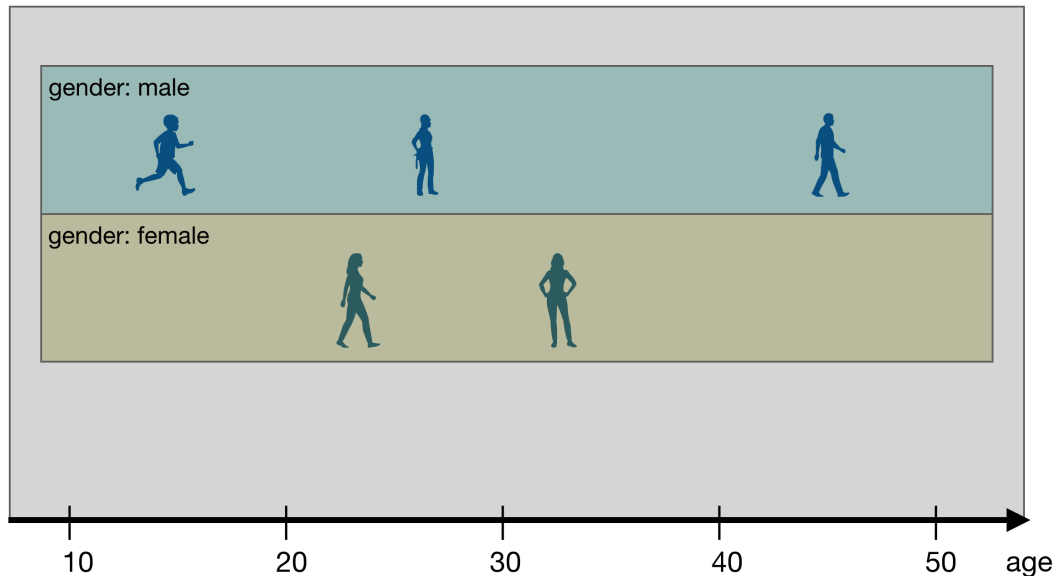


Figure 26: An example of objects which are participating in two basic syntactic structures in a simultaneous combination.

As syntactic structures determine, which spatial properties of a graphic space provide meaning, objects relating to this space are thus, to use my terminology, structured by it. Among the objects which can be present in a meaningful space, Engelhardt distinguishes between information objects, reference objects and decoration objects<sup>175</sup>.

**Information objects** are those objects which are positioned in relation to meaningful space. They are what is depicted or, as Engelhardt puts it, they are “the graphical objects that would have to be adjusted if the *information* (data) that one intends to represent would change”<sup>176</sup>. In figure 26, the blue icon-style objects are information object.

In order to see how an object relates to meaningful space, it must not only be possible to perceive the object itself, but there must also be visual clues within the

---

<sup>175</sup> See *ibid.*, page 127,133

<sup>176</sup> *Ibid.*, page 129

meaningful space which convey the syntactic relations. In Engelhardt's concept, this is achieved through the placement of reference objects.

**Reference objects** are objects which enable the viewer to interpret the position of the information objects. Especially interesting in my context are the *spatial* reference objects<sup>177,178</sup> which “mark a meaningful space”. Spatial reference objects are necessary in order for a meaningful space to be perceivable. Examples, according to Engelhardt, are axes and their annotations, grid lines, or familiar landmark features on a map. If partitioned spaces had been considered as object-to-space relations, the boundaries of these partitions surely would also have to be considered spatial reference objects<sup>179</sup>. In figure 26, the axes, as well as the coloured boxes, are spatial reference objects. They indicate, which area a syntactic structure applies to and, in case of the metric space, also indicate the respective values at different spatial positions.

**Decoration objects** are objects which, from a structural point of view, have nothing to do with the meaningful space as such and, if at all, only refer to it on a symbolic level. This would, for example, be the case when an information graphic contains a text element explaining the reason for the graphical arrangement. Such a text does not refer to what is described by means of spatial arrangement but only through its textual content. Other typical examples of decoration objects found in many information graphics are ornamental images which have some relation to the topic depicted but are not part of the “meaningful” object arrangement itself. Figure 26, for this matter, does not contain any decoration objects.

---

<sup>177</sup> See *ibid.*, page 129

<sup>178</sup> Non-spatial reference objects are, for example, those object present in a map's legend.

<sup>179</sup> In Engelhardt (2002) they are considered a container object which he defined under the object-to-object relations.

### 4.2 Structured Space and Semantic Positioning

In this subchapter, I adapt Engelhardt's concepts introduced in the previous subchapter and thereby develop definitions for structured space as well as for semantic positioning. While my definitions are based on the work of Engelhardt, there are distinct differences not only in my choice of words but also in conceptual details. In order not to impair the readability of my own definitions, I decided to spatially separate the levels of argumentation. The main text of this subchapter, to the most part, contains only my definitions as well as some necessary comparisons with related concepts while relations to Engelhardt's work have been moved into the footnotes.

*Semantic positioning*, in simple terms, means positioning objects in relation to structured space. This structured space, in essence, is quite the same as what Engelhardt described as meaningful space.

While semantic positioning can be performed using analogue media, the room management scenario at Paderborn University (see figure 41) can again prove as an example, the technique, of course, rises to its full potential in a digital environment. First of all, by using digital media the process of creating the structured space, as well as positioning objects in relation to it, benefit strongly from the potential of rearrangeable and manipulable graphical objects. In addition to that, digital media could, of course, also allow for automatic evaluations of attribute changes or manipulations of object positions, though, as will be clear at the end of this chapter, additional specifications have to be made for this to be possible.

By using the term semantic positioning in order to describe the positioning of objects in relation to structured space, I redefine the term of a related concept by that name which has first been introduced by Erren&Keil (2006) and later been described in detail by Erren (2010), who defined:

“Semantic Positioning means that (media) objects gain semantic meaning, by their position in a perceivable context, alone. The context, as a dynamically adjustable arrangement of (media) objects, determines the semantic interpretation of position by mapping information dimensions to space. This allows the definition of

perceivable spatial semantic relations between objects by their relative positioning.”<sup>180</sup>

While not obvious at first glance, Erren’s definition of semantic positioning, in contrast to a number of the structures he proposes, is based almost entirely on object-to-object relations<sup>181</sup>. When Erren speaks about “relative positioning”, he explicitly refers to relations between objects. His spatial-semantic concepts<sup>182</sup>, namely distance, order, inclusion, combination, and path, underline this impression as all of these are defined between objects. His selection of “arrangement types”<sup>183</sup> though includes a “coordinate topography” and a “categorizing collection”, which both are object-to-space relations and can be directly translated to the terminology which will be developed in this chapter. Further arrangement types he suggested, e.g. ordered lists and relational graphs, on the other hand, do not refer to a structured background and thus will not be part of my definition of semantic positioning.

In contrast to Erren’s concepts, my definition of semantic positioning is based on structured space, thus on object-to-space relations, only.

---

180 Erren (2010) page 75

181 On an interesting side note, in the initial publication which introduced semantic positioning, Erren&Keil (2006), while not explicitly pointing it out, emphasised on spatial structures in the sense of object-to-space relations by describing the disadvantages of “techniques simply based on PicMents”, which in Engelhardt’s terms would mean graphics based on object-to-object relations only. Erren&Keil speak about objects “overlaid in spatial areas” and claiming “the utilization of the spatial dimension within the corresponding overlay of documents and graphical structures” to be the reason for coining the term in the first place.

182 See Erren (2010), page 88

183 See *ibid.*, page 89

**Definition:** *Structured space*<sup>184</sup> (or a *structured background*) is space which has an inherent graphical and thus visible structure<sup>185</sup> which enables the mapping of semantic attributes<sup>186</sup> to spatial positions as well as the mapping of spatial positions to semantic attributes. This bi-directional mapping is called *correspondence*<sup>187</sup>.

A mapping between attributes and positions for all practical matters implies the presence of *objects*<sup>188</sup> which are positioned in relation to structured space and for which such a correspondence between object positions and object attributes can be assumed. Whenever objects are positioned in relation to structured space, I speak about semantic positioning.

**Definition:** *Semantic positioning* means positioning objects in relation to structured space which provides a correspondence between positions and attributes, which means the position of an object can be determined according to its attribute values, and the attribute values of an object can be determined according to its current spatial position.

The objects which are positioned in a semantic positioning process are not part of the structured space. When objects are positioned in relation to structured space, they are graphically positioned in front of it, so when performing semantic positioning there always is a clear distinction between the background layer (the

---

184 Engelhardt uses the word “meaningful space” in order to describe this relationship. In order to avoid the somewhat problematic word “meaningful”, I stick with the more technical term “structured” which would be true even if the structure were completely meaningless or if there were nobody there who could assign meaning to it.

185 In contrast to Engelhardt’s hierarchical definition, I define a non-hierarchical (and therefore single) background. This allows me to clearly distinguish between the background, which is structured, and objects in the foreground, which are positioned in relation to the structured background.

186 Instead of speaking about the “meaning” of a position, which can be problematic in the context of data processing, I rather speak about an object attribute which corresponds to a certain position in structured space.

187 Engelhardt speaks about an interpretation function from position to information. As for me, this function has to be bi-directional, I chose the word “correspondence”. This choice of words has the advantage of avoiding the word “interpretation” which always is problematic due to its connotations.

188 In contrast to Engelhardt, I use the word “object” only for graphical elements which represent the content or are an object of an application (those objects Engelhardt calls information objects). The visible elements of the structured background (e.g. the lines which constitute an axis) are part of the spatial structures and therefore are not called object in the sense of the definitions given here.

structured space) and the foreground layer where the objects reside. Object positions always relate to the structured background and therefore, regarding their correspondence, are independent of the presence, the state and the position of other objects<sup>189</sup>. While objects have both a position and an extension in space, in relation to structured space they are interpreted as single points<sup>190</sup>, so for the correspondence between attributes and positions, the spatial extension of an object is neglected.

### Notes on how to “Read” the Examples

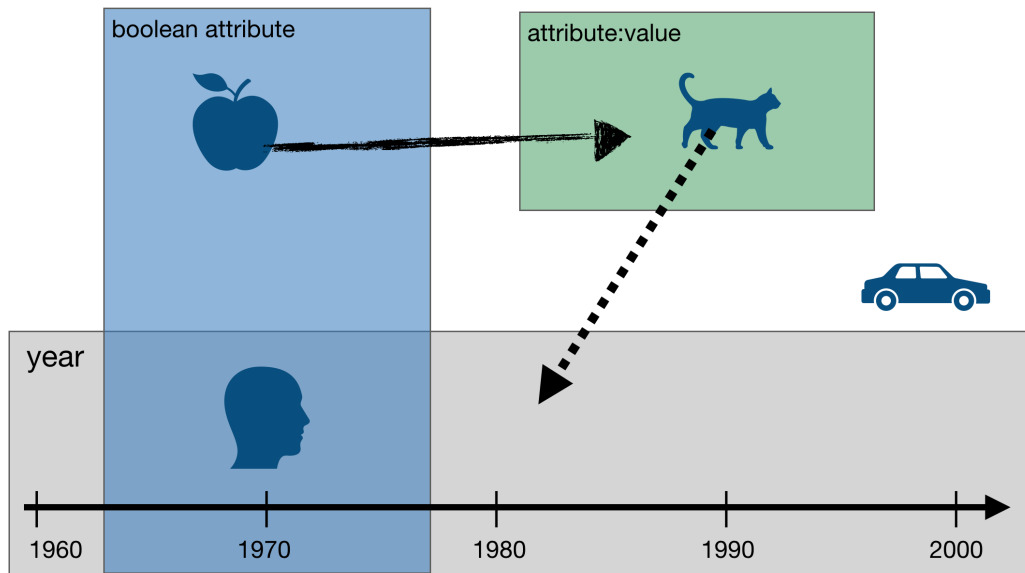


Figure 27: A typical depiction of the elements of a structured space, related objects and movement indicators

In the remainder of this chapter as well as in the following chapters, I present many examples which illustrate structures and applications for semantic positioning and Responsive Positioning. While in actual implementations, user interfaces based on these techniques may look different and probably would hide many technical details about their functionality, I decided that for the examples I am about to present here, I will always include a vast amount of indicators and

<sup>189</sup> This concurs with Engelhardt’s definition of meaningful space and thereby means that all spatial positions in relation to structured space are object-to-space relations.

<sup>190</sup> Engelhardt calls this a point locator, which is anchored to one specific point in meaningful space.

signifiers which make clear, which attributes are involved and which basic structure covers which area of space.

Figure 27 shows a typical example of how I present semantic and Responsive Positioning structures throughout this thesis. The structured space depicted in the figure consists of 3 basic structures. Each of these structures can easily be recognised by its scope which is visualised by an area of a common background colour as well as its bounding box, which is the outer limit of its scope. Every structure contains a number of labels which specify which attributes and which values or range of values the structure provides correspondence for. When a structure contains only a single label, this label represents the name of a boolean (or binary) attribute. The attribute value, in this case, is omitted in order to reduce visual clutter. Such a label implies that within the structure the attribute value is considered to be ‘set’, ‘true’ or 1.

Blue or red icon-like images represent (foreground) objects. Everything which is blue is an object which is subject to spatial manipulation by the user and by the semantics of Responsive Positioning system. If an object is depicted in red, this markup indicates that there is some kind of problem concerning this object in relation to the structured background. All other objects, which are neither part of the background nor the foreground are considered to be decoration objects. Arrows pointing away from the depicted positions of foreground objects indicate object movement. Arrows which look like being drawn using a pencil (————→) indicate an object manipulation by the knowledge worker or, in general, by a user of the computer system, whereas dotted arrows (······→) indicate object movements performed by the computer system.

### 4.2.1 Structured Spaces as Combinations of Basic Structures

Semantic positioning requires a structured space as a basis for its correspondence between attributes and positions. Such a structured space can be put together using one or more *spatial structures* (or structures for short). Each of these spatial structures defines its own correspondence between positions and properties which is indicated by visual features of the structures<sup>191</sup>. The correspondence of the axis in figure 28, for example, is indicated by a labelled arrow and a bounding box

---

<sup>191</sup> In Engelhardt’s terms, every spatial structure can be interpreted as a combination of a spatial relation and its spatial reference objects.



which indicates the scope of the structure. On a high level, there are only two *basic structures*: Axes, which provide a continuous correspondence between positions and attribute values, and regions, which provide a discrete correspondence in this respect.

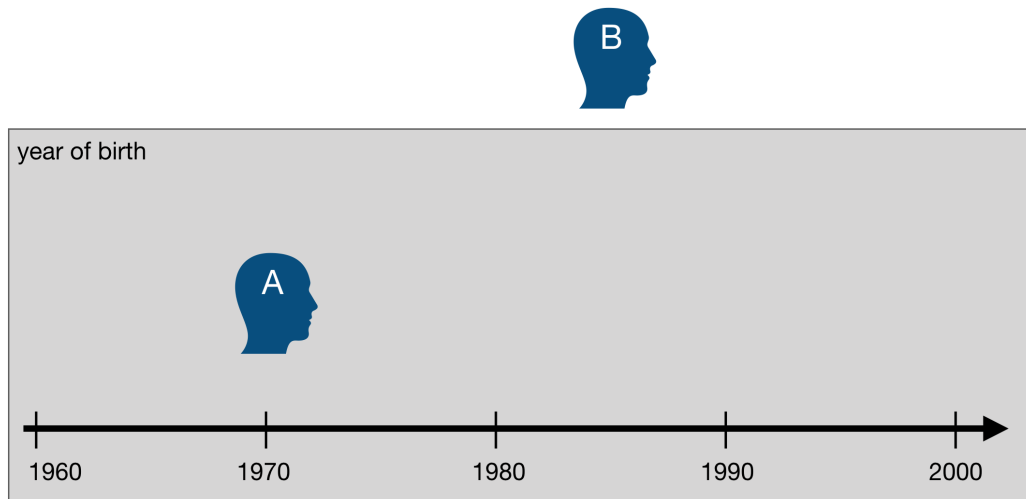


Figure 28: An axis providing a continuous correspondence.

An *axis* defines a continuous (normed) correspondence<sup>192</sup> between an attribute and a property. It is typically indicated by a line or an arrow which specifies what attribute the axis refers to and which attribute values it corresponds to at which position. An axis has a *scope*<sup>193</sup> in which its correspondence is defined. In the example in figure 28, object A lies within the scope of the given axis while object B does not. The position of object A according to the axis corresponds to a “year of birth” value of 1970.

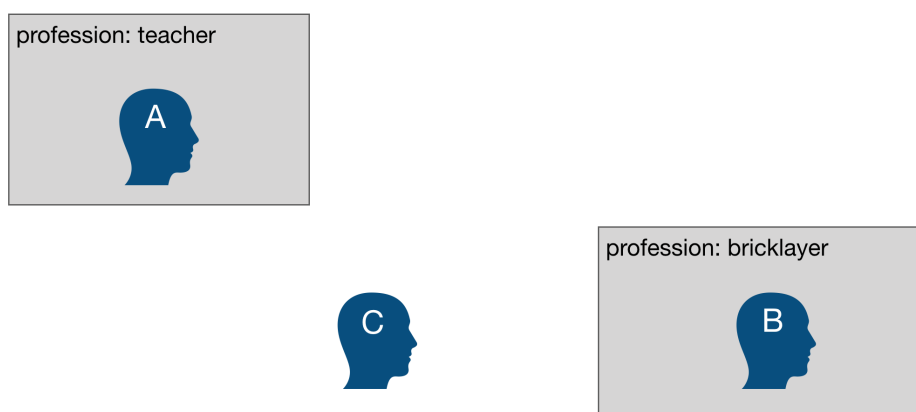
A *region* defines areas in space<sup>194</sup> and thereby describes a discrete correspondence between positions and attributes, which means that every position within a region corresponds to the same attribute-value combination, so, in turn, an attribute

<sup>192</sup> For structured spaces, I define only what Engelhardt calls “metric spaces”, which include non-linear (e.g. logarithmic) spaces. Distorted metric spaces in Engelhardt’s sense (which can be thought of as a metric space that was printed on a “rubber sheet” and then stretched non-homogeneously) are problematic for my purposes, as in such a space, defining a correspondence is almost impossible.

<sup>193</sup> Engelhardt supports a scope only indirectly as the metric space of an axis could be the space within a composite graphical object of a parent structure which would provide boundaries.

<sup>194</sup> This is what Engelhardt calls partitioned graphic spaces. In contrast to Engelhardt (2002), I consider these to be subject to object-to-space relations.

corresponds not only to a single position but to a whole spatial area. The two regions in figure 29 refer to the same attribute with two different values, so for both objects A and B, the attribute ‘profession’ can be determined according to their positions, where A corresponds to value ‘teacher’ and B corresponds to value ‘bricklayer’.



*Figure 29: Two regions provide discrete correspondence.*

Object B in figure 28 and object C in figure 29 are positioned outside the scopes of the spatial structures. The question whether a spatial structure does provide any kind of correspondence for these objects cannot be answered universally as it strongly depends on the semantics of object placements. This problem will be further discussed in the next subchapters.

Structured spaces can be built by combining structures of the aforementioned structure types.<sup>195</sup> Figure 29, for example, is an arrangement of two regions. The possibilities of arrangements are, of course, manifold. Many kinds of more or less arbitrary arrangements can be imagined and can be sensible in a certain context. While it is, of course, not possible to elaborate on all kinds of combinations here, there are a number of arrangements which are that common that most people would likely not even perceive them as combinations of structures but regard them as structures of their own. An arrangement of regions which refer to the

---

<sup>195</sup> In Engelhardt's terms arrangements are object-to-object relations. Not all of Engelhardt's object-to-object relations do make sense as structure-to-structure relations though. I can, for example, not imagine any use for clustered axes.

same attribute with different values in such a way that every region has a distinct left and right neighbour—a lineup of regions if you will—and where the boundaries of those regions collapse into one boundary, creates a spatial *segmentation*<sup>196</sup> or a *sorted segmentation* if there is an inherent order within the regions. The collapsed boundaries, in this case, can be interpreted as a visual hint, indicating that all regions refer to the same attribute. Regions referring to different attributes arranged in the same way would not be considered to be segmentations. Figure 30 shows a segmentation which refers to the ‘country’ attribute. The three distinct spatial areas defined by the segmentation relate to three different values.



Figure 30: A segmentation consisting of three segments

Two or more structures which overlap are called an *overlay*<sup>197</sup>. A position relative to such an overlay corresponds to all of the overlapping structures simultaneously. Graphically an overlay can be interpreted as basic structures being painted onto overhead transparencies which are put on top of each other. The top structure does not completely occlude the lower ones, which graphically supports that all overlaying structures are providing correspondence for the same positions.

An example of an overlay structure can be seen in figure 31. The hypothetical scenario behind this figure is a visualisation of the data of a measuring device which measured concentrations of some kind of toxic substance over a period of several weeks. The structured space in figure 31 consists of three spatial

<sup>196</sup> The word “segmentation” was used in Engelhardt (1999) for exactly this purpose.

<sup>197</sup> Engelhardt speaks about the superimposition of two objects as one of his object-to-object relations and calls the fact that an object within an overlay takes part in several correspondence contexts a “simultaneous combination of basic syntactic structures”.

#### 4 Structured Space and Semantic Positioning

structures. Each measurement object, which here is depicted as an X, corresponds to all three structures simultaneously.

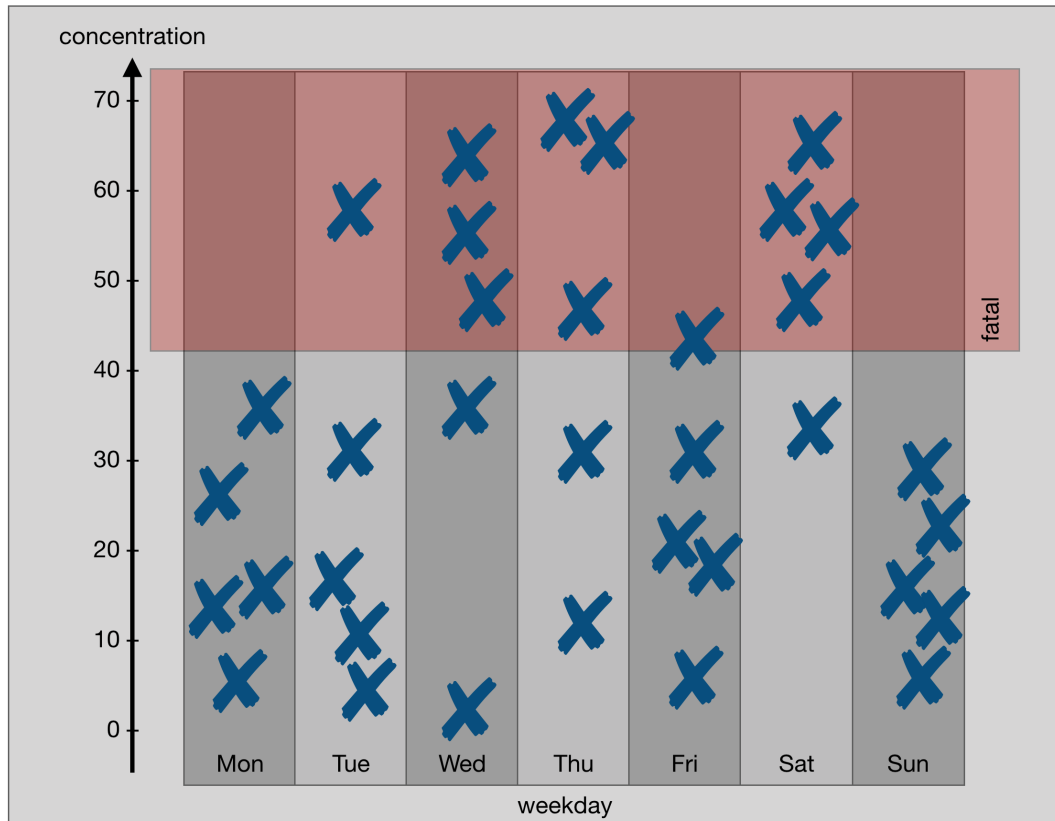


Figure 31: A complex example of overlaying spatial structures

First of all, the objects correspond to an axis according to their 'concentration' attribute. This axis structure is overlaid by a segmentation containing seven segments representing the days of the week from Monday to Sunday. Each measurement object carries a 'weekday' attribute which is set to one of these values and is therefore positioned in its respective segment, so each measurement horizontally corresponds to a weekday and, at the same time, vertically corresponds to its concentration.

A third structure is a region which spans only the upper part of the structured space. It refers to a boolean attribute 'fatal' which means that every position within this region corresponds to 'fatal', every position outside of it does not.

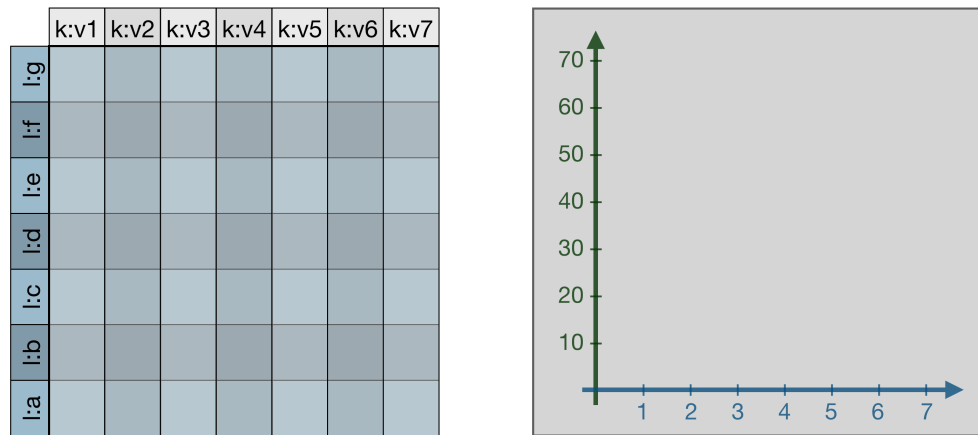


Figure 32: Matrices and coordinate planes can be explained as overlays of segmentations and axes

Analogous to a segmentation, which is such a common occurrence of an arrangement of regions that it can be considered a structure of its own, there are two kinds of overlays that are similarly common so they also can be considered as structures of their own. Both of these, matrices and coordinate plane, are depicted in figure 32. The combination of two segmentations sharing the same scope running in orthogonal directions while referring to two different attributes constitutes a *matrix*. In a great number of practical cases, such matrices are called tables. A good example of such a matrix or table is a school timetable where space is segmented according to weekdays horizontally and according to the periods of a school day vertically. The school timetable is a sorted matrix in which both rows and columns are sorted<sup>198</sup>. Note that the sorting does not affect the correspondence itself. In order to place “English” into the 9-10 AM slot on a Monday, for example, it does not matter if the matrix has Monday as the first, the second or the last column.

A similar overlay of two axes referring to different attributes and spatially pointing into orthogonal directions, while having the same scope constitutes a *coordinate plane*. Just as in a matrix, an object position in relation to a coordinate plane refers to two attributes at the same time, yet in contrast to a matrix, in a coordinate plane, both dimensions structure space in a continuous manner. As a consequence of that, when values for both dimensions are specified, a set of object attributes corresponds to *exactly one point* in the coordinate plane.

<sup>198</sup> There can also be matrices in which either the columns or the rows are sorted.

Figure 33 shows the full set of basic and common structure types which can be used to create structured spaces for semantic positioning. Complex structures like the district map in figure 17 can be created using these structure types when interpreting the areas representing the towns and municipalities as funny-shaped regions. Nevertheless, in the remainder of the thesis, I neglect such structures, as the provided set of orthogonally shaped structures provided in figure 33 is sufficient for all purposes described.

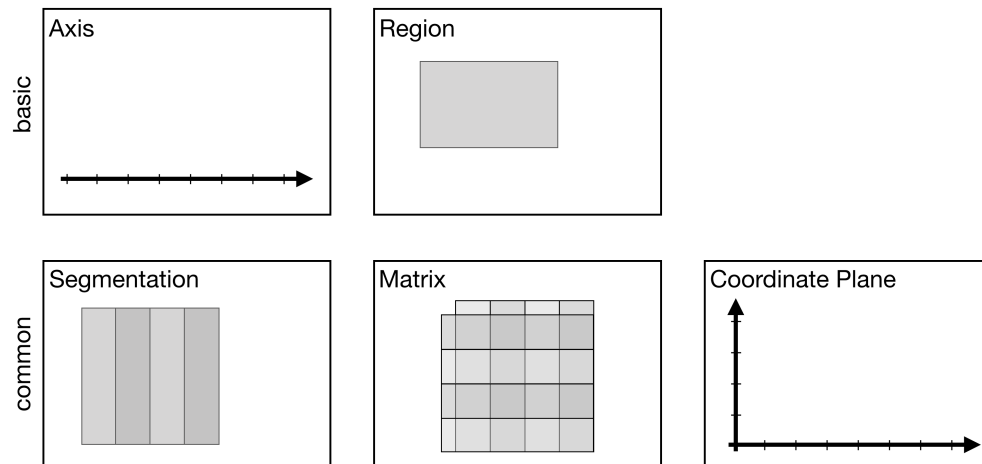


Figure 33: Basic and common structures are the building blocks for structured spaces

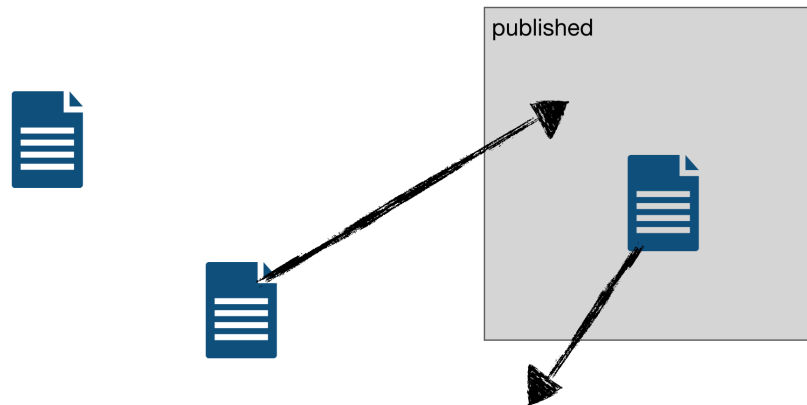
### 4.2.2 Semantic Positioning as an Umbrella Term

Notice that the definition of semantic positioning in chapter 4.2.1 only refers to being able to determine attribute values and positions and does not specify any actual attribute assignments, object movements or any other evaluation. The reason for that is the lack of a context needed to determine the appropriate evaluation semantics.

Whenever semantic positioning is to be used for an application or for evaluations in knowledge work scenarios, the semantics of object manipulations and attribute changes<sup>199</sup> have to be defined beyond the mere correspondence described so far. For every basic arrangement operation (create, insert, remove, rearrange and recompose, see chapter 3.3), as far as they are supported, there must be rules

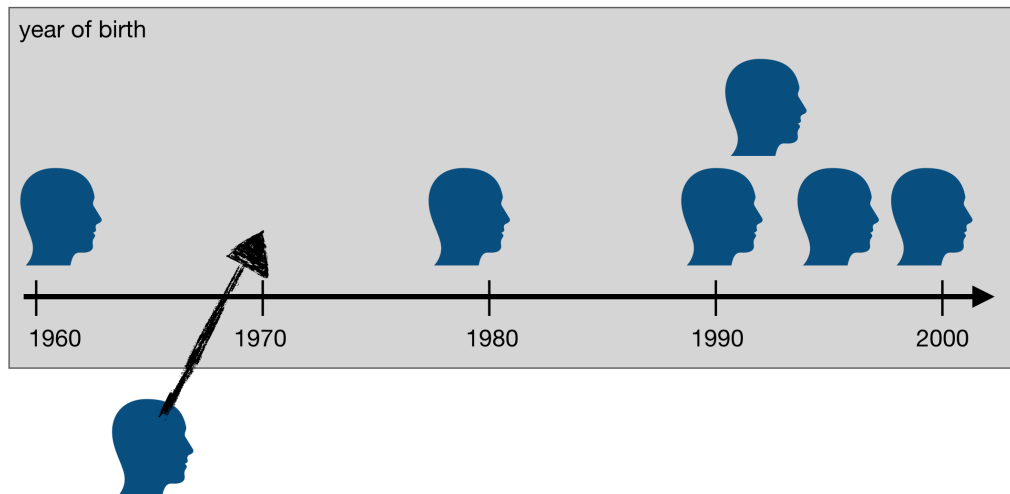
<sup>199</sup> Erren (2002) recognised the need for different evaluation semantics by referring to different “action schemes” (pages 136-140). In his thesis, he tacitly applies a number of these action schemes in different examples, which underlines their dependency on the context in which the positioning technique is used.

about how these operations correspond to the evaluation of attributes and positions. By looking at a number of examples, which are by no means exhaustive, I will show that these questions can only be answered knowing the application semantics or that they may even depend on the current situation.



*Figure 34: A very simple structured space. In this case, the semantics of positioning objects can easily be made out.*

### Semantics of object placement



*Figure 35: An object is dragged into the scope of an axis*

One of the most basic operations in semantic positioning is, of course, dragging an object to a spatial position which is within the scope of a spatial structure. Figure 35 shows a number of objects which are positioned in relation to a

timeline. A new object which was previously positioned outside this structure is moved into a position which allows determining its correspondence to the timeline. At first glance, one might think this is a simple case, but in fact, the semantics of putting an object into a position inside the scope of this structure is far from clear, as a number of interpretations are plausible:

- The object could get attributed with a ‘year of birth’ value of 1970, which corresponds to its new position. This would make sense in a data input or attribute assignment scenario.
- An evaluation function could compare the attribute value which corresponds to the new position (1970) with that one the object has as its own attribute (say 1960) and determine the deviation of 10. Such an evaluation could, for example, be used in a test scenario at school.
- Dragging the object into the scope of the structure could just “tell” the structure it should care about the object which would then result in an immediate movement of the object to the position which corresponds to its internal attribute of 1960. This would be a drag&drop operation which triggers a kind of context switch for the moved object.

### Semantics of object removal

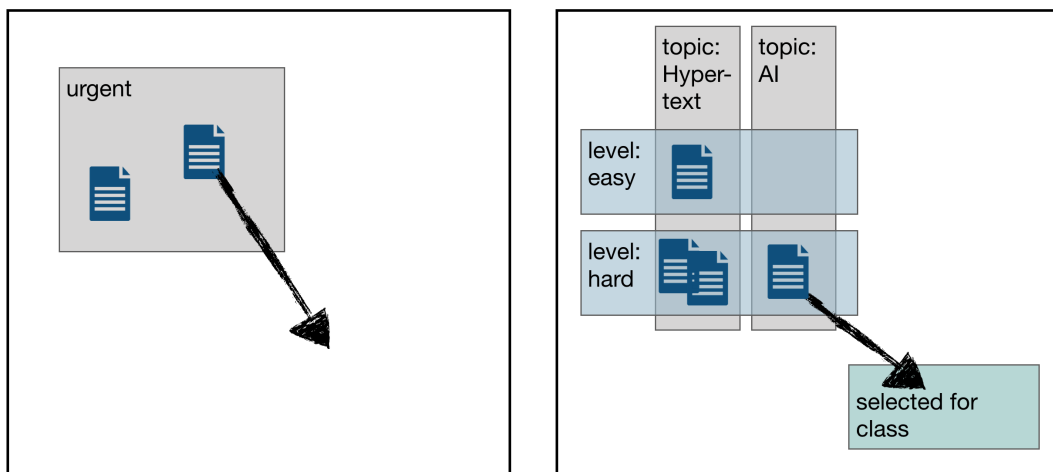


Figure 36: Two examples of objects being removed from the spatial structures they referred to.



Removing an object from a spatial structure is another field of ambiguity. Consider the two examples in figure 36. The left example shows a region which is labelled ‘urgent’ and which is supposed to contain objects carrying the ‘urgent’ attribute. Moving an object from this region into a neutral part of space would imply this ‘urgent’ attribute to be removed or unset. As a result, all objects within the region are urgent, all objects outside of it are not.

In the right-hand-side example of figure 36, one can distinguish two structures which are separated by distance. On the top-left, there is a structure combined of an overlay of regions in which documents are positioned according to topic and level of complexity. The region on the bottom-right, in contrast, does not refer to such library-oriented properties but refers to a “selected for class” attribute.

Applying the “removing from structure” logic to the left-hand-side example, moving an object from the top-left structure into the bottom-right region would implicate that the object loses the attributes the top-left structure corresponds to, as dragging it into the bottom-right region, of course, spatially removes the objects from the structure on the top-left so consequently their attributes are removed. Such an interpretation would not be very sensible in this context as one would not assume an object, which is likely only temporary moved into the new position in order to be assigned for classwork, getting its topic and complexity information ripped off. On the contrary, keeping this information would allow the semantic positioning system to automatically move the object back to its place in the top-left structure when needed.

Both examples, by the way, assume that structures provide a correspondence between position and attributes even in regions outside their scope. The position of an object outside a region corresponds to a negative attribute assignment meaning objects at this position may *not* have a certain attribute value. If spatial structures only cared about objects within their scope, removing an object from a structure would not have any consequences at all which in the left example of figure 36 would mean that objects could be made urgent through this structured space, yet the same structured space could not be used to unset the same attribute.

### Semantics of automatic positioning

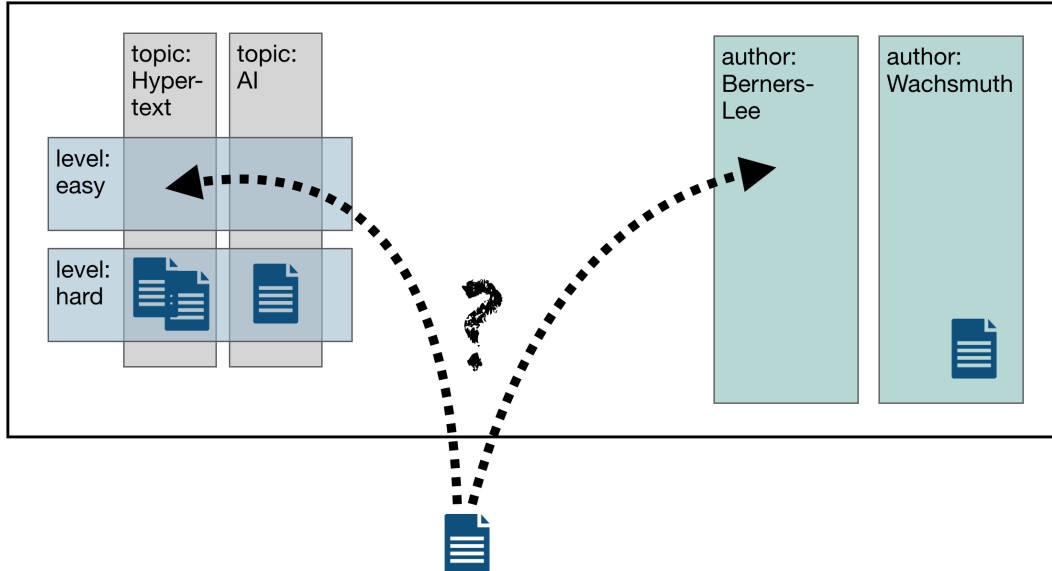


Figure 37: The automatic positioning of an object can be ambiguous.

As indicated, semantic positioning can, of course, be capable of automatically moving objects to their corresponding positions. There are situations where such a position can be determined without any problems, yet the object in figure 37, for example, could be meaningfully positioned in more than one spatial structure. Depending on the positioning semantics, the system could either choose one of the possibilities, let the user decide, or simply declare the object to be unplaceable. Again, which of these possibilities is the right one, very much depends on the context. In knowledge structuring contexts, for example, making users aware of the ambiguity and letting them decide what to do could be exactly that kind of differential experience they need as it could make them aware of the fact that the structured space they created does not reflect the semantic dependencies of the attributes of those objects they want to find something out about.

### Semantics of a change in the spatial background

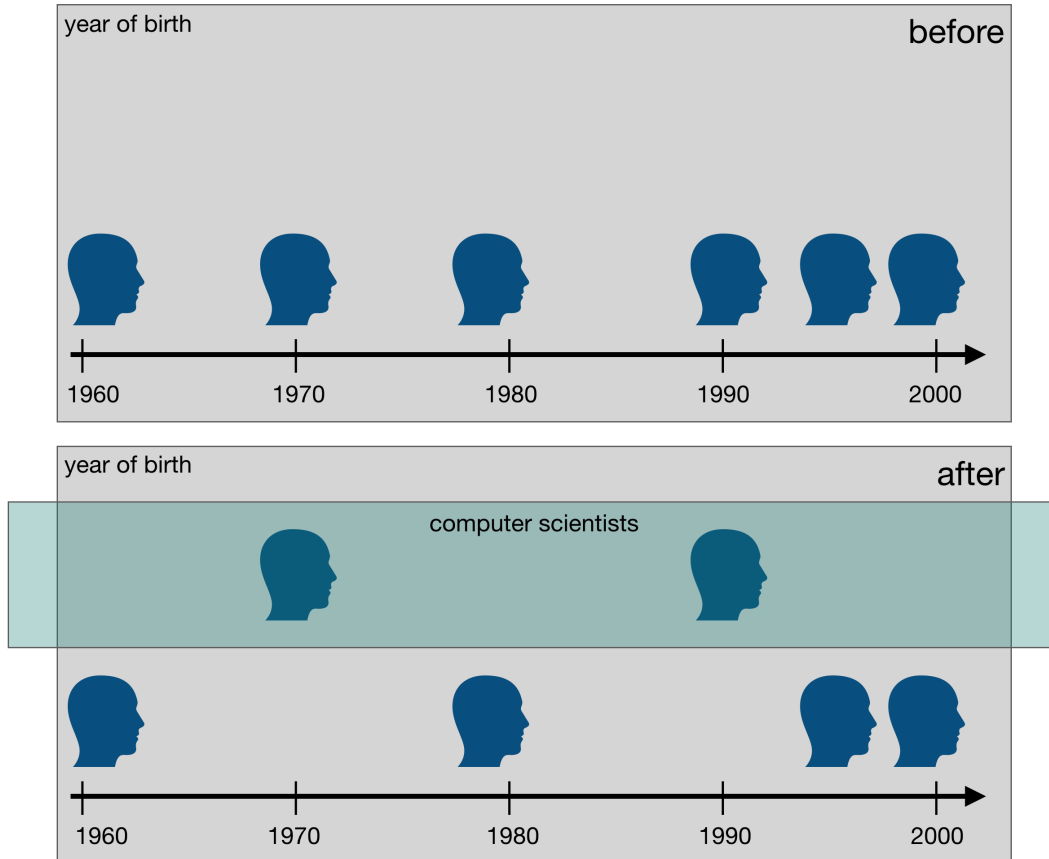


Figure 38: The addition of a region into the background of an existing arrangement serves as a filter for the content objects.

All examples mentioned so far considered modifications of the foreground objects while in the following examples, the structured background of existing arrangements is changed. Figure 38 shows objects positioned in relation to ‘the year of birth’ attribute on an axis. The lower part of the figure shows that the introduction of a new region into the spatial background can work as a filter or separator, dragging the objects referring to the ‘computer scientists’ property into its boundaries while leaving the other objects untouched.

Figure 39, on the other hand, shows a structure in which objects represent measurements of a certain poisonous substance. They are positioned according to the measured value and to the weeks the measurement took place. By introducing a new region (in red) which refers to a ‘fatal’ attribute these objects, which are

now positioned in front of the new region, are reevaluated, and thus the ‘fatal’ attribute is assigned to them.

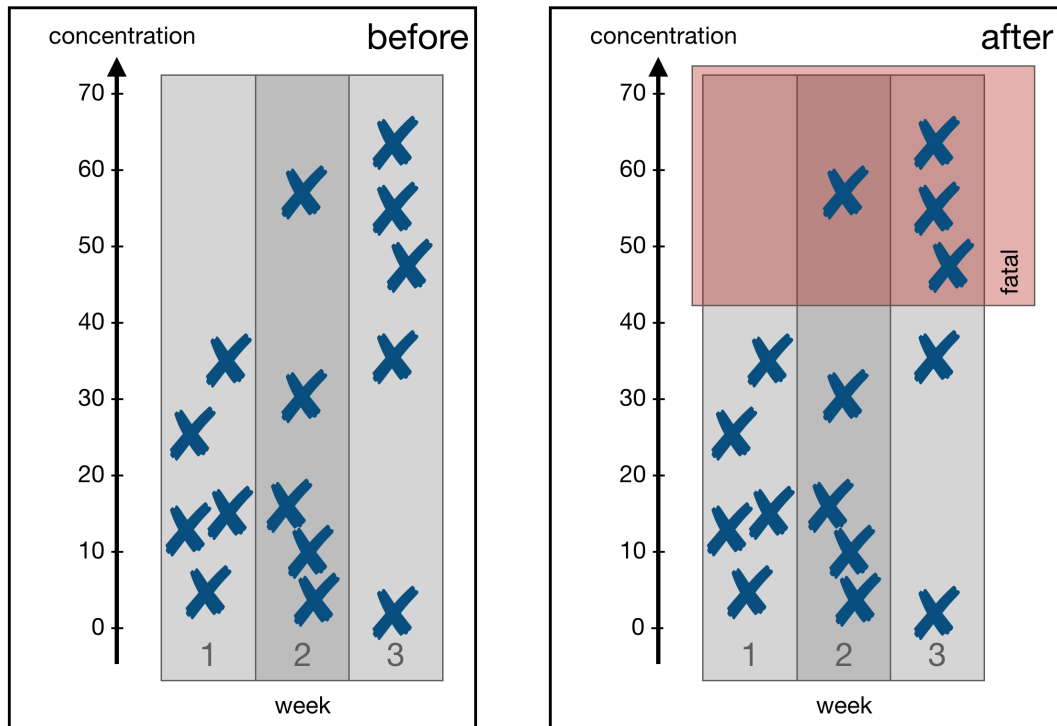


Figure 39: The addition of a region into the background of an existing arrangement results in a new attribute assignment for the objects now positioned within that region.

The two examples given here describe sensible reactions to changes in the background structures. From a formal point of view, the changes are quite similar. In both cases, a new region which overlaps part of the existing structure is added to the structured background, yet the described interpretation is very different. If the insertion of a new region into the structured background in figure 39 was interpreted like that one in figure 38, the existing measurement objects would have to disappear from their positions (assuming they do not carry the ‘fatal’ attribute already), rendering them effectively unplaceable. On the other hand, if in figure 38 the new region was interpreted like in figure 39, nothing would happen because no elements are positioned in that part of the background now covered by the new region.

The ambiguities in the examples presented above show that, while semantic positioning relies on a set of basic spatial structures for which a correspondence between positions and attributes can be described, its application in a context requires the definition of the semantics of manipulation for that specific context. In other words, there must be specifications for what happens in case of object movements, attribute changes and changes of the structured background. Considering this, I extend my definition of semantic positioning by explicitly including the possibility of evaluating the results of the described correspondence depending on the context:

**Definition – extended:** *Semantic positioning* means positioning objects in relation to structured space which provides a correspondence between positions and attributes, which means the position of an object can be determined according to its attribute values, and the attribute values of an object can be determined according to its current spatial position. The results of these determinations are subject to an evaluation which provide the concrete semantics specific for its intended purpose.

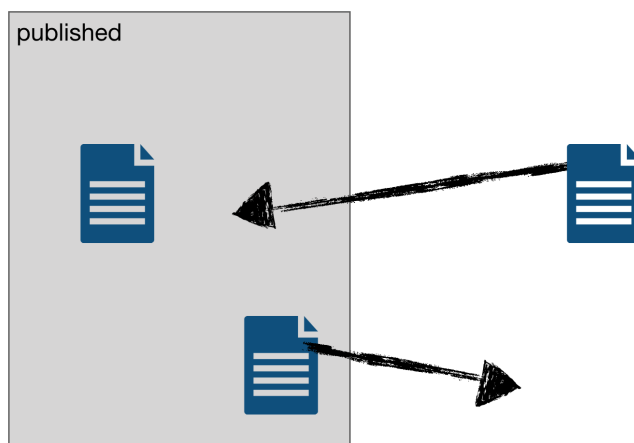
In the context of this thesis, the intended purpose of the semantic positioning implementation named Responsive Positioning is to be able to perceive and manipulate object attributes so that the object positions can be the interface for their object attributes. This purpose implies that every object position can be directly converted into an attribute set and vice versa, so that either one can be changed by changing the other. Only this way, attribute and position can virtually become the same. This “illusion” of being the same can only work when the conversions from position to attributes and from attributes to position are carried out responsively, which means they have to be carried out without any noticeable delay. This, of course, implies that every evaluation must be unambiguous, as any system feedback asking for clarification in case of ambiguity would destroy the responsiveness and thus would destroy the illusion of object position and object attributes being the same. The following chapter develops specific semantics for this Responsive Positioning.



## 5 Responsive Positioning Semantics

On the basis of the concepts of structured space and semantic positioning, as presented in chapter 4, this chapter develops semantics for Responsive Positioning.

The context of Responsive Positioning is its application as a user interface technique for those kinds of user interfaces in which object attributes have to be both visualised and manipulated. Figure 40 shows a simple example of such a user interface which could be part of a larger document management application. This particular interface serves the purpose of determining and deciding which documents are published (e.g. by being available on a website) and which remain unpublished for the time being.



*Figure 40: A simple application using Responsive Positioning.*

A specific variant of semantic positioning which allows for the scenario depicted in the figure must meet two criteria. On the one hand, users must be able to use the user interface as a means of differential experience, which in this case means that they must be able to use the user interface in order to determine the current state of the documents. On the other hand, the same interface must allow the manipulation of the object attributes, which means that the status of the documents, whether or not they are published, must be manipulable by manipulating the perceivable objects within the user interface. Both requirements, as trivial as they sound, already have implications on the evaluation semantics.

## 5 Responsive Positioning Semantics

Possible interpretations provided in the previous chapter are not suitable for the task described here. Moving the rightmost icon into the region labelled ‘published’, for example, can only be interpreted as a change of the respective attribute. It cannot, like described as one possible interpretation before, be interpreted in a test context in which the movement would be interpreted as the answer of some student and thus trigger an evaluation function.

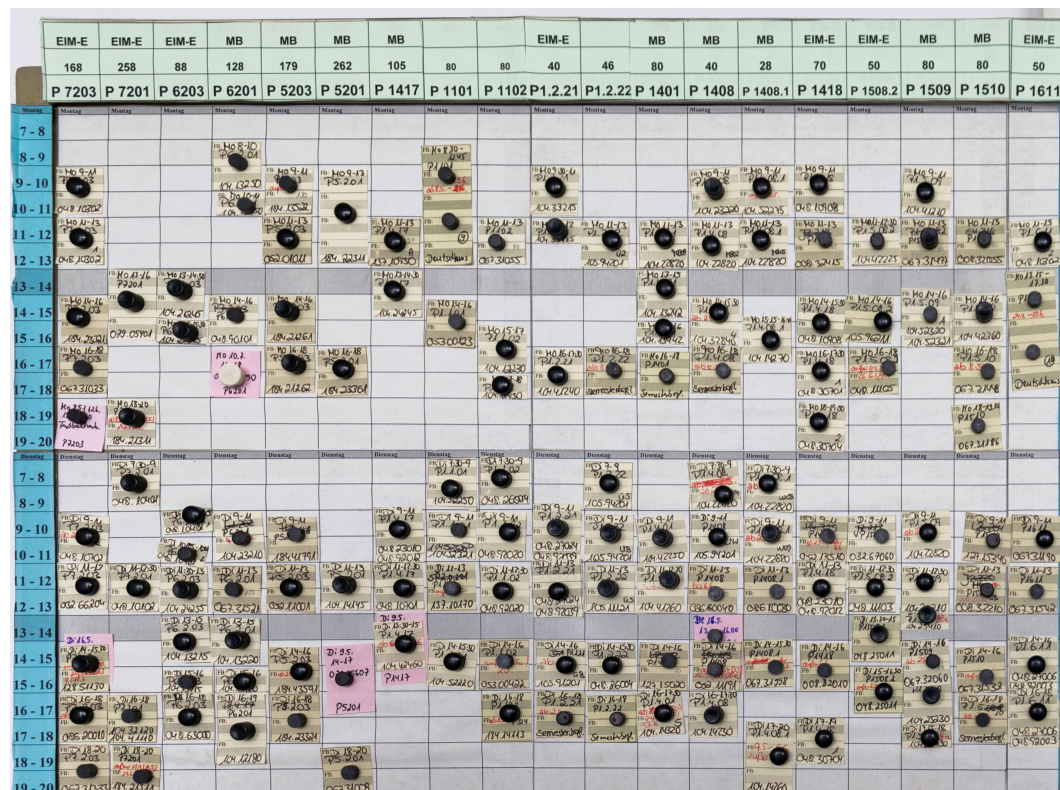


Figure 41: A detailed view of one of the boards in the room management office at Paderborn University

The example in figure 40 is comparable to the room management scenario I referred to a number of times. As shown in figure 41, every spatial position of one of the little cards which are magnetically attached to the wall-sized timetables directly corresponds to a room and a timeslot, thus to a room assignment. Should the room assignment of a lecture change, the respective card has to be moved accordingly. If, in turn, a card is moved to another position within the structure, its associated room assignment consequently is supposed to change as well. The room assignments are not only represented on these walls but are also kept in a



computer database. As the magnetically attached cards on the walls are of course an analogue technology, the synchronisation between what can be seen and manipulated on the wall and what is stored in the database has to be done manually, hence the timetables on the walls are a manually created visualisation with dynamic qualities, but they are not yet a user interface for the data stored in the database.

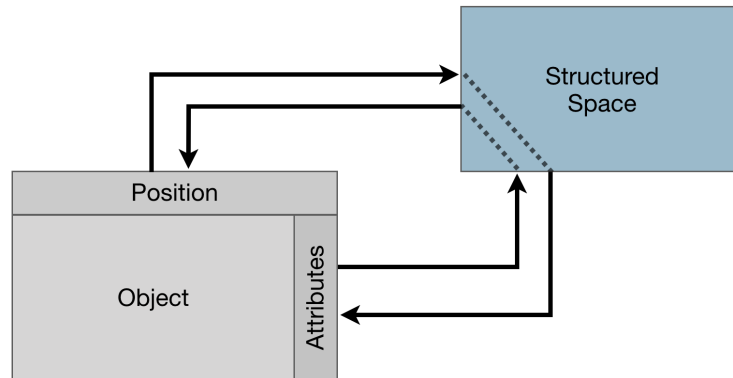
For the timetables to become a user interface for the room assignment database, there would have to be a continuous and automatic connection between the two. Imagine a technically enhanced version of the timetables in which every spatial manipulation of a card within the structure would directly be translated into a change in the database and every change in the database would immediately translate into a position change in the timetable (or the appearance or removal of a card in case of new or called off course)<sup>200</sup>. If this was the case, the people working in the room management office would likely not perceive the timetable on the wall and the room assignment database as two separate entities anymore. The immediateness of the change in one and the change in the other would for all practical purposes make the one stand for the other, or even make the one *be* the other.

The technical potential which can make such a coupling possible has been described in the context of *direct manipulation* in chapter 2. In direct manipulation interfaces, objects on the screen are repainted at such a high pace that users get the illusion of directly manipulable objects. Technically, when selecting a line of text and clicking on a “boldface” button, for example, the text is erased and repainted. As this repainting happens automatically and in most cases is carried out very rapidly, users have the illusion of the objects themselves being directly altered by the command. Responsive Positioning takes advantage of this potential of *responsiveness* in two stages. On the one hand, it assures the continuous display of an object while it is dragged around, thereby providing the impression of a stable, manipulable object, while on the other hand, and that is

---

200 When visiting the room management office I was told that the timetables on the walls are still used as they have found a good computer-based solution yet. The walls have the advantage of being very big on the one hand, thus allowing for a good overview of the situation, and allowing changes to be made quickly just by moving around cards without having to master any kind of computer-based user interface. Inconsistencies are avoided by following a strict order of operations in which changes are always carried out on the timetables first and are then synchronised to the database in a second step.

why it is called *Responsive Positioning*, it assures a responsive synchronisation between object positions and the attributes they correspond to.



*Figure 42: Responsive Positioning couples object attributes with object positions through their respective evaluations in relation to a structured space*

Figure 42 shows the basic idea behind Responsive Positioning in a schematic way. Responsive Positioning makes an object's position and its attributes correspond to each other by relating both to a structured space. In the sense of the examples given above, from a users point of view, this makes object positions and object attributes be the same. This chapter describes the semantics<sup>201</sup> for this correspondence. In chapter 6, I will extend on these semantics by complementing them with an application layer which has access to object attributes and which can thus invoke application functions according to attribute changes and can as well change these attributes.

---

<sup>201</sup> Great parts of the evaluation semantics described in this chapter are the product of intensive and productive discussions with Christoph Sens while supervising his master thesis (see Sens 2015) in which he described and prototypically implemented great parts of what I explain here. As not only time but also the work on this doctoral thesis progressed, and its conceptual foundation became more profound, my choice of words, as well as some semantic details, changed. Because of that, I decided not to include citations of Sens' work but to make references to those parts of his thesis, where his practical implementations and definitions are a proof of the technical and conceptual feasibility of the Responsive Positioning technique.

## 5.1 Correspondence in Responsive Positioning

In chapter 4 I defined correspondence in semantic positioning as a bi-directional evaluation function which translates between spatial positions and attributes. This definition explicitly did *not* define what happens in case of spatial object manipulations or attribute changes. As indicated, a variety of evaluations are possible. To meet the requirements of a user interface technique, in Responsive Positioning correspondence is defined in a more active fashion. It does not only describe the relation between attributes and position but it also describes concrete attribute assignments and object repositioning operations,. This means that in Responsive Positioning, moving an object immediately (thus responsively) changes its attributes, and changing its attributes immediately triggers a repositioning process<sup>202</sup>.

Definition: The **correspondence function  $c$**  defines a bi-directional mapping between object positions and object attributes in relation to a structured background. It consists of an attribution function  $a$  and a positioning function  $p$ .

When an object is placed at a certain position, the attribution function changes its attribute set by determining a number of attribute assignments:

Definition: The **attribution function  $a$**  calculates from a given position<sup>203</sup>  $p$  a set of attribute assignments  $A$  where  $A$  consists of assignments in the form of (*type*, *attribute*, *valueOrValues*) where *type* is either *set* or *unset* and *valueOrValues* either a single value, a set of values or a range of values.

Figure 43 shows a structured background defining two regions which refer to an attribute ‘species’ with values of ‘dog’ or ‘cat’, respectively. Objects A and B are supposed to be representations of actual animals which carry way more attributes than just the ‘species’ attribute such as, for example, their names or weights. The positions of A and B can be interpreted only in relation to the attributes the structure refers to. For object A, an attribution function can determine that this object must carry the ‘species’ attribute with a value of ‘cat’ while for object B

---

202 This, of course, means changing those attributes a structured background provides a structure for. Other attributes may also change but would have no perceivable effect on the object.

203 As explained in chapter 4, within the argumentation of this thesis, I assume that any object refers to only a single point in space, so width, height and shape of objects are not considered in their evaluation.

this function can determine that it must not have its ‘species’ attribute set to either ‘cat’ or ‘dog’.

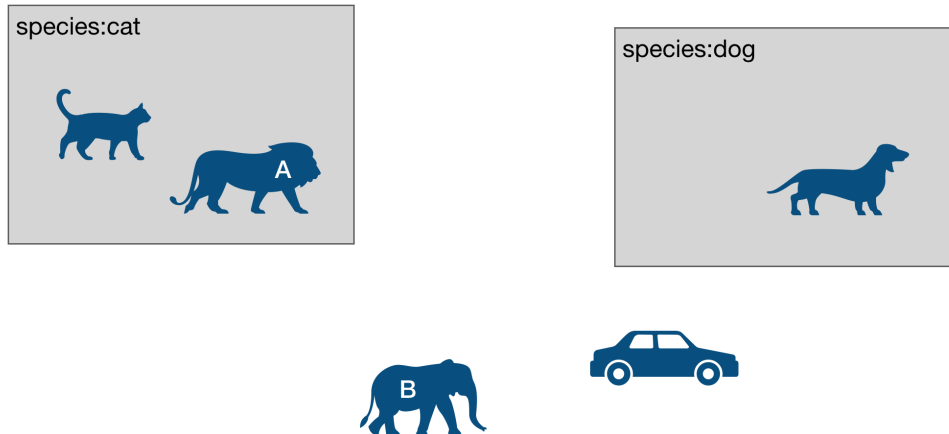


Figure 43: A structured background referring to only one attribute

All other attributes the objects might have are not affected by it and thus should not be influenced by the correspondence function. This is the reason why the attribution function does not determine attributes but rather determines *attribute assignments* or *attributions* which, in essence, are commands which have to be applied to the objects. They can also be understood as conditions the attribute set of an object must meet. Assume the car object is dragged into the ‘species:dog’ region. When, before the movement, the object had an attribute set like {brand:Mercedes; colour:dark blue; wheels:4} after that the attribute set would be {brand:Mercedes; colour:dark blue; wheels:4; species:dog} leaving all the attributes which are “superfluous” for the evaluation of this structured background untouched.

Attribute assignments can either *set* or *unset* an object attribute. If an assignment is *setting*, the attribute in the assignment has to be set to the specified value. If it is an *unsetting* assignment, the given value or range is a condition under which the attribute is unset. This means that an attribute is unset by an unsetting attribute assignment only if the object happens to carry the said attribute with the specified value or within the specified set or range of values.

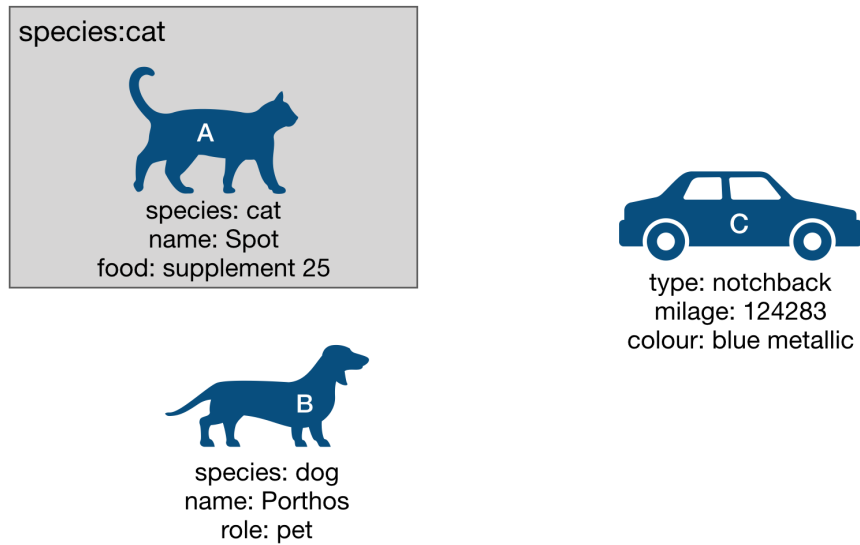


Figure 44: A background structure consisting of only one region, and three objects relating to it.

Figure 44 shows a number of objects and their attribute sets. The objects are positioned in relation to a structured background which contains a single region which refers to the ‘species’ attribute. When object C is moved into the region which is defined to refer to the value of ‘cat’, the attribution function calculates a set containing exactly one attribute assignment  $\{(set, species, cat)\}$  which, when applied, sets the species attribute of object C resulting in an attribute set of  $\{type: notchback; milage: 124283; colour: blue metallic; species: cat\}$ . When, in turn, object A is dragged out of the scope of the region, the attribution function calculates  $\{(unset, species, cat)\}$ , which means that in case the object has its ‘species’ attribute set to ‘cat’, which is the case here, it is unset. So, when this attribute assignment is applied, the object ends up with an attribute set of  $\{name: spot; food: supplement 25\}$ . The species attribute is gone. Notice that such a removal only happens under the given condition, so the ‘species’ attribute is only removed if its value is ‘cat’. When object B is moved to a position close to C the attribution function determines  $\{(unset, species, cat)\}$ , which in this case would be without consequence as the ‘species’ attribute of object B carries the value ‘dog’, so it is not unset by the calculated attribute assignment. The consequence of this kind of performing attribute assignments, both in the setting as well as in the unsetting sense, is that in Responsive Positioning one can only unset those attribute-value pairs one can also set with it. As the structure in figure 44 provides

no means of setting the ‘species’ attribute to ‘dog’, there is also no way of unsetting it using this structure.

When an attribution is setting, the attribute is set to a concrete value which is determined by the object’s relation to the structure. In case of a region, there is always only one value regardless of where within the region an object is positioned. In case of an axis though, as for object A in figure 45, the value for the ‘year of birth’ attribute, which in this case would be 1978, has to be calculated and set.

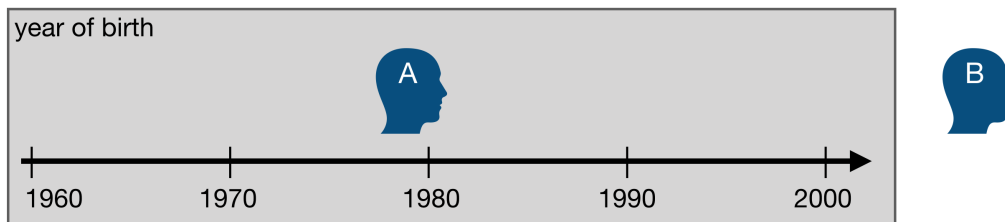


Figure 45: Setting and unsetting according to an axis

When determining the attribute assignments for object B, the attribution function determines an unsetting of the ‘year of birth’ attribute under the condition of its value being between 1960 and 2000 so, if it carried this object, and if its value were set to 1990 it would be unset, whereas, if its value were 1949, it would remain untouched. As before, the attribution function determines a condition under which the object would be unset. For an axis, like in figure 45, objects may keep the attribute when their value is outside the range of axis, so the condition, in this case, is the full range of the values of the axis. In case of a matrix or segmentation, the condition would be the list of distinct values.

While the attribution function determines a set of attribute assignments for a given position, the positioning function determines possible object positions from a given attribute set:

Definition: The **Positioning function**  $p$  determines for an attribute set  $S$  positioning information  $P$  where  $P$  is a tuple (*selector*, *Area*) where *selector* is either *must* or *mustnot* and *Area* a set of spatial positions<sup>204</sup>.

---

<sup>204</sup> For the argumentations in this chapter, I define the placement area as a set of points. This is a theoretical definition of  $P$  as the number of points within an area is, of course, infinite. In computer implementations, where space is quantised in pixels, a function could be implemented using a set of pixels, but such a solution has proven to be memory exhaustive.

Figure 44 can again serve as an example. The positioning function applied to object A calculates  $(must, R)$  where  $R$  is the set of points of the whole ‘species:cat’ region. This means that according to the background structure, object A has to be positioned within the boundaries of the region. Which of the points within  $R$  is chosen does not matter. For objects B and C, the positioning function calculates  $(mustnot, R)$  which means that those objects may be in almost every position of the structured space but explicitly must not be within the boundaries of the region.

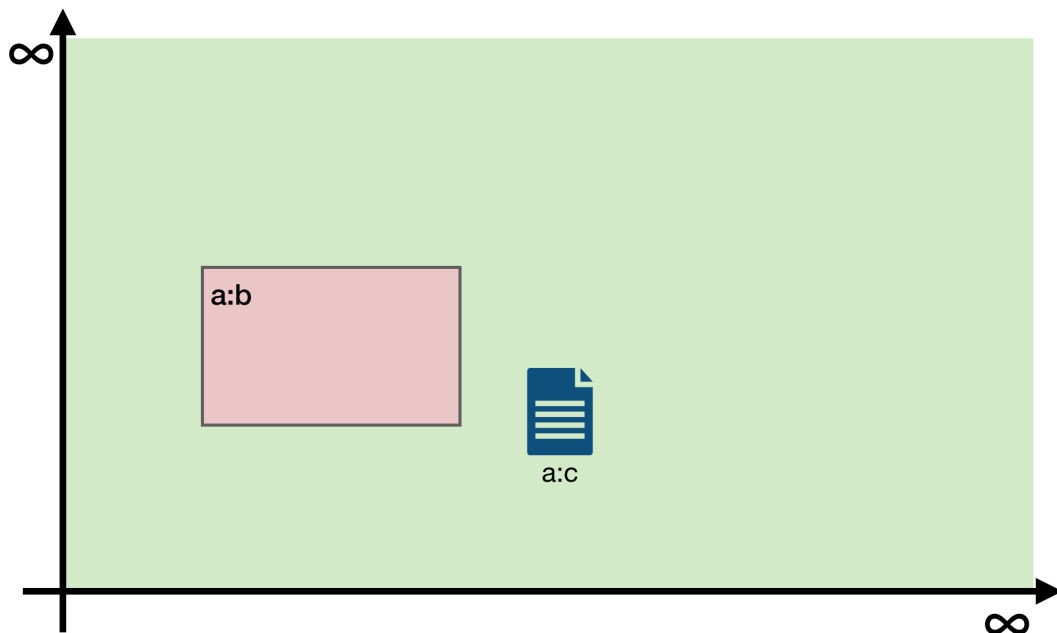


Figure 46: The result of determining the allowed positions for an object, which due to its attribute set may not be positioned within the scope of the region depicted.

The purpose of the positioning function is to find a position for every object relating to the structured background. It mainly does that by determining those positions where an object *may* be. Figure 46 shows an example where a position is to be determined for an object with certain tags. The structured background consists of only one region which refers to a tag of the object but requires it to have a different value. As a consequence of that, said object may be everywhere *except* within the region. The area where objects may be, here coloured green, thus is of an infinite size. While infinite areas are acceptable in theoretical

---

Sens (2015) described a polygon based solution in quite some detail (page 27) and implemented it in a prototype.

considerations and calculations, for computational reasons as well as for the sake of algorithmic simplicity, in such a case, instead of determining the area in which an object must be, I rather determine where it must not be. The resulting mustnot-area in figure 46 is marked in red. Doing so has the advantage of only having to consider non-infinite areas in the calculation, so whether or not a structure can place an object within its boundaries, the positioning function calculates a finite, bounded area in every case.

In the following, I specify the correspondence function for very simple background structures, namely those structures which contain exactly one region or exactly one axis. Later, these structures can be combined to more complex structured backgrounds.

### The Correspondence Function for a Region

The correspondence function for a region is very simple as regions do not have any differentiation within themselves. They correspond to a simple attribute-value pair. All objects within the scope of a region must have this attribute set to this value. Every object outside the scope must not carry the specified attribute with the specified value but may carry the same attribute with another value. Formally the correspondence function of a region can thus be specified as follows:

Given is a background structure containing only one region which refers to attribute  $k$  with the value  $v$  and which covers the spatial area  $S$ .

The correspondence function determines its values for an object  $O$ , where  $p$  is the object's spatial position, and  $A$  is the object's current attribute set.

$a(O): p \rightarrow \{(set, k, v)\}$  if  $p$  lies in  $S$ ,  $\{(unset, k, v)\}$  else

$p(O): A \rightarrow (must, S)$  if  $a$  contains  $(k, v)$ ,  $(mustnot, S)$  else

For objects A and B in figure 44, the correspondence functions of the structured background calculate the following values:

$a(A) = \{(set, species, cat)\}$

$p(A) = (must, P)$  where  $P$  is equivalent to the region labelled 'species:cat'

$a(B) = \{(unset, species, cat)\}$

$p(B) = (mustnot, P)$  where  $P$  is equivalent to the region labelled 'species:cat'



### The Correspondence Function for an Axis

Compared to the correspondence function of regions, the one for axes is slightly more complicated as for an axis there is not one single attribute value for a whole area, so positions and attribute values have to be calculated.

In case of an object being positioned outside the scope of an axis, it may either not carry the axis' attribute or have it set to a value outside the range of the axis. In this case, the result of the attribution function is an unsetting assignment for the attribute the axis refers to under the condition that the to-be-assigned object happens to have the specified value within the range of the values of the axis. Consequently in case of an object having the attribute the axis refers to either not set at all or set to a value outside the range of the axis, the result of the positioning function is a must-not area covering the whole scope of the axis. This assures that objects inevitably have to be placed outside of the scope of the axis.

In case of an object being positioned within the scope of an axis or having an attribute value being within the range of the axis, the correspondence functions must convert spatial coordinates into numerical values and vice versa. Sens (2015) describes the calculations behind these conversions in great detail<sup>205</sup>. For the argumentation here, I use a simplified model in which I neglect optical features like paddings, for example, so the axis I define here is graphically squeezed to the top of its bounding box<sup>206</sup>, which indicates its scope (see figure 47).

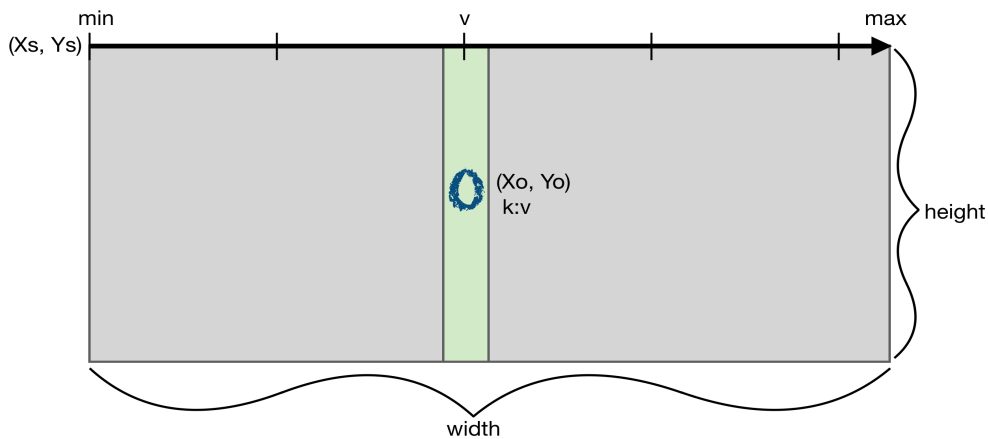


Figure 47: Illustration of the correspondence function of an axis

<sup>205</sup> See Sens (2015), pages 52-61

<sup>206</sup> In most computer systems, in contrast to the mathematical convention, the vertical coordinates go from top to bottom.

For such an axis I can determine the following formal definitions:

Given is a background structure consisting of only one horizontal axis<sup>207</sup> which refers to attribute  $k$  with a minimum value of  $min$  and a maximum value of  $max$  and which is covering the spatial area  $S$ . Let  $width$  be the structure's width and  $X_s$  the structure's  $x$  position in space.

The correspondence function determines its values for an object  $O$ , where  $(X_o, Y_o)$  is the object's spatial position,  $A$  is the object's attribute set and  $v$  is the object's value for attribute  $k$ .

$a(O): p \rightarrow \{(set, k, n)\}$  if  $p$  lies in  $S$ ,  
 $\{(unset, k, [min...max])\}$  else

$$\text{where } n = min + \frac{X_o - X_s}{width} \cdot (max - min)$$

$p(O): A \rightarrow (must, L)$  if  $A$  contains  $(k, v)$ ,  $(mustnot, S)$  else

$$\text{where } L \text{ is a vertical line of points in } S \text{ with } x = X_s + \frac{max - min}{width} \cdot v$$

According to the definition, the attribution function can have two different types of result.

1. Setting a value  $n$  has to be done if object  $O$  is located at a position within the boundaries of the axis. In this case, the attribution function  $a$  calculates the new value  $n$  in relation to the position of object  $O$ .
2. Unsetting its range  $[min...max]$  has to be done if object  $O$  is located at a position outside the boundaries of the axis. This results in attribute  $k$  being unset if object  $O$  happens to have the value of attribute  $k$  within the range between the  $min$  and  $max$  values.

Just like the attribution function, the positioning function also has two cases to cover, as either the attribute set of object  $O$  contains attribute  $k$  with value  $v$ , or it does not.

1. If  $O$  does have attribute  $k$  set to  $v$ , the positioning function returns a very slim line of points as a must-area<sup>208,209</sup>.

---

<sup>207</sup> A vertical axis can, of course, be defined similarly.

2. If  $k$  is not set to  $v$ , the result of the positioning function is similar to that of a region, meaning the whole scope of the axis is determined as the mustnot-area.

### Some notes on the actual placement of objects regarding the results of the positioning function

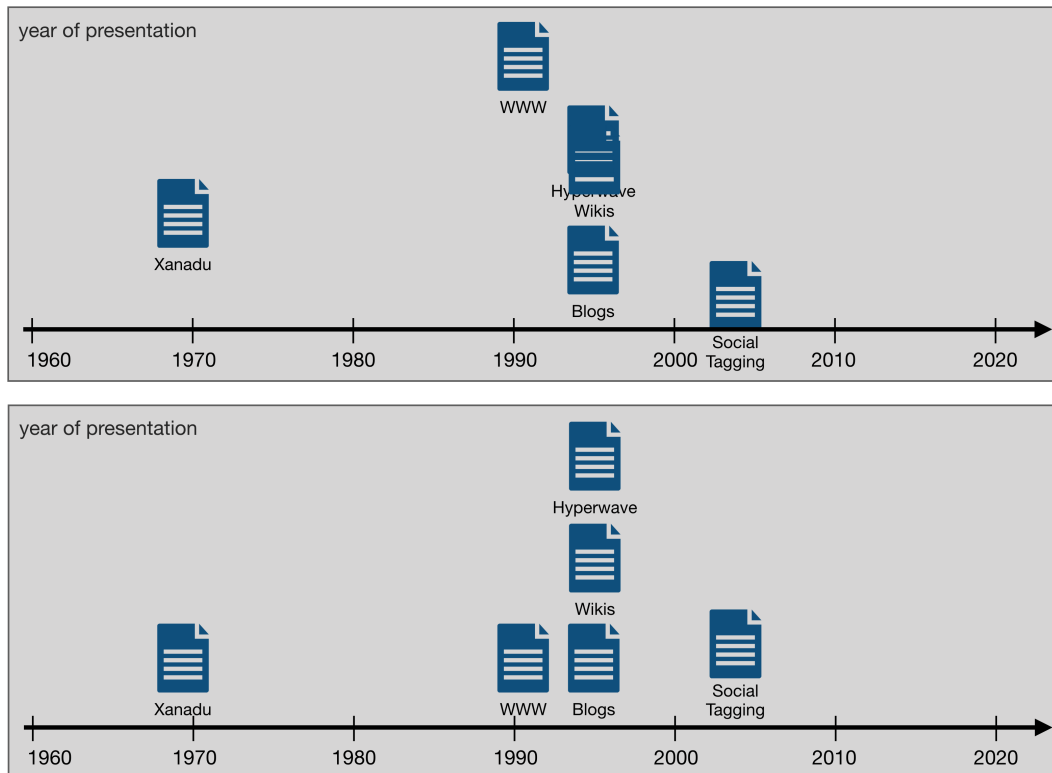


Figure 48: Two possible object placements for the same attribute set

The semantics described here generate information about in which area an object may be positioned or which attributes have to be set or possibly unset. It does neither directly determine an object position nor an attribute set but rather provides conditions for them. An object can be everywhere within the provided

208 Sens' definitions include axes in which the values are quantised instead of being continuous (on a timeline, for example, one normally does not specify values such as 1995,839). As this distinction, while being important in real-world scenarios, does not influence the argumentation I am putting forward here, I leave it with the continuous definition I made above.

209 In figure 47 this area is depicted uncharacteristically wide. For all practical and technical purposes, the resulting area would have a width of 1 pixel.

area, and it may have every attribute set which does not contradict the attribute assignments. The consequence of this is that for performing the actual object placements and attribute assignments, this information has to be ‘interpreted’. When executing the attribute assignments, this interpretation is very clear. Attributes just have to be set or unset depending on the condition provided by the attribution function.

In contrast to them, when positioning objects, there often are many possible choices. The positioning function does not directly calculate a position for an object, but rather provides the Responsive Positioning system with information which restricts the placement of an object to an area or, in the other case, provides specifications about where an object must not be positioned. Within these limits, every position is a valid one, so a Responsive Positioning function could just randomly choose a position. Doing so would be compatible with the semantics specified here, but the resulting object distributions would probably not look nice and defy typical conventions. Consider the two examples in figure 48. Both the upper and the lower example show “correct” placements according to the same attribute values. It is obvious that the first version has some issues which even diminish usability. Despite sufficient space, objects overlap, and the unnecessary distance between the axis and some objects impedes the possibility of quickly grasping the attribute value. The second version, on the other hand, is not only more usable and ergonomic, but also more aesthetically pleasing.

As indicated above, within its limits a Responsive Positioning system can “decide” where an object should be placed. If this “decision” kept track of the positions of the objects, it could prevent objects from overlapping which is maybe the worst problem in the unaesthetic example of figure 48. For further improvements, the placement functions themselves would have to be adapted. When creating the more aesthetically pleasing placement, I implicitly followed a rule which could be formulated as “When placing objects on a timeline, place them as low as possible while preventing any overlapping between the objects as well as between the objects and the visual part of the timeline itself.” This implies that some positions within the timeline should be preferred in relation to others.

All of this being true, within this thesis I stick with the rather simple specifications given so far as the calculation of positioning information in complex overlays of basic structures are complicated enough already. Still, for the

examples and illustrations, I allow myself to choose a good-looking interpretation instead of an unaesthetic and unergonomic one.

### Extending the correspondence functions towards more complex structures

The definition of correspondence functions for regions and axes are the foundation for the definition of a correspondence function for structured spaces with complex structures<sup>210</sup> which can be created by combining regions and axes. Moving object B in figure 49 to the indicated position, for example, results in an interpretation in which all the basic structures of the complex structure participate, so three assignments are calculated.

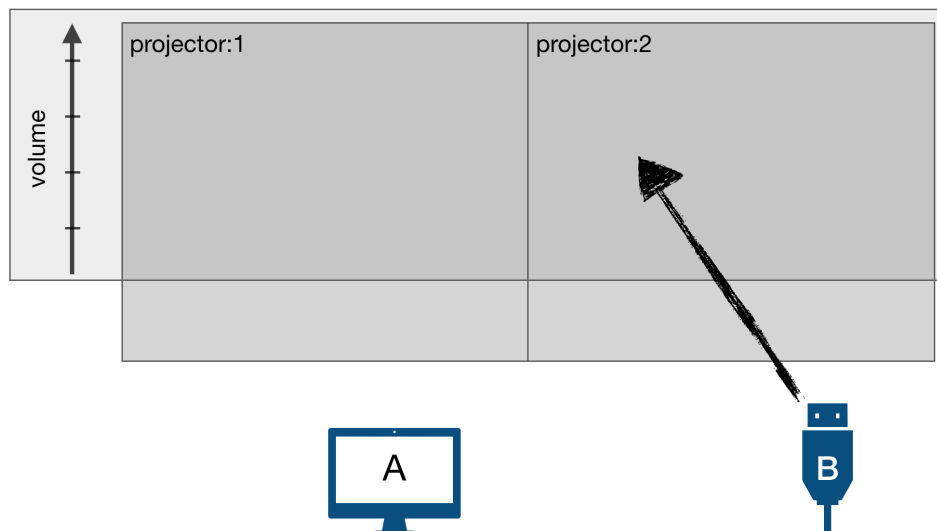


Figure 49: A simplified view of a media control application

First of all, there is the volume axis. It determines a new value of 2 for the ‘volume’ attribute. Even though the object does not lie within the ‘projector:1’ region, this region determines an assignment for the object, namely that its ‘projector’ attribute must not be 1. For the ‘projector:2’ region the situation is different, as object B does lie within its boundaries. Consequently, the determined

<sup>210</sup> Sens also defined correspondence and positioning functions for segmentations and matrices (see Sens 2015, pages 61-66) as well as for coordinate planes. In practical implementations where these structures are realised on the same level as are the basic structures, such definitions are necessary. For my argumentation here, I refer to the fact that segmentations and matrices are combinations of regions, and coordinate planes are combinations of axes.

attribution sets the ‘projector’ attribute to 2. An attribution function for such complex structures has to combine these assignments into a common set of attribute assignments.

Before being able to define correspondence functions for combined structures, I need to specify a number of criteria which a structured background must fulfil in order to be suitable for Responsive Positioning. Those criteria can then be used to check which kinds of complex structures are still suitable and which properties have to be checked in order to find out if they are.

### 5.2 Suitability Criteria

In this section, I describe suitability criteria for structured backgrounds. The correspondence function of every structured background used for Responsive Positioning must fulfil those criteria. If they were not fulfilled, the evaluation function would show unreliable characteristics where results would be ambiguous or contradicting. This would render them unusable for many kinds of semantic positioning in general and would surely make them useless for user interface purposes as one could never be sure about the state of objects or the result of object manipulations. It will be shown that the simple structures defined in the previous subchapter fulfil the criteria. In the next step, the criteria will be used to check if and under which circumstances the combination of basic structures results in a complex background structure which is suitable for Responsive Positioning.

#### 5.2.1 Independence

Independence means that when an object is processed by the correspondence functions of a structured background, the results of the correspondence must not depend on positions or attributes of any other object. If this were the case, the correspondence function would at least partially be an object-to-object relation, which implies that the background would not be a structured background in the sense of the definition given in chapter 4.

Independence is given for structures consisting of only one region or axis because of the way their correspondence functions are defined. They both have only one object as their input, and their definitions only refer to this object and properties of the structures, not to any other object.

### 5.2.2 Unambiguousness

Ambiguous structures would result in non-deterministic behaviour. When an object is placed in space, an ambiguous attribution function would provide two or more results to choose from. Such an outcome can be meaningful in those types of semantic positioning where the ambiguity might exactly be the differential experience the user wants to have. In case of Responsive Positioning, where an object should visualise its attribute state, thereby making it manipulable, such ambiguities would make the user interface itself non-deterministic and thereby unreliable. If, e.g. the placement of an object in relation to a structure could either set an attribute value to 5 or to 9, such a structure could not be used in order to set this attribute.

In formal terms I define:

*c is unambiguous if and only if a and p are unambiguous.*

- (1) *a* is unambiguous if and only if for every position *p* there is only one possible set of attribute assignments *A*.
- (2) *p* is unambiguous if and only if for every attribute set *S* there is only one possible positioning information *P*.

As for part (1) for structures consisting of only one region or axis, in both cases, the attribution function creates exactly one attribute assignment for every given position, so there cannot be a situation where the attribution function gives an “either or” result. The same arguments can be provided for (2), so in conclusion, there is no attribute set for which a structure consisting of a region or an axis could not determine a set of possible object positions in an unambiguous way. Notice that the argumentation here is about the set of possible object positions. It does not suggest that the result of the positioning function should be a single position. All positions within a resulting set are possible positions for an object, yet this set itself must not be ambiguous, so for the same input, the function must always calculate a single result set.

### 5.2.3 Absence of Conflict

If a correspondence function was not conflict-free, the attribution function would result in a set of assignments which contradict each other. This could mean that the set of assignments contains elements which assign different values to the same attribute or which define the same attribute-value-combination to be both set and unset.

Given this description, it might be difficult to distinguish absence of conflicts from unambiguousness. A simple example can explain the difference. Assume you have a device which you could use in order to determine whether or not it is raining. While if your device were ambiguous it would tell you “raining *or* not raining”, if it were not conflict-free though, it would tell you “raining *and* not raining” so the result would not be indifferent, as it would be in the case of an ambiguity but it would be contradicting in itself. In a user interface a conflict would mean that an object could be placed in a position for which it would not be possible to determine a clear attribute assignment.

In formal terms:

$a$  is conflict-free if and only if for every position  $p$ :

$A$  does not contain any  $(t1,a1,w1),(t2,a2,w2)$  with  $(t1=t2=set;a1=a2;w1\neq w2)$  or  $(t1=set;t2=unset;a1=a2;w1=w2 \text{ or } w1 \text{ in } w2)$ .

This is true for regions and axes because they both only create exactly one attribute assignment which, of course, could not be in conflict with itself.

### 5.2.4 Stability

A correspondence function needs to be stable. An example of a non-stable correspondence function would be a weird kind of axis with a correspondence function in which the attribution function assigns to an attribute the value of 10 minus the position on the axis while the positioning function translates the value to the position depicted on the axis. Figure 10 shows that an object placed at position 2 on such an axis would jump back and forth between the values of 2 and 8 indefinitely.



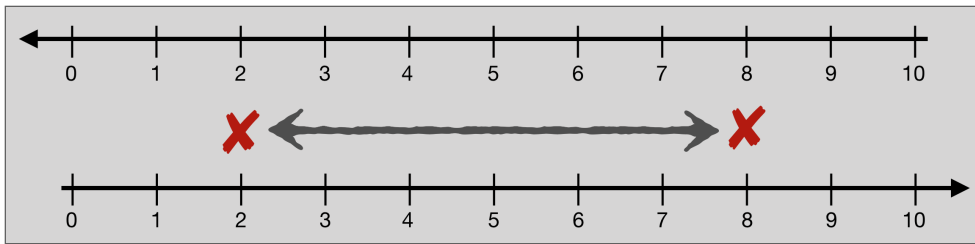


Figure 50: An unstable structure. Object attributes are determined in relation to the bottom axis whereas object positions are determined in relation to the top axis.

Stability basically means that the attribution function and the positioning function fit to each other. In a simplified definition, one could say that each of the subfunctions of the correspondence function must be the inverse function for the other. That being said, formally describing this “inversability” is a little more complicated than one might think as the attribution function and the positioning function are not directly compatible with each other. The attribution function determines a set of attributions for a given single position whereas the positioning function determines a set of possible positions for a single attribute set, so the result of one of these functions cannot directly be the input of the other function. Nevertheless, on the level describing the outcomes, stability can be defined by describing the necessary conditions for object placements and attribute changes. This description can then be the starting point for a more formal definition.

Assume for the following description that the criteria of independence, unambiguousness and absence of conflict have been confirmed. For a correspondence function to be stable, the following conditions must be met:

- When an attribute is changed and consequently an object moves according to the new attribute value, the position it ends up at must correspond to the attribute value that has been changed.
- When an object is moved into a position and attributes change according to this change in position, the resulting attribute set of the object must correspond to this position.

Generalizing these conditions so that they apply to all positions and all attribute sets, a correspondence function is stable, if

- (1) there is no position for which the attribution function assigns<sup>211</sup> attributes for which, when evaluated, the positioning function triggers a repositioning, and
- (2) there is no attribute set for which the positioning function determines a set of positions which contains at least one position for which the attribution function determines a set of attribute assignments which, when applied to the object, would result in a different attribute set than the initial one.

These two conditions have to be transformed into formal definitions. Doing so brings up the problem of the infinite number of attribute sets which would have to be considered. Objects in Responsive Positioning can carry any number of attributes which are not referred to by the current structured background, so they are not subject to the relation between the object and its background structure. This means there is an infinite number of attribute sets which is processed exactly the same way. In order to get rid of this problem and only have to check a single attribute set or to be able to check an attribute assignment against any attribute set, two additional definitions have to be made

### **Minimum attribute set function $a'$**

The attribution function  $a(p)$  calculates a set of attribute assignments  $A$ . There is an infinite number of concrete attribute sets which are compatible with such an attribute assignment. Let, for example,  $a(p)$  be  $\{\{unset, x, 9\}; \{set, y, 7\}\}$ . These assignments are fulfilled in the attribute sets  $\{x:20; y:7; z:nonsense\}$  as well as in  $\{y:7; animal:dog\}$ . To avoid having to care about all attribute sets which fulfil an assignment, a helper function *minimum attribute set  $a'$*  is defined as follows:

**Definition:** Let the *minimum attribute set  $a'(p)$*  be a function which determines the minimum attribute set which fulfils the attribute assignments  $a(p)$ .

In the above example, this minimum attribute set would be  $\{y:7\}$ .  $a'$  can be implemented by taking an empty attribute set to which all assignments are applied. This attribute set fulfils the assignment, as all setting assignments are represented as an attribute, and all unsetting assignments are not<sup>212</sup>. Using this

---

<sup>211</sup> This includes both setting and unsetting assignments.

<sup>212</sup> This assumes that the correspondence function is conflict-free, which is a prerequisite for stability in general so it can be assumed here.

minimum attribute set is sufficient in order to define stability, as only those attributes which take part in an attribute assignment are considered in the correspondence.

### **Contradicting attribute assignments**

For similar reasons, I define contradicting attribute assignments as following:

A set of attribute assignments  $A'$  is *contradicting* an attribute set  $A$  when there is an attribute in  $A$  which contradicts the assignment made in  $A'$ , or in other words,  $A'$  is contradicting  $A$  when  $A'$  applied to  $A$  changes  $A$ .

Example:

$\{(set, x, 90), (unset, y, 200)\}$  is not contradicting  $\{(x, 90), (y:300)\}$  because none of the assignments changes anything as the value of  $x$  already is 90 and the value of  $y$  does not have to be unset as its value is not 200.

$\{(set, x, 90), (unset, y, 300)\}$  is contradicting  $\{(x, 90), (y:300)\}$  because, in this case,  $y$  is 300 so after performing the assignments the object attribute set would be  $\{(x, 90)\}$ .

### **Formal definition of stability**

With  $a'$  and the definition of *contradiction* I can formally define stability based on the conditions made for (1) and (2):

The correspondence function  $c$  is stable if and only if it is independent, unambiguous and conflict-free and if

(1) for every position  $q$ :

$p(a'(q)) = (must, P)$  and  $q$  is within  $P$  or  $p(a'(q)) = (mustnot, P)$  and  $q$  is not within  $P$

and

(2) for every attribute set  $A$ :

For every position  $r$  in  $p(A)$ :  $a'(r)$  is not contradicting  $A$ .

### **Suitability for Responsive Positioning**

As all of the above criteria are necessary requirements for Responsive Positioning, I call a structured background which has a correspondence function which is independent, unambiguous, conflict-free, and stable *suitable for Responsive Positioning* (or *RP-suitable* for short).

In the next step, I define an algorithm for a complex correspondence function using which one can create complex structures by combining basic structures. The suitability criteria defined here are then used in order to determine under which circumstances basic structures can be combined to create more complex structured backgrounds.

### 5.3 Combined Structures

A combination of basic structures allows for graphical structures which correspond to more complex attribute combinations. Figure 51 shows an example of a background structure which consists of two simple structures, an axis named *X1* and a region named *A1*. These two structures are partially overlaid. A combined correspondence function for such a background structure works on the assumption that the results of the correspondence functions of the individual basic structures can be combined in a formal way. In case of an attribution function for figure 51, for object *A* the attribution function of *X1* calculates  $\{(set, x, 1)\}$  and the attribution function of *A1* calculates  $\{(unset, k, v)\}$  so consequently, the attribution function for the structure as a whole results in  $\{(set, x, 1); (unset, k, v)\}$ . Object *B* results in  $\{(set, x, 4); (set, k, v)\}$ , object *C* in  $\{(set, k, v); (unset, x, [0..10])\}$  and object *D* in  $\{(unset, k, v); (unset, x, [0..10])\}$ .

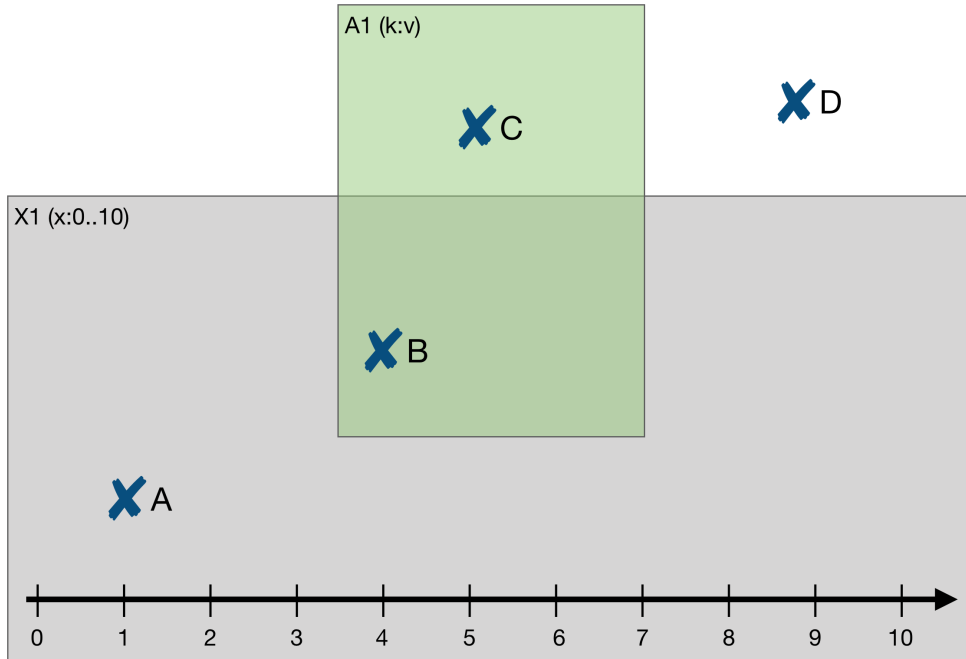


Figure 51: A structured background consisting of an axis and a region arranged in a way they partially overlap

When basic structures are combined, ambiguities may arise. Not every combination of basic structures which have by themselves shown to be suitable

for Responsive Positioning is suitable per se. Which combinations are possible does not only depend on the arrangement of basic structures but also on how the combined correspondence function is defined. For the combined correspondence function I provide here, I will show that, using the basic structures as building blocks, only one criterion has to be checked when building the background structures.

I define the following combined correspondence function  $C_c$  which consists of a combined attribution function  $C_a$  and a combined positioning function  $C_p$  as follows:

**ALGORITHM  $C_a$  (combined attribution) Input: Object  $O$**

```
1: let A be an empty set
2: for each basic structure G {
3:   AG = G.a(O)
4:   A = Union (A, AG)
5: }
6: return A
```

**ALGORITHM  $C_p$  (combined positioning) Input: Object  $O$**

```
10: let A,R and T be empty sets
11: for each basic structure G {
12:   PG = G.p(O)
13:   T = Union (T, TG)
14:   if PG is must-area
15:     If A is empty: A = PG else A = Intersection (A, PG)
16:   else
17:     R = Union (R, PG)
18: }
19: for every p in R{
20:   remove p from A
21: }
22: if A is not empty
23:   return ('must',A)
24: else
25:   return ('mustnot',T)
```

The combined attribution function  $C_a$  is very simple. In a loop, it just creates the union of all attribute assignments of the basic structures.

The combined positioning function  $C_p$  calculates the positioning area of a combination of basic structures in a two-step process. The loop in lines 11 to 18 goes through all basic structures and determines the positioning data for the given object according to each structure. The determined data is aggregated in variables T, A and R where in the end T contains all points covered by the structure, A contains all those points which are must-points of every structure that provides must-points and R contains all mustnot-points provided by any structure that provides mustnot-areas.

In line 19 to 21, all points which are in a mustnot-area are removed from the must set because if just one structure declares a position as mustnot, there is no way for any structure to overrule this.

In lines 22 to 25, the return value of the algorithm is determined. If there still are points in the must-area, this area is the result of the composite positioning function. In case there are no more such points, the background structure cannot place the object, so its whole surface becomes the mustnot-area which means the object for which a position is to be determined must be positioned outside the structure.

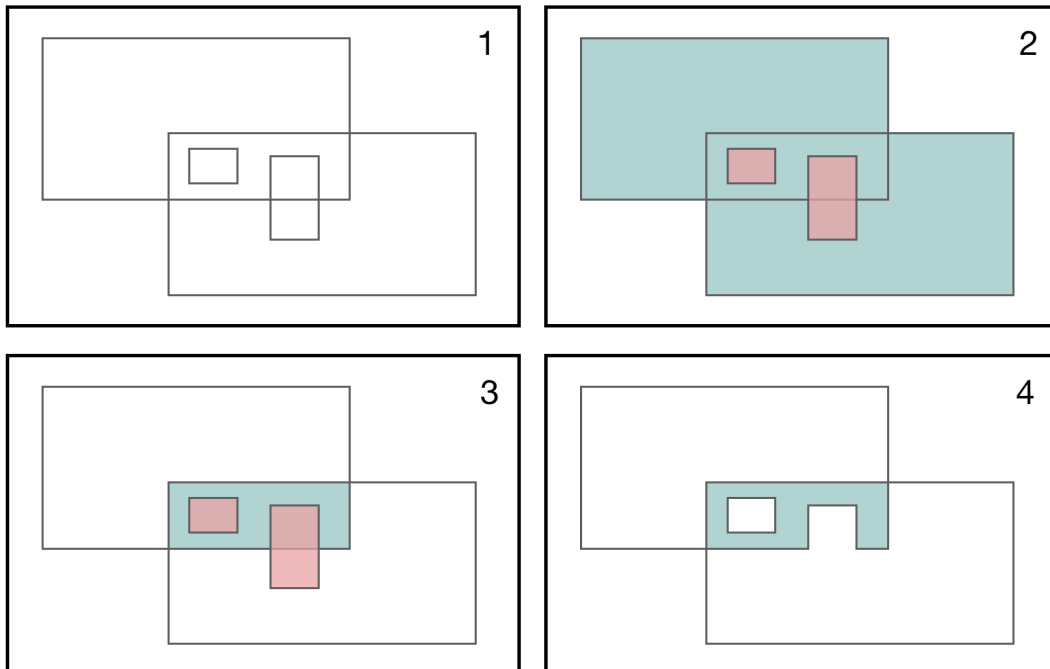


Figure 52: Graphical explanation of the composite positioning function

Figure 52 shows how the combined positioning function works. For a background structure of four regions which partially overlap (image 1), the positioning area is to be determined. For this, in a first step, the positioning areas of each individual basic structure are determined (see image 2 in which green represent must-areas while red represents mustnot-areas). The position of the object in relation to this combined structure must in every case be in all the given must-areas, so the given algorithm determines the intersections of the must-areas (image 3). As the position of the object must, of course, not be in any of the mustnot-areas, the algorithm subtracts all mustnot-areas from the must-areas. This effectively means the red areas are cut out of the remaining green areas resulting in a must-area with a number of holes in it (image 4). As there still is a must-area, it is the result of the composite positioning function.

Considering the suitability criteria specified in chapter 5.2 and given the combined correspondence function as defined above, it can now be checked, if and under which circumstances combined structures created as combinations of basic structures still are suitable for Responsive Positioning.

**Independence** can be confirmed for every combination of basic structures as any single basic structure is independent, and neither the combined positioning function nor the combined attribution function does induce any cross-references between objects.

**Unambiguousness** can also be confirmed for every combination of basic structures. Every basic structure creates one and only one set of attribute assignments and one and only one set of positioning information. The combined attribution function does not introduce any ambiguity as it processes those data into exactly one solution in a deterministic manner. There is no “either or” in the algorithm.

An **absence of conflict**, in contrast to the above criteria, is not predetermined. A combined structure is not conflict-free if two structures have a position in which they refer to the same attribute but assign a different value to it or if an attribute-value combination has to be equally set and unset. Such structures can easily be constructed.



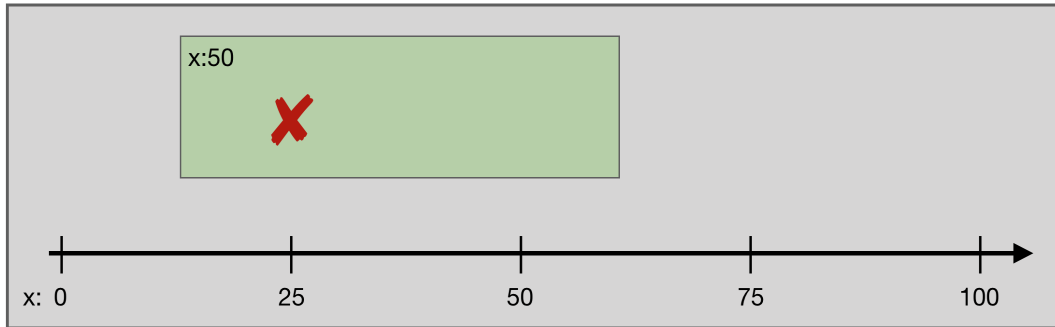


Figure 53: An overlay of 2 structures which refer to the same attribute leads to conflicting attribute assignments.

The example in figure 53 consists of an overlay of two structures. For one there is an axis which refers to attribute *x* and corresponds to values from 0 to 100. Secondly, there is a region which overlaps part of the scope of the axis. It also refers to attribute *x* with a value 50. This structure is not conflict-free. Positioning an object at the position marked by the red X, for example, results in an attribute assignment information of  $\{(set, x, 25); (set, x, 50)\}$ , so value *x* should be set to two different values at the same time which is a conflict within the attribute assignments.

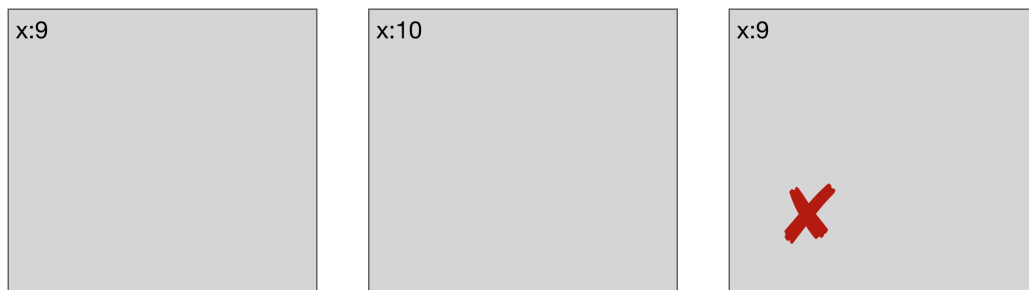


Figure 54: Two separated regions with the same key-value pair lead to conflicting attribute assignments.

Figure 54 shows a less obvious example of a combined structure which creates conflicting attributions even though the basic structures involved are not even overlapping. When an object is put in the place marked by the red X, all three regions evaluate this position, so the *x* value of 9 is set by the rightmost region and unset by the leftmost region, so the resulting attribution information is  $\{(unset, x, 9), (unset, x, 10), (set, x, 9)\}$  and thus contains a conflict<sup>213</sup>.

<sup>213</sup> There is a slightly more complicated correspondence function which allows such structures, but as I do not see any relevance for actual UI purposes I refrain from presenting them here.

### Stability

The stability criterion assures that, when an object is positioned, its attributes are set to values which correspond to this position and that, when an attribute value is set the object is moved to a position which corresponds to the newly set attribute.<sup>214</sup>

The following proof shows that stability is automatically given if the correspondence function of a background structure is independent, unambiguous and conflict-free, and if the basic structures it is created from are stable (which I have already shown in chapter 5.2).

**Proof:** In a combined structured space consisting of basic structures, which by themselves are suitable for Responsive Positioning and for which the combined correspondence function  $C_c$  is independent, unambiguous and conflict-free,  $C_c$  also is stable.

Assume for the purpose of contradiction that positioning an object in relation to a structured space consisting of basic structures which are suitable for Responsive Positioning and having a combined correspondence function  $C_c$  which is independent, unambiguous and conflict-free, does cause unstable behaviour.

When a correspondence function is unstable,

- (a) there is a point  $q$  for which the combined attribution function  $Ca$  results in an attribute set for which the combined positioning function  $Cp$  calculates positioning information which does not contain  $q$ , or
- (b) there is an attribute set  $A$  for which the combined positioning function  $Cp$  determines a set of positions in which there is at least one position for which the combined attribution function  $Ca$  determines an attribute assignment  $A'$  which contradicts  $A$ .

---

<sup>214</sup>See the argumentation in chapter 5.2.

I first attend to part (a):

By examining the algorithm  $C_p$ , I can make the following assessments.

- (1) Line 15 within the loop which is going through every basic structure of a composite structure (lines 11-18) assures that every point in a resulting must-area must have been in all must-areas of the basic structures which provide must-areas.
- (2) Lines 19-21 assure that all points which appear in a mustnot-area of any basic structure which provides must-not areas (collected in line 17) are removed from the potential must-area.

So for a point  $q$  not being in a resulting must-area, (1) it must not have been in the must area of at least one basic structure, or (2) it must have been in the mustnot-area of at least one basic structure.

There are two possibilities how this could have happened.

- (3) There is a basic structure for which, given position  $q$ , the attribution function has set an object attribute which according to the *same* structure's positioning function results in a must-area where  $q$  does not lie in (1) or results in a mustnot-area where  $q$  lies in (2). This means that one of the basic structures is by itself unstable.
- (4) There is a basic structure  $S1$  for which, given position  $q$ , the attribution function sets an object attribute  $a$  to value  $v$  which, when evaluated by the positioning function of *another* basic structure  $S2$  ( $S1 \neq S2$ ), results in a must-area where  $q$  does not lie in (1) or a mustnot-area where  $q$  does lie in (2). This means, for  $S2$  attribute  $a$  with value  $v$  does not correspond to position  $q$ .

According to the assumption, structure  $S2$  is stable. As for  $S2$  attribute  $a$  with value  $v$  does not correspond to position  $q$ , the attribution function of  $S2$ , given position  $q$ , consequently results in an attribution for attribute  $a$  which assigns a different value than  $v$  or which unsets  $a$  under a condition which includes  $v$ . In this case, the results of the attribution functions of  $S2$  and  $S1$  are in conflict. The attribution set of  $Ca$  thus contains a conflict, and thus the structure is not conflict-free.

The proof for (b) is of a similar structure:

In order for  $A'$  to contradict  $A$ , there must be at least one assignment in  $A'$  which

- (5) sets an attribute to a value which in  $A$  has not been set at all or which was set to a different value, or
- (6) unsets an attribute which is set in  $A$ .

The way the composite attribution function  $Ca$  is specified (lines 2 to 5), the determined set of assignments is the union of the attribute assignments of its basic structures, so if  $A'$  contradicts  $A$

- (7) there is a basic structure which positioning function for  $A$  results in a set of positions which includes one position which, according to the *same* structure's attribution function, results in an attribute assignment  $A'$  which contradicts  $A$ . This means that one of the basic structures is unstable, or
- (8) there is a basic structure which positioning function for  $A$  results in a set of positions which includes one position  $p$  which, according to *another* structure's attribution function, results in an attribute assignment  $A'$  which contradicts  $A$ . This would mean that those two basic structures must refer to the same position, yet provide contradicting attributions for it. In such a case, the results of the attribution function  $Ca$  for this position  $p$  contains a conflict, and thus the structured background is not conflict-free.

In (3) and (7) one of the structures would have to be unstable, and in (4) and (8), the background structure would not be conflict-free. Those properties are in contradiction to the assumption, which proves the initial claim, namely that if a composite structured space consists of basic structures which are suitable for Responsive Positioning and the combined correspondence function  $Cc$  is independent, unambiguous and conflict-free, it is also stable.

q.e.d.

Due to the definition of *Ca* and the properties of basic structures, independence and unambiguousness are always given (as shown above) and as, under the supposition of creating a combined structured space out of basic structures which are RP-suitable by themselves, stability can be reduced to independence, unambiguousness and absence of conflicts. The only problematic criterion thus is the absence of conflicts. Knowing this, I can check the *common structures* segmentation, matrix, and coordinate plane for suitability for Responsive Positioning (RP-suitability) by checking if they are conflict-free.

- For a segmentation, which is a lineup of non-overlapping regions referring to the same attribute, the only possibility of creating a conflict would be two of those regions referring to the same value (like in the example in figure 54). This is already ruled out in the definition of the structure (see chapter 4.2.1), so segmentations by design are RP-suitable.
- A matrix is an overlay of 2 segmentations which occupy the same area of space. It would be conflicting if both segmentations, meaning both rows and columns, referred to the same attribute. Again, this is ruled out in the definition of the structure (see chapter 4.2.1) making matrices RP-suitable.
- A coordinate plane is an overlay of two axes in orthogonal directions (typically one vertical, the other horizontal). Those axes could become conflicting if they both referred to the same attribute, which is, as one can assume, again ruled out by the definition of a coordinate plane (see chapter 4.2.1) so coordinate planes are also RP-suitable by design.

With these considerations, I have shown that not only the basic structures region and axis but also the common structures segmentation, matrix, and coordinate plane are RP-suitable by design. All other structures need to be individually checked for suitability by checking whether they are conflict-free.

## 5.4 Placeability

Having defined suitability criteria as well as correspondence functions for complex overlays of basic structures, in order to avoid any misconceptions, it must be clear that having a background structure which is suitable for Responsive Positioning does not mean that every object someone intends to relate to it can actually be sensibly evaluated by it. I call objects which can be positioned by a background structure *placeable* by that background structure. Other objects may be *unplaceable* even though the background structure may refer to their attributes.

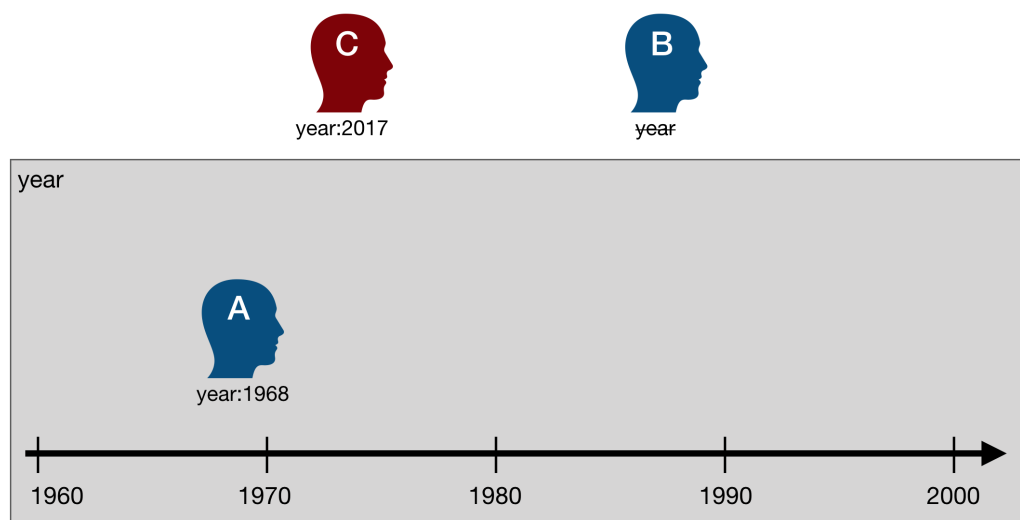


Figure 55: A simple axis structure and three objects relating to it, where two of those (B and C) are not placeable

The example in figure 55 shows a background structure consisting of an axis which refers to a year attribute in the range from 1960 to 2000. While object A is placeable by this structure, for different reasons objects B and C are not. While object B does not carry a year attribute at all, the attribute value of object C is 2017 which is outside of the range of the axis. The result in both cases is the same: The positioning function results in a mustnot-area for the axis structure, effectively pushing objects B and C to the side. While this is not problematic for object B, in many practical cases it would be for object C. Object B is an object which never carried the attributes relevant for this structure or which has been removed from it, so it does not carry any of its attributes anymore. Object C, on the other hand, is an object which should be placeable by this structure, yet the

structure happens to be poorly configured so that the object cannot be placed by it. Figure 56 shows the same phenomenon in a more complex structure which contains an axis which in this case refers to an ‘invented’ attribute. It is overlapped by two regions which refer to certain topics. While the Memex object would well fit into the ‘topic: Hypertext’ region, the structure cannot place the object there as the overlapping axis structure does not provide for its ‘invented’ value of 1945. This placeability problem is not an issue with Responsive Positioning but with the chosen background structure. It is suitable for Responsive Positioning, but it is not suitable for the task it is used for.

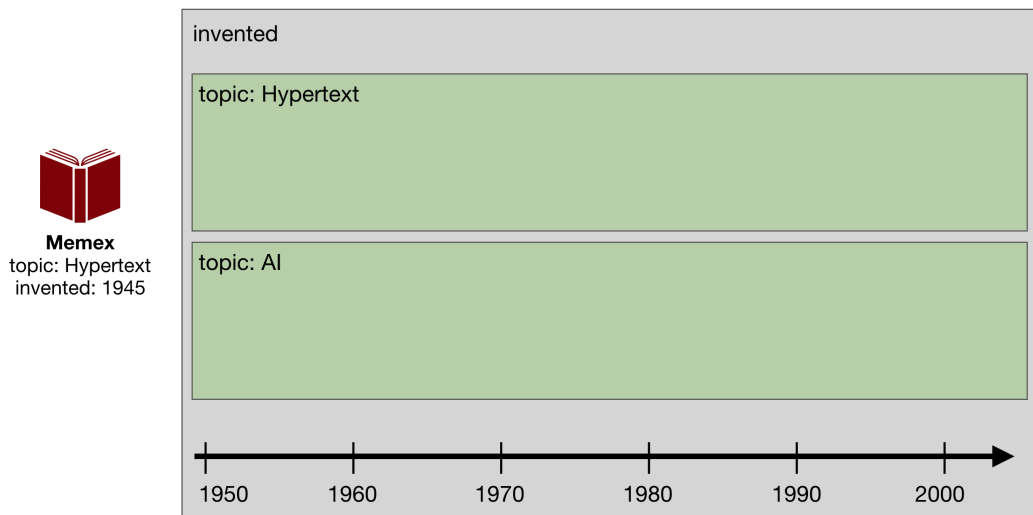


Figure 56: A composite structure consisting of an axis and two regions. The object "Memex" cannot be placed by it.

### 5.5 Structures, Sections and Contexts

In this chapter, I described and defined Responsive Positioning as positioning objects in relation to a structured background which provides a correspondence function using which object attributes correspond to object positions. This makes object attributes accessible to direct manipulation through their ability to be spatially repositioned.

While in the simple examples provided so far all relevant aspects of an object could be represented using a single structured background, for many practical applications this proves to be impossible. If all attribute manipulations of an object had to be made through a relation to only one single background structure, the possibilities would most likely be very restricted, as it is not likely that all attributes of an object which are of interest can be sensibly combined in a single spatial structure. A scientific document, for example, could be positioned along various timelines according to the publication date or according to the date or the subject of the document. It could also be positioned in regions according to its topics, in association with its author, according to language, etc. Some of those aspects can be combined by overlaying structures, but even this technique has its limits both spatially, having only two dimensions available, as well as semantically when not all attributes can sensibly be combined in one single structure.

As a consequence, it must be possible to evaluate the same objects in different contexts, which means in relation to different structured backgrounds. This could be achieved by switching between different structures and thereby between different evaluation contexts. I call this *switching* to another view. Each view displays the same set of objects in different places relating to different structures allowing the manipulations of other attributes, or even the same attributes in a different way.



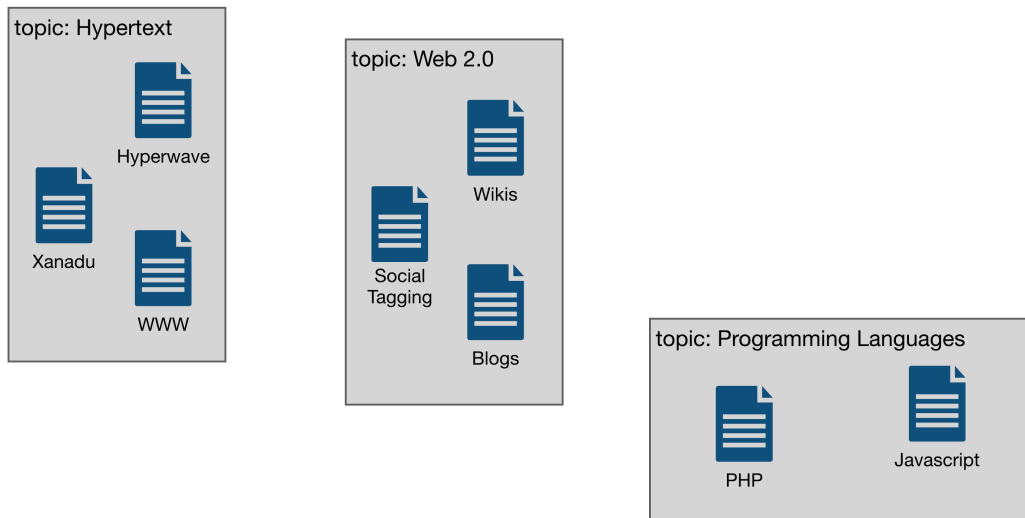


Figure 57: Documents positioned according to regions in a thematic view

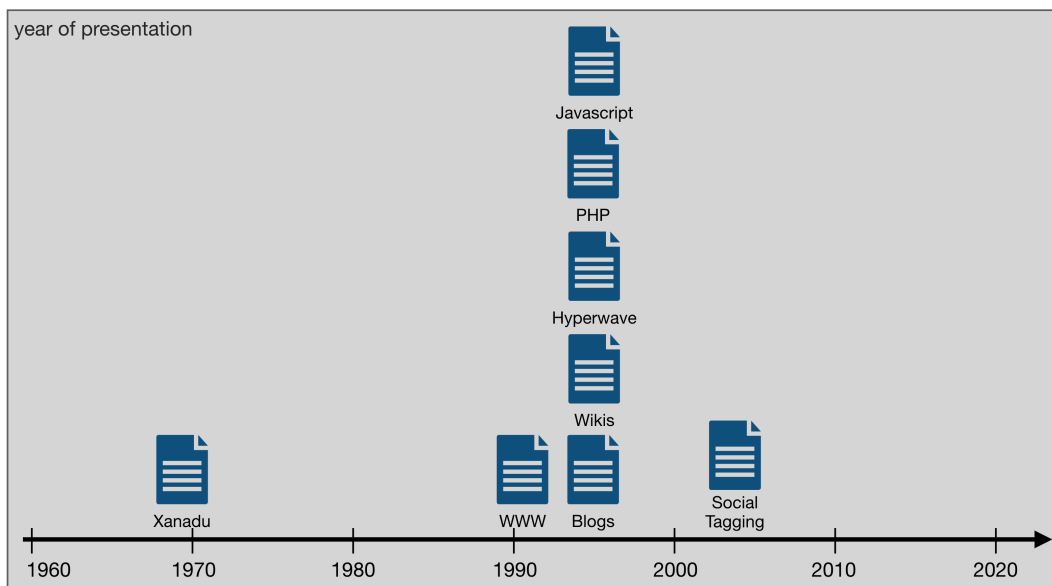


Figure 58: The same documents positioned according to a timeline view

Figures 57 and 58 show the same set of attributes. In figure 57 they are put into regions according to the topic they cover, while in figure 58 they are positioned on a timeline referring to the time of the invention of the respective technology. Both structures provide different insights into the attribute ensemble and allow for

different hypotheses regarding their distribution<sup>215</sup>. Such a switch of views can be considered a good solution when it is not necessary to have one object arrangement in sight while performing actions in relation to the other. This already indicates the biggest problem with a switch of views, which can best be made plausible by considering an example.














	topic:1	topic:2	topic:3	topic:4
complexity: easy				
complexity: medium				
complexity: advanced				
complexity: expert				

Figure 59: A structure in which documents are arranged according to topic and degree of complexity

Assume a teacher using a document organisation application which is based on Responsive Positioning. The teacher created a structure in which he can organise his material according to topic as well as according to complexity (see figure 59). As part of the administrative work for a course the teacher is organising, he wants to assign a number of his documents to different student groups. In order to do this, he switches to the view depicted in figure 60 where the same objects can be positioned according to three regions which represent three student groups. While it might be possible to work this way, the solution is far from ideal not only because the view has to be explicitly switched, which can be complicated enough,

<sup>215</sup> I came up with this object ensemble when creating a presentation about the possibilities of evaluable object positions. When creating the region based categorization, I thought I had chosen random elements, yet when I repositioned them according to the timeline the agglomeration of objects in the year 1995 actually came as a surprise and indeed caused questions on my part. Was the year 1995 special or am I somehow biased to 1995?

but especially because the arrangement in one view, which only exists for the purpose of bringing order into chaos, is lost in the other view. Bringing order into chaos first, yet not being able to refer to this ordered arrangement when trying to use the objects, cannot be an ideal solution.

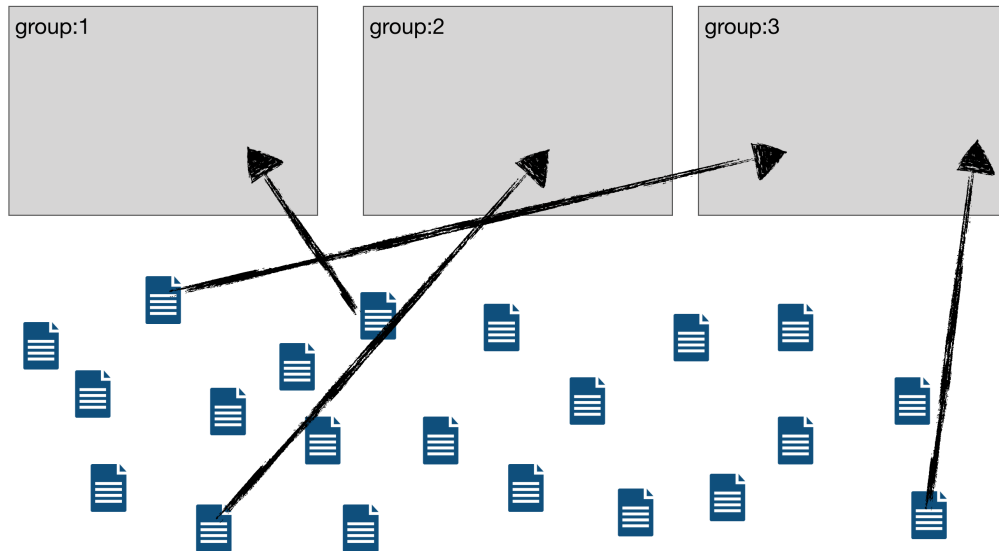


Figure 60: A structure in which the same documents can be assigned to student groups

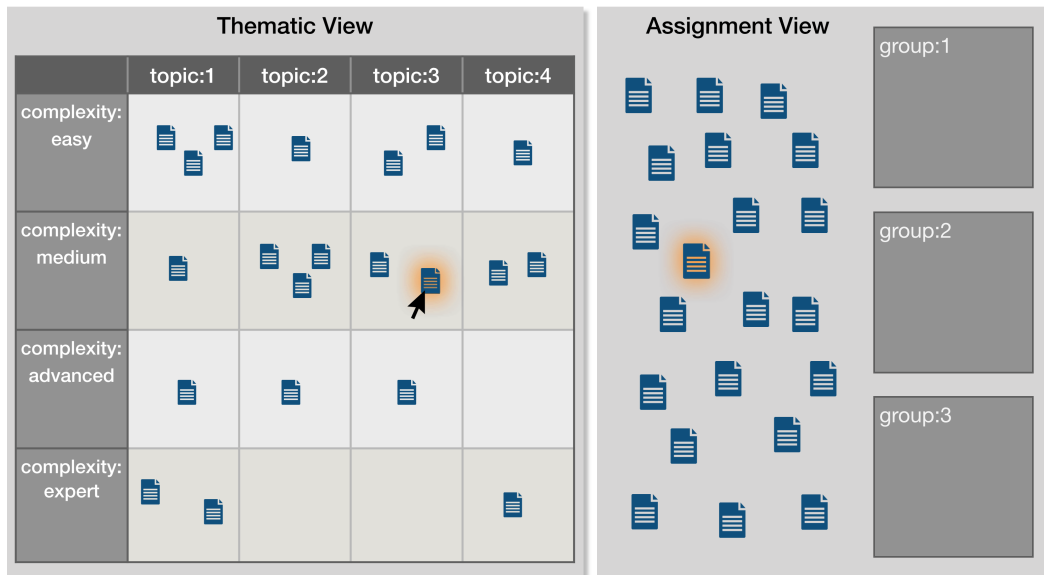


Figure 61: Both views being visible at the same time

Figure 61 shows a possible solution to this problem where both views are visible at the same time. When an object is selected in one view, it is highlighted in the other view. That way, the teacher can use his structured arrangement of documents to select the document he wants to use for the student group assignment. Such an arrangement of two or more views visible at the same time might be extremely useful in cases where it is necessary to see various attribute assignment of objects at the same time and where an object selected in one view has to be easily found in the other view.

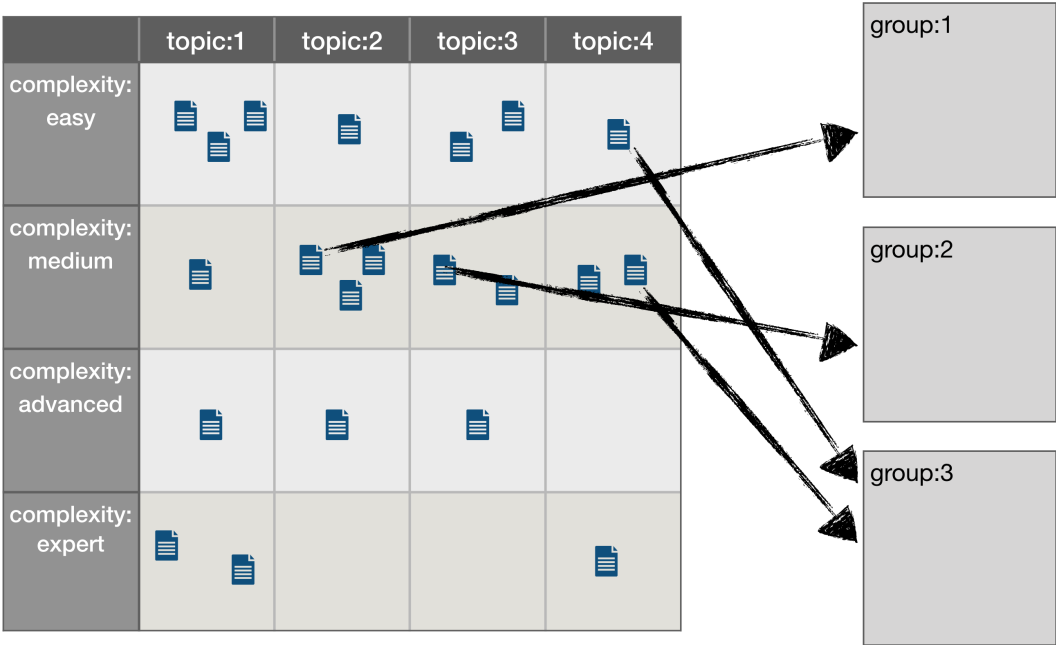


Figure 62: A solution with two evaluation contexts in one single view

A much simpler way of combining both the ordered thematic arrangement as well as allowing the assignment operation can be seen in figure 62. It shows the thematic matrix on the left and regions representing the groups on the right. In order to assign an object to a group, the teacher can just grab the document of his choice and drag it into the region of a specific group. Performing such an action based on the semantics which have been described so far, though, would deprive the documents of the thematic attributes they got in the matrix structure. Compare that to an office scenario: There are books on the bookshelf. You grab a selection of them and put them on your desk in order to work with them. On the bookshelf,

the books were ordered by author<sup>216</sup> while on the desk you want them to be ordered by topic. If the office in this scenario would “work” with the Responsive Positioning semantics defined here so far, the moment the books were removed from the bookshelf, they would lose any notion of their respective authors. In real life, this thankfully does not happen, so one can use the author attribute of a book, which one can find in the imprint or on the cover, in order to determine where to put it back.

In Responsive Positioning, the background provides the structure in relation to which objects are positioned. When examining figure 62, one can make out two both semantically as well as spatially distinct structures. Either an object relates to the structure on the left, so it is evaluated thematically in context of the matrix, or it relates to the structure on the right, so it is evaluated and thus positioned by the regions representing the group assignments<sup>217</sup>.

To make such structures possible, the semantics of Responsive Positioning are extended by the notions of *structures*, *sections* and *contexts*. A structured background, which can serve as a view on an object ensemble, provides one or more structures in relation to which object positions and object attributes are evaluated. Each structure can be understood as its own, distinct structured space. It provides its own evaluation context, which means that each structure provides correspondence for a certain set of attributes while other structures provide correspondence for other attribute sets. Each object at a time is evaluated in one evaluation context, thus in relation to one of the structures within the structured background. From a spatial point of view, having different structures within a structured background divides this structured background into several sections. Despite slight differences in focus, for all practical purposes, the terms structure and section can be used synonymously. Whenever an object is positioned in a

---

216 For the sake of the example, I ignore the fact that the order on the bookshelf likely is not a structured space as defined in this thesis, as books ordered by an author are most likely ordered on an object-to-object basis.

217 Notice that in figure 62 the sections are not explicitly made visible. There is an algorithm implemented in the WebArena prototype (see chapter 6.5) using which sections can automatically be determined. As in the following chapter, sections often play a semantic role within an application logic, and thus have to be explicitly defined and named within the Responsive Positioning views, I refrain from providing the algorithm here.

section of the structured background, it is positioned in relation to the structure which provides the evaluation context for that section.

When dragging an object from the matrix to one of the group regions, the object's context changes because the object is moved to another section. To allow for the concepts of structures, sections and contexts, the semantics of the correspondence functions provided in this chapter have to be extended in a way that both the attribution function as well as the positioning function always consider only those basic structures which correspond to the context of the to-be-positioned or to-be-attributed object. In figure 62 this means the moment the document is dragged to a group region and thereby to the grouping section of the structured background, which provides the grouping structure, its context changes from 'thematic' to 'grouping'. When the attribution function then evaluates the new position of the document, it only considers the three regions which are placed in the 'grouping' structure, leaving all thematic associations untouched. In order to "put back"<sup>218</sup> the object, its context is changed back to 'thematic'. The thereby triggered positioning function determines the object position only considering the basic structures which belong to the thematic structure which in this case only is the matrix. The object consequently moves back into its position within this matrix.

By allowing Responsive Positioning views to consist of several structures, I effectively allow more than one structured background to be simultaneously visible in one view by dividing the background into sections. This allows objects to be moved back and forth between different structured spaces. Each object at a time is assigned to only one of the structures as it can be in only one section at a time because the sections are spatially disjunct. This is a disadvantage for scenarios in which it would be necessary to be able to simultaneously perceive the object in several contexts. In scenarios where this is not necessary or not even wanted, it has the advantage of much easier means of manipulation and orientation.

Having more than one section in a single Responsive Positioning view taps new potentials for spatial structures as the association of an object to a structure is equivalent to its current evaluation context. If every object *has* a context at every time, this context becomes one of its *attributes*, so in the example above, moving

---

<sup>218</sup> This feature has a resemblance to the "put away" function of the Apple Lisa explained in chapter 3.

an object from the thematic section on the left into the group association section on the right changes its 'context' attribute from 'thematic' to 'assigned'. In order to bring the object back to its position in the other section, its 'context' attribute thus has to be changed back to 'thematic', which could, for example, happen automatically after the group work is over.

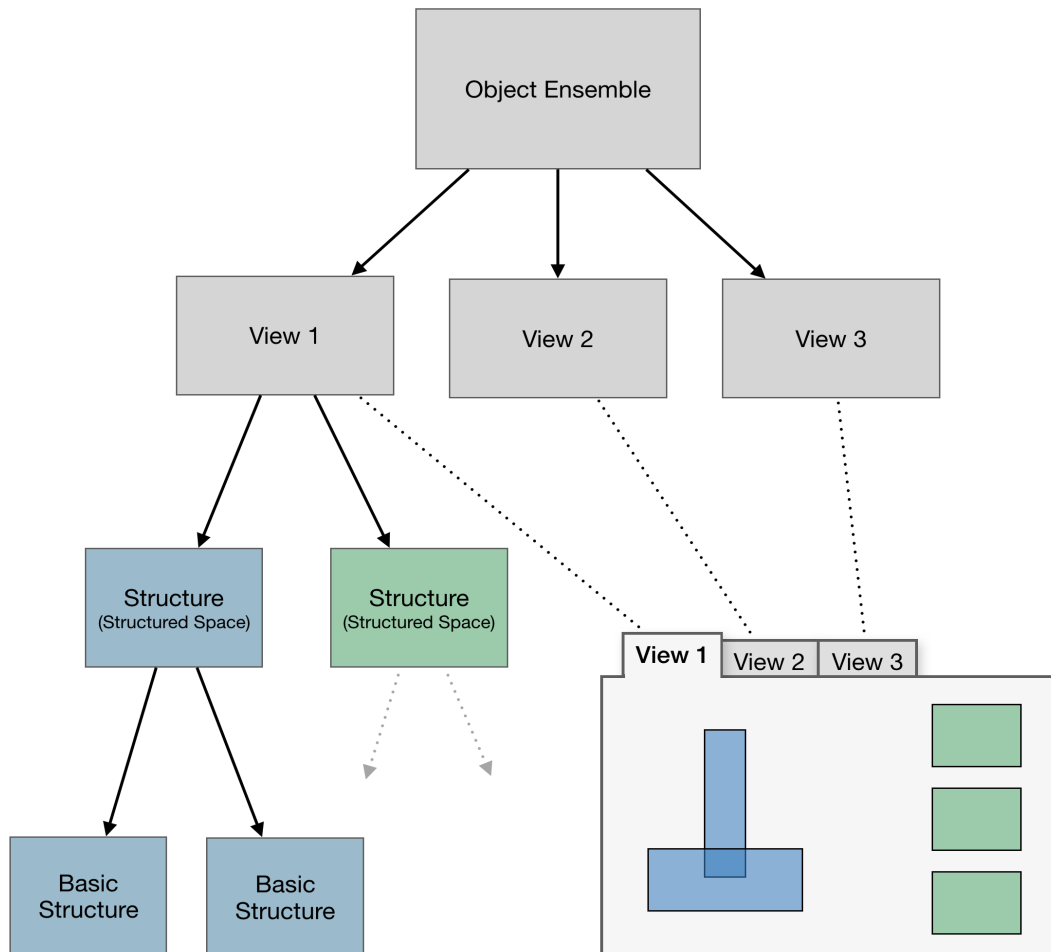


Figure 63: The relations between views, structures, and basic structures

Figure 63 schematically shows the relation I described here. For one object ensemble, there can be a number of different views which could either be visible simultaneously or, as indicated here, be accessible by switching between them. Each view consists of one or more sections or structures. Each structure, in a way, provides its own separated structured space which is integrated into the common structured background of the view. The active view in figure 63 consists of two

structures in two sections (red and green). Every structure itself consists of one or more basic structures.

This concludes the definition of semantics for “naked” Responsive Positioning. Using the semantics described in this chapter, object attributes can be both perceived as well as manipulated by their placement in relation to a structured background. It also explained the criteria according to which such structured backgrounds can be created, and defined concepts which allow an object ensemble to be subject to the evaluations of several structured backgrounds both at the same time as well as sequentially by switching the object’s contexts in relation to sections of a structured space. In chapter 6, I build on those concepts by complementing Responsive Positioning with an application layer, thereby making Responsive Positioning not only an interface using which one can assign and manipulate object attributes but one in which application functions are triggered by performing spatial object manipulations.



## 6 Applications and Architectures

In the previous chapter, I defined semantics for Responsive Positioning which allow the positioning technique to be used for manipulating object properties by manipulating their positions in relation to a structured background. This makes the positioning technique a proper user interface technique for applications in which such attribute manipulations are a central part.

This chapter goes a step further in describing how Responsive Positioning can be used in user interfaces whose primary purpose is not a pure attribute manipulation of already existing objects like documents or photos but, for example, the control of multimedia equipment or the configuration of a beverage dispenser. Both scenarios are not immediately associated with manipulable objects, so in order to use Responsive Positioning as their user interfaces, developers have to ask themselves which properties of the application can be made objects, which attributes they have, and what kind of structured background can be created in order to allow their necessary manipulations. In the case of the control of multimedia equipment, object manipulations have some effect beyond the mere manipulation of its attributes. The manipulations rather result in projectors turning on or off, or switching their input channels to a chosen source. For this and other applications, the Responsive Positioning logic described so far is complemented by an application layer which creates, removes and manipulates the objects in the user interface and reacts upon their attribute changes.

Figure 64 schematically shows how Responsive Positioning can be complemented by an application logic. An application using Responsive Positioning as its user interface, a *Responsive Positioning application* for short, can be divided into three layers. The functionality and semantics of the object and the structure layers have been described in chapter 5. These layers cover the presentational part of an application. This includes the Responsive Positioning semantics which provide the correspondence between object positions and object attributes. The application layer is decoupled from all aspects directly concerning the presentation. It can influence the user interface and gets information from it only through its access to the non-spatial object attributes.

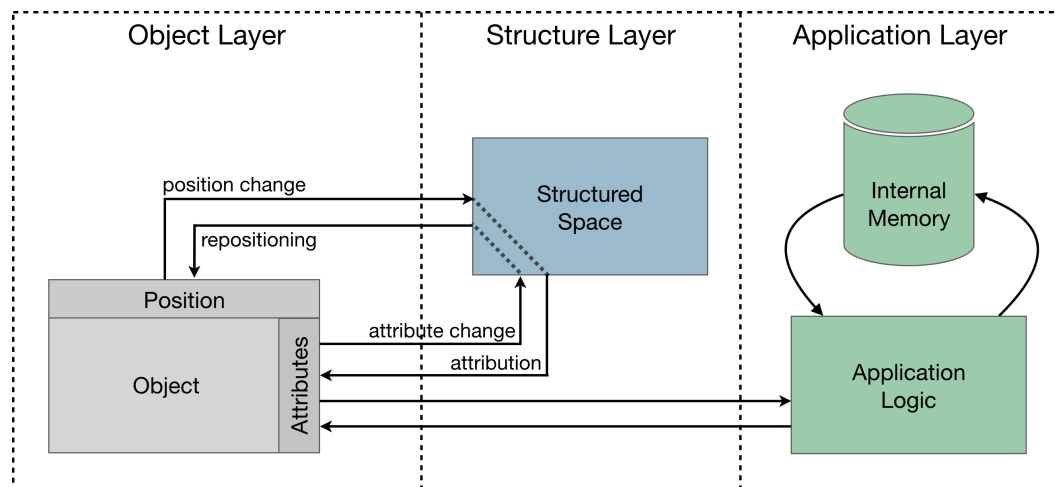


Figure 64: A schematic view of an application using Responsive Positioning.

The **object layer** consists of perceivable and manipulable objects. Spatial input, e.g. provided by the mouse pointer, and fast evaluations of this input allow the objects to be spatially selected and manipulated<sup>219</sup>. Objects have a position in space and carry an arbitrary number of object attributes which by themselves are neither perceivable nor manipulable.

The **structure layer** consists of one or more structured spaces which are created according to the criteria described in chapter 5. The structures provide correspondence between object attributes and object positions for the objects in the object layer. This correspondence makes object attributes both perceivable and manipulable through their positions in relation to the structured spaces provided by the structure layer.

The **application layer** consists of an application logic which has access to an internal memory, which means it can store and manipulate its own data structures. Apart from that, it can manipulate the objects on the object layer by deleting them, creating new ones, or manipulating the existing objects by manipulating their attributes. While the objects on the object layer and the structures on the structure layer both consist of visual elements, the application layer is non-visual and thus can only reveal itself through the manipulations it applies to the objects. It does not have access to spatial attributes of the objects so it cannot directly reposition

<sup>219</sup> See direct manipulation chapter 2.4.3.

an object but it can indirectly do so by changing objects attributes and thereby triggering object movements due to the Responsive Positioning evaluations.

The layers of a Responsive Positioning application are decoupled on two levels. First of all, the structure layer is decoupled from the object layer inasmuch as spatial structures can be created and defined independently of concrete objects which are to be structured by it. This aspect has already been covered in the previous chapters. Additionally, the application layer is decoupled from all spatial aspects of both the objects in the object layer as well as the structures in the structure layer. Though applications may create and delete objects in the object layer and, as will be explained later in this chapter, must also be able to configure aspects of the structured spaces in the structure layer, the application logic never aligns structures on the surface nor does it ever directly position objects in the user's action and perception space.

## 6.1 An Example of an Application Logic

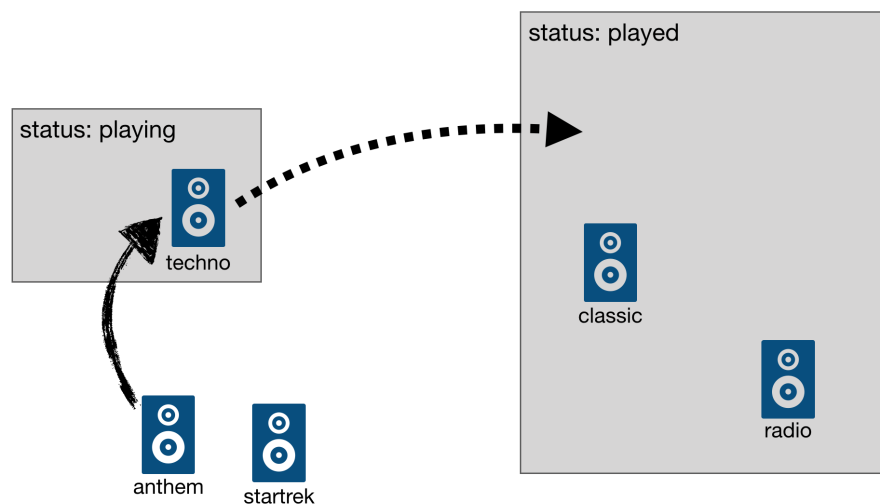


Figure 65: The user interface of an oversimplified music player.

To understand how the application logic refers to the objects on the object layer, assume the simple example of an oversimplified music player depicted in figure 65. On the structure level, it consists of a structured background composed of two regions. The smaller region on the left refers to a 'status' attribute set to 'playing' while the larger region on the right refers to the same 'status' attribute but is configured to refer to 'played'. In the current state of the application depicted in

the figure, there already are a number of song objects present in the object layer<sup>220</sup>. Each of them carries a ‘name’ attribute which is necessary to make them distinguishable from each other. In the course of the interaction with the application, the song objects gain ‘status’ attributes which initially are unset or are set to an explicit ‘undefined’ value.

The purpose of this application view is to allow its users to perceive which song is currently playing, which songs have already been played, and which songs have not. The view allows users to make the player application play a song of their liking by dragging any song object into the ‘status: playing’ region. The application logic internally contains a player component which can play mp3 files. A data structure within the internal memory of the application layer contains the path to an mp3 file for every object<sup>221</sup>. The internal memory further holds a variable which contains a reference to the currently playing song object, if there is any.

The song currently played can always be changed by dragging another song object to the ‘status: playing’ region. According to the Responsive Positioning semantics, dragging an object there changes its ‘status’ attribute to ‘playing’. Behind the scenes, the application logic reacts to such changes of the ‘status’ attributes of any object within the object layer. The interesting part of the application logic is therefore implemented like in the following listing<sup>222</sup>.

---

220 These objects must have been created by the application logic, yet as this part of the application logic is not interesting in terms of positioning aspects it is disregarded here.

221 It would also be possible to save this path as an object attribute, yet by doing this, application data which is not subject to user manipulation nor of user perception would be saved in the realm of user interface objects. With the goal of a separation of application logic and presentation in mind, such a situation should be avoided.

222 This source code in Javascript style resembles the way the application logic is implemented in the WebArena prototype (see chapter 6.5). Notice that this source code is simplified. In a real-world application, the application logic should, for example, check whether the object which is processed actually is a song object. I neglect such aspects here for the sake of code simplicity.

```

1 app.statusChanged=function(object, oldValue, newValue,
internal){
2     if (internal) return;
3     if (newValue=="playing"){
4         app.player.stop();
5         app.player.setPath(app.getPathFor(object));
6         app.player.start();
7         app.currentlyPlaying=object;
8         return;
9     }
10    else {
11        Object.setAttribute("status",oldValue);
12    }
13 }
14 app.player.onStop=function(){
15     if (app.currentlyPlaying)
16         app.currentlyPlaying.setAttribute("status","played");
17     app.currentlyPlaying = false;
18 }
19 app.player.onFinished = app.player.onStop;

```

The statusChanged function defined here receives four parameters: the object for which the status has changed, the old value of the status attribute, the newly assigned value of the status attribute and a flag which indicates whether the status change has come from within the application layer itself or is a result of the Responsive Positioning evaluation. This distinction is necessary as often, like in this example, the handling of an attribute change can trigger attribute changes of their own and thus trigger further evaluations. This could easily result in recursive evaluation calls. In order to avoid these and in order to generally be able not to react to internal attribute changes, it must be possible to distinguish between external attribute changes, originating from the user interface, and internal attribute changes, originating from the evaluation logic itself. In the music player example, only external manipulations are to be handled, so in line 2, the evaluation of an attribute change is immediately terminated in case it originated from within the application logic.

Figure 65 shows the application in a state before dragging the object named ‘anthem’ into the ‘status: playing’ region where an object ‘techno’ is currently

placed. In accordance with above evaluation logic, the internal `currentlyPlaying` variable of the application logic thus currently contains a reference to this ‘techno’ object. When the ‘anthem’ object is dragged into the ‘status: playing’ region, its ‘status’ attribute changes to ‘playing’ and the `statusChanged` function is called. In this function, the internal player component, which is still playing the mp3 file associated to the ‘techno’ object, is stopped (line 4). This triggers the `onStop` function of the internal player (lines 14 to 18). As only one song can play at a time, and thus only one song object at a time can have the ‘status’ attribute set to ‘playing’, in line 16 the ‘status’ attribute of the previously playing ‘techno’ object is changed from ‘playing’ to ‘played’. On the user interface this results in the previously played song object, the ‘techno’ object, moving itself to the ‘status: played’ region. After that, the internal player is configured to play the mp3 resource saved for object ‘anthem’ (line 5) and is subsequently started (line 6). Additionally, a reference to the object is saved in the internal `currentlyPlaying` variable (line 7). At this point, the evaluation of the attribute change ends. The internal sound player plays the mp3 file associated with the ‘anthem’ object, and the previously playing ‘techno’ object has been moved according to its new status.

The remaining lines of the function, lines 10 to 12, cover ‘status’ changes apart from those to ‘playing’. These occur when a song object is dragged to neutral ground or when it is dragged into the ‘status: played’ region without ever having been played at all. These actions are supposed to have no sensible meaning, so in these cases, the application logic discards the changes by immediately changing the ‘status’ attribute back to its previous value. On the user interface level, this means that, for example, whenever an object is dragged to the neutral ground, it immediately snaps back to the region it was dragged from<sup>223</sup>.

The source code of an application logic, like the one in the given example, is completely independent of the actual graphical rendering of the application objects and the graphical structure these objects relate to. It does not refer in any way to the spatial structure at all. The application logic could without any changes be controlled using a generic command line interface. The interaction with such an interface could look like this:

---

<sup>223</sup> Later in this chapter, I will describe manipulation restrictions which would have prevented the dragging operation in the first place. As the application logic should always result in well-defined object states regardless of the user interface used, the “resets” indicated above should be implemented nonetheless.

```
>list
Id:1 name: classic  status: played
Id:2 name: startrek status: undefined
Id:3 name: techno   status: playing
Id:4 name: radio    status: played
Id:5 name: anthem   status: undefined

>set status 5 playing
attribute "status" of object 5 now is "playing"

>list
Id:1 name: classic  status: played
Id:2 name: startrek status: undefined
Id:3 name: techno   status: played
Id:4 name: radio    status: played
Id:5 name: anthem   status: playing
```

## 6.2 Responsive Positioning and the Model-View-Controller Paradigm

As shown above, in Responsive Positioning the application logic is separated from its rendering in the user interface. Efforts to separate the business logic of an application from those parts responsible for the presentation of content and user interface already came up in the early days of graphical user interfaces and found one of their first systematic descriptions in the Model-View-Controller (MVC) concept<sup>224,225</sup>.

Tracing the MVC concept back to its roots shows that it has been reinterpreted frequently ever since. What all the interpretations since the late 1970s have in common is said separation of the user interface from the business logic, which in MVC is called a *model*. Not mixing those two has quite a number of advantages, including better testability especially for the application logic, the ability to change the user interface while keeping the underlying application logic and being able to use different user interfaces for the same application. As Responsive

---

224 Model-View-Controller is sometimes called a paradigm, on other occasions it is called a pattern, and at times it is even referred to as a metaphor. I consider “concept” to be neutral enough to cover all these interpretations.

225 Another prominent example of a concept for such a separation is the Seeheim model (see Green 1985). Without extending my examinations to this model, it can be said that the Seeheim model, in contrast to MVC, was targeted at text based “dialogue design”. It describes three layers and thereby separates input capturing and output rendering (the “presentation”) from the a command interpreter (the “dialogue control”) and the application (the “Application Interface Model”). In contrast to MVC, it does not support the flexibility of several views and several controllers being connected to a single model.

Positioning shares the general idea of separating the user interface from the application logic, determining whether Responsive Positioning user interfaces are compatible<sup>226</sup> with the MVC concept proves to be very interesting. On the one hand, a compatibility would prove that Responsive Positioning could easily be integrated into user interface frameworks based on the concept. On the other hand, and this might be even more interesting, an incompatibility between MVC and Responsive Positioning could be an indicator pointing towards the innovative aspects of Responsive Positioning.

### **Case 1: The Original Concept by Trygve Reenskaug**

The Model-View-Controller paradigm has originally been developed by Trygve Reenskaug<sup>227</sup> at Xerox PARC for the Smalltalk 76 programming environment in the late 1970s. Reenskaug distinguished between model, view, controller, and editor. After briefly describing a model as “an active representation of an abstraction in the form of data in a computing system”<sup>228</sup> he defined view and controller as follows:

“To any given Model there is attached one or more Views, each View being capable of showing one or more pictorial representations of the Model on the screen and on hardcopy. A View is also able to perform such operations upon the Model that is reasonably associated with that View.”<sup>229</sup>

“A controller is the link between a user and the system. It provides the user with input by arranging for relevant views to present themselves in appropriate places on the screen. It provides means for user output by presenting the user with menus or other means of giving commands and data. The controller receives such user output,

---

226 By being compatible I understand an MVC variation conceptually providing structures in which the Responsive Positioning model shown in figure 64 can find its place. I would not consider an MVC concept to be compatible with Responsive Positioning if it was only possible to somehow “squeeze in” the Responsive Positioning semantics.

227 See Reenskaug (1979a) and Reenskaug (1979b). Between those two memos, his terminology changed slightly as far as what is called the editor in 1979a is called controller in 1979b, as in this extension to his concepts he defined a specialised role for editors. In the presentation given here, I refer to both memos using the terminology of 1979b.

228 See Reenskaug (1979a)

229 Ibid.



translates it into the appropriate messages and passes these messages on to one or more of the views.”<sup>230</sup>

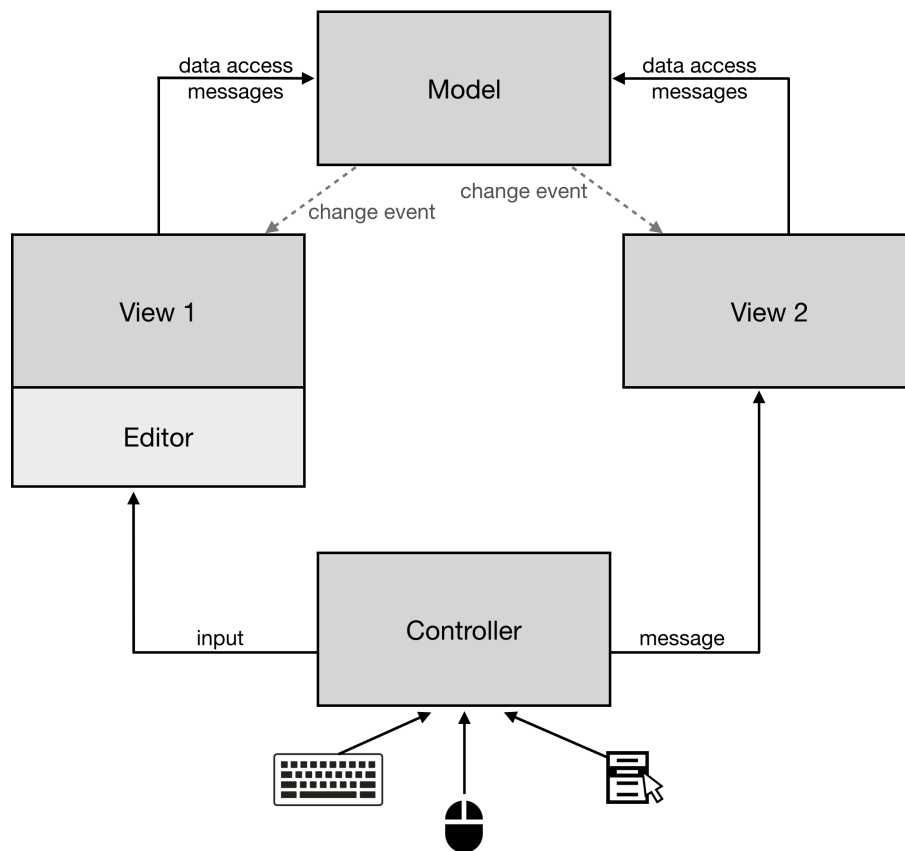
Reenskaug understands the controller and its views as a tool whose purpose it is to perceive and manipulate the model. His concepts of view and controller are quite unique. Views provide the user with different renderings of the model. In Reenskaug (1979a), where he describes a planning system, a model can be viewed as a network diagram, a Gantt diagram, a resource diagram, or as a list of elements. A controller knows these views and displays them as well as additional user interface elements such as menus or text input fields by arranging them on the screen. The controller accepts user input and either handles it itself or relays it to one or more views by sending messages to them<sup>231</sup>. A message to a view could either just change its rendering, e.g. when a zoom level is changed, or it could mean a modification of its content which the view translates into operations to be performed on the model. Model changes are broadcast to all views as well as to the editor which then update their renderings or behaviours as necessary.

In Reenskaug’s MVC interpretation, which is schematically depicted in figure 66, there is a *single* controller which controls *its* views, meaning the views *are owned by* the controller. Reenskaug also specifies editors<sup>232</sup>, unfortunately without too much further explanation, as special kinds of controllers which are specific for a certain view. The purpose of such an editor is to enable the user to not only send messages to a view but to directly change its content. Reenskaug explains that editors have to be provided by the views as they are closely associated with “the metaphors of the connected view”. Editors are “spliced into the path between the controller and its view” as long as the content of the view is edited. Without mentioning the concept, I suppose that what Reenskaug describes here essentially is the possibility of direct manipulation where instead of updating a view through a command, the displayed content itself is manipulated.

230 See Reenskaug (1979b). Reenskaug defines input and output from the user’s point of view. When he speaks about “means for user output” he speaks about what today mostly would be called “means of input” as most computer scientists describe matters from the perspective of the machine.

231 “Sending messages” is Smalltalk terminology, where messages were sent back and forth between objects. More on this concept can be found, e.g. in Goldberg&Robson (1983). In modern object programming terminology, one would speak about function calls and passing arguments instead.

232 See Reenskaug (1979b)



*Figure 66: Schematics of Reenskaug's MVC paradigm according to his descriptions in Reenskaug (1979b)*

Reenskaug's MVC model can quite well be applied to Responsive Positioning applications especially because of his interpretation of views as views on the model which can be manipulated. In Responsive Positioning, the arrangement of objects on the screen indeed is a view on the application objects which can be interpreted as a model. The specific configuration, which describes how the objects appear, is provided through the structured background. Each view visualises the same objects (the same model) in the form of graphical objects in relation to different structured backgrounds. In this sense, each structured background constitutes a different view, just like in the examples Reenskaug explains in which the same model is interpreted as a graph in one view and as a Gantt diagram in another. Each view provides its own editor. Given the fact that the objects in a Responsive Positioning application are themselves manipulable, view and editor in this regard are very closely related. Whenever there is more

than one Responsive Positioning view available, a Reenskaug-style controller is needed in order to select the appropriate views and to provide a mechanism which allows users to switch between them.

## Case 2: Model-View-Controller in Smalltalk 80

While Reenskaug's work, unfortunately, remained in the state of internal technical notes at Xerox PARC and have only quite recently been published on Reenskaug's website<sup>233</sup>, a variation of the Model-View-Controller concept found its way into the Smalltalk 80 environment. It was presented almost a decade later by Krasner&Pope (1988). The Smalltalk 80 adaptation distinguishes model, view, and controller as follows:

*"The model of an application is the domain specific software simulation or implementation of the application's control structure.*

*Views [...] deal with everything graphical: they request data from their model, and display the data.*

*Controllers contain the interface between their associated models and views and the input devices (e.g., keyboard, pointing device, time)."*<sup>234</sup>

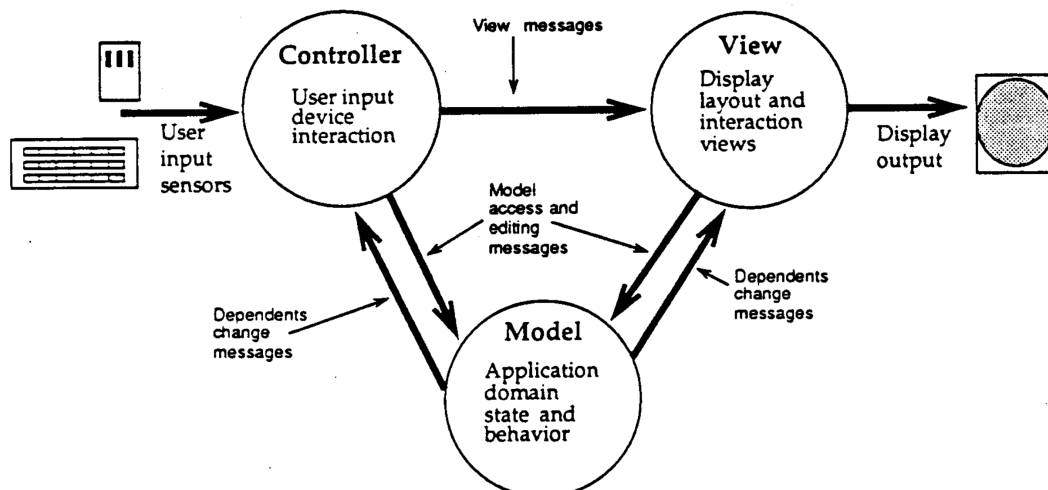


Figure 67: A schematic view of the relations between Model, View, and Controller in Smalltalk 80 (image taken from Krasner&Pope 1988)

<sup>233</sup> See [heim.ifi.uio.no/~trygver/](http://heim.ifi.uio.no/~trygver/)

<sup>234</sup> Extracts from the descriptions in Krasner&Pope (1988), page 27

Based on those definitions, Krasner&Pope describe the “standard interaction cycle” for an MVC-based application, schematically depicted in figure 67, as follows:

“The user takes some input action and the active controller notifies the model to change itself accordingly. The model carries out the prescribed operations, probably changing its state, and broadcasts to its dependents (views and controllers) that it has changed, possibly telling them the nature of the change. Views can then enquire the model about its new state, and update their display if necessary. Controllers may change their method of interaction depending on the new state of the model.”<sup>235</sup>

Comparing the Smalltalk 80 interpretation of MVC with Reenskaug’s, there are some major differences. While Reenskaug spoke about *one* controller which arranges views and provides an interface for them, according to Krasner&Pope (1988) there always is a controller-view *pair*, so every view has its own controller. In Reenskaug’s interpretation, the views were owned by the controller, which was necessary in his concept as it was one of the purposes of the controller to provide the views, arrange them and inform them about intended manipulations by the user. A controller in Krasner&Pope’s explanation, in contrast, refers to exactly one view. As there are many views in one application<sup>236</sup> and each of them has a controller of its own, the Smalltalk environment constantly determines the active view by sending messages to the controllers asking them if their associated view is active, e.g. by checking if the mouse pointer is within its view. While Reenskaug describes one controller for many views which by themselves interpret manipulations made to them, translating them into operations on the model, in the Smalltalk 80 version, a controller controls its single view and has access to the model on which it alone performs modifications as necessary. Change messages are sent to the model only from the controller, never from a view.

---

<sup>235</sup> Krasner&Pope (1988), page 28

<sup>236</sup> A view of a typical Smalltalk 80 application consist of many subviews. Every button, for example, consists of its own view and controller components, so a button controller controls only this very button and performs model manipulations only regarding manipulations to this very button.

The example of a tic-tac-toe application in LaLonde&Pugh (1991)<sup>237</sup> gives a good impression of the resulting back and forth between control and view which is unavoidable in those cases where the elements of the view are to be directly manipulated.

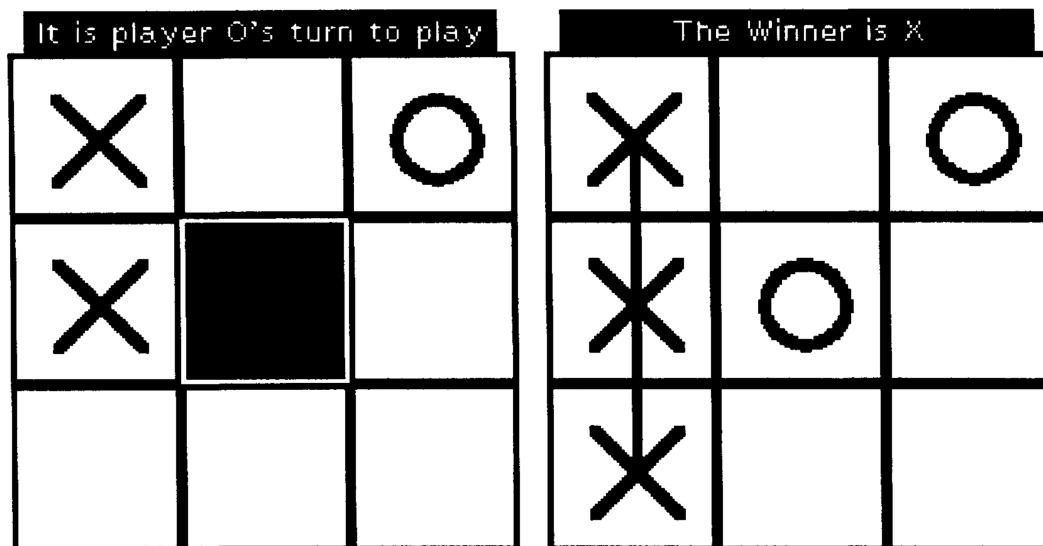


Figure 68: A snapshot of the tic-tac-toe game (image taken from LaLonde&Pugh 1991, page 107)

The model of the tic-tac-toe game contains all necessary game data such as the data structures for the board, which player's turn it is, or if there already is a winner. It also is a model of the game logic, so it provides methods for game actions which manipulate the internal data structures accordingly<sup>238</sup>. The TicTacToeView displayed in figure 68 shows part of the information stored in the model. Despite a visualisation of the current state of the board, it consists of a message area at the very top of the view. This message area shows, whose turn it is, if an attempted action happens to be illegal, e.g. when someone tries to set a new X or O onto a place where there already is one, or in case the game is over, who the winner is.

A TicTacToeController handles the input made to the view. All relevant input for this simple application comes from the position of the mouse pointer and the status of the mouse keys. Part of the input handling in the controller is related to

<sup>237</sup> See LaLonde&Pugh (1991), pages 103-113

<sup>238</sup> I assume the game logic of tic-tac-toe is well known, I shall not explain it any further.

the circumstances of closing the application. I disregard those here as they are not important for the concept. All further input made to the application has the purpose of placing an X or an O on the board. Users click on positions relative to what they see in the view. The coordinates of this position are captured by the controller, yet the controller by itself does not know anything about the rendering on the screen and thus does not hold information on which position of the screen corresponds to which square. Because of that, when the mouse button is clicked, the controller has to send a message to the view in order to determine row and column of the square the mouse pointer is hovering over. After receiving this information, the controller sends a ‘play’ message to the model which processes it according to the game logic and sends an update message to the view which then repaints the visualization of the changed state of the game.

The tic-tac-toe example reveals an unavoidable characteristic of this kind of controller. Users perform actions which refer to what they see in a view. Technically though, the input is processed by the controller which cannot handle it all by itself though, as it has no information about the graphical rendering on the screen. One might now suggest that the controller could just relay the user input information to the view in order to allow it to handle the manipulation to its rendering. This would be an interpretation compatible with Reenskaug’s, who saw the views as those instances which “know” how manipulations to themselves ought to be handled. Such an interpretation is not compatible with the Smalltalk 80 model though, so in the tic tac toe game, the view can only determine the clicked element in order to make the controller able to apply changes to the model using this information.<sup>239</sup>

Interpreting Responsive Positioning within the limits of the MVC concept of the Smalltalk 80 environment is only partially possible. Of course, the model could hold information about application objects. Views would render these objects as well as the structured space. The controllers for each view would be mostly

---

<sup>239</sup> I am well aware that when using standard elements, which is the clear purpose of this MVC variation, this back and forth between view and controller is not apparent for the programmer, as, for example, in order to implement a button, programmers would inherit from a general button class and would just override those parts of a ButtonController which determine the actions of a button click. All the aspects of determining button locations in the button view and sending the correct messages to the controller in case of a button click are encapsulated in classes from which a concrete instance of a button inherits. They thus are not part of the code programmers generally get to see or edit.

identical, inheriting from a hypothetical `ObjectDragAndDropController`<sup>240</sup> allowing the application objects to be dragged around. A problem though would arise regarding the location of the correspondence logic (consisting of the responsive positioning semantics and a structured background). This correspondence logic, by design, connects object positions, which are in the realm of the view, with object attributes, which are subject to the model. Consider the following possibilities:

- If the correspondence logic were part of the view, the view could display the spatial structure as well as the application objects according to the correspondence logic. In case of manipulations though, the logic which determines the new values for new object positions would then be part of the view which is problematic regarding MVC interpretation. To a part this problem also exists in the tic-tac-toe example where the view is necessary to translate the mouse position into that kind of coordinates the controller needs to be able to invoke the necessary model updates, yet in case of Responsive Positioning this would go even further as all spatio-logical operations necessary to update the model would have to take place within the view.
- If the correspondence logic were implemented as part of the model, there would be no separation between application logic and presentation as the necessary data for the view (the positions of the structures and the objects) would be evaluated and determined in the model which effectively means the model would directly control the view.
- If the correspondence logic were implemented within the controller, a view would have to access the controller in order to render its content. This is not possible according to this MCV version though, as views are required to be able to render the model independent of their controller.

---

240 Such a structure, of course, does not exist in Smalltalk. It is my own imagination.

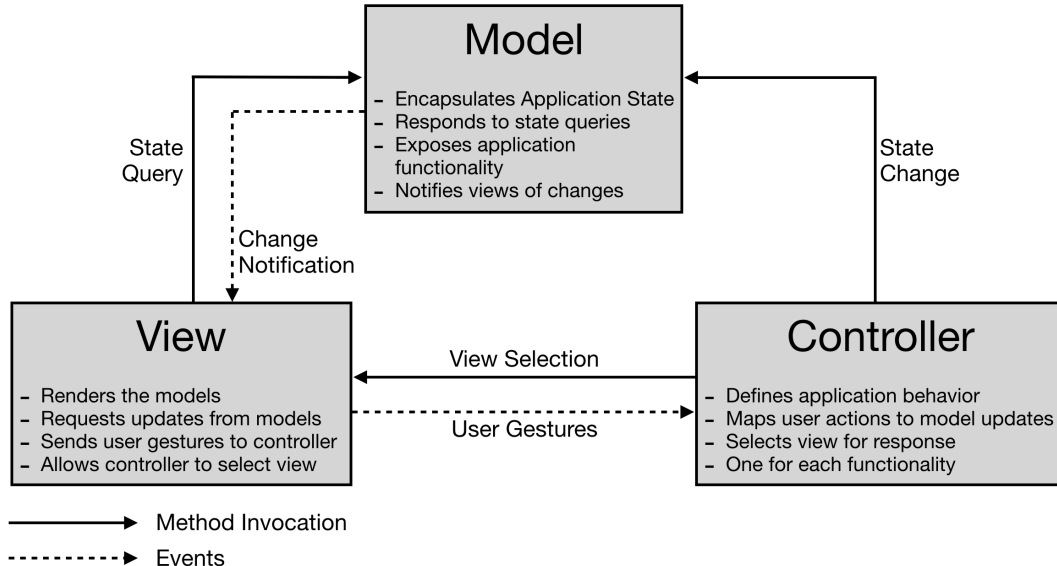
**Case 3: Today's Standard Model-View-Controller Implementation**

Figure 69: "A Common MVC Implementation" (redrawn from Eckstein 2007)

When MVC is used today, many developers refer to an interpretation similar to that of figure 69. This interpretation again sets a different focus on view and controller, which at least in case of the controller is quite different than both concepts described so far. In the modern interpretation, e.g. in Eckstein (2007), low-level considerations about how basic input provided by keyboard or mouse is interpreted, is not in focus anymore, so consequently the view component not only “renders the contents of a model”, which it does in every MVC concept, but also “recognizes a GUI action” which it sends to the controller in the form of “user gestures”. The role of the controller, in this case, is one of translating the gesture into a command for the model:

“The controller translates the user’s interactions with the view into actions that the model will perform. In a stand-alone GUI client, user interactions could be button clicks or menu selections[...]<sup>241</sup>

In this MVC interpretation, there neither is one controller per tool (Reenskaug) nor is there one controller per view (Smalltalk 80) but, according to Eckstein, there rather is one controller per functionality. One controller controls several views, so, similar to Reenskaug’s interpretation,

<sup>241</sup> Eckstein (2007)



it is the instance which selects the appropriate views for a certain functionality but, as in Krasner&Pope, it is the interface to the model, so all model changes have to originate in the controller or be relayed by it.

Applying this MVC interpretation to Responsive Positioning does not work well as the instances of model rendering model update are even more separated than in the Smalltalk 80 interpretation, resulting in similar problems for the correspondence logic:

- Like in the Smalltalk 80 interpretation, the correspondence logic could be in the view this would effectively mean that “GUI actions” had to be fully evaluated in the view, so the result would not be just a “user gesture” but rather already a manipulation command for the model.
- Like in the Smalltalk 80 interpretation, it is not possible to implement the correspondence logic in the controller as it is needed in the views in order to render the objects and the structured space.
- It is, of course, technically possible to interpret the correspondence in the model, but in that case, view and controller would not just access the application state but also the representation data. As a consequence of this, information only necessary for the views would be determined and processed in the model, effectively dissolving the separation between application logic and presentation.

Wrapping up the results of this brief look into different (partially historical) MVC interpretations, Responsive Positioning, if at all, is best represented by the original, yet widely unknown, Model-View-Controller concept provided by Reenskaug. In his interpretation, the views have a central role in interpreting the modifications made to them. Only his interpretation allows for manipulations *in the view* which are also evaluated *in the view* according to structures which happen to be *in the view*. In Responsive Positioning, a manipulation always relates to something spatially visible, the structured background, which is independent of the manipulated object itself, so the correspondence between object attributes and object positions requires a view which contains the manipulable objects, the structured background as well as the correspondence logic that connects them. It needs a view which can interpret the manipulations made to in relation to what is

shown. The MVC concepts described here, except the one by Reenskaug, do not provide this.

There are more modern adaptations of the Model-View-Controller concept which reinterpret the role of the controller in such a way it becomes an intermediary between the model and the view or even a separate model for a view. In the latter case, a view is not a rendering of the application model, but rather becomes a rendering of an explicit presentation model. Such a model can contain all necessary data about the objects, the structured spaces as well as an implementation of the Responsive Positioning logic. I will explain a Responsive Positioning specific adaptation of such a model in chapter 6.4 after investigating important aspects of Responsive Positioning applications as these substantiate the need for such a concept.

### **6.3 Aspects of Responsive Positioning User Interfaces**

To explain important aspects of Responsive Positioning applications, in this section, I present near-real-world examples of such applications. By presenting and explaining the examples, on the one hand, I want to show that there are some sensible applications for which Responsive Positioning can be used as a user interface technique. On the other hand, these applications show typical characteristics of Responsive Positioning interfaces as well as aspects of the design of structured backgrounds. As the examples have been chosen in order to explain characteristics and typical obstacles, regarding real-world suitability there are some restrictions to keep in mind:

- The applications presented here resemble real-world applications which could exist but are simplified in order to be easily explainable. If one wanted to actually implement and use these applications, their user interfaces as well as their functionalities would have to be extended and would need a final polish.
- In most real-world examples, one would likely not rely on Responsive Positioning alone. In the examples given here, for the sake of simplicity, I mostly restricted myself to Responsive Positioning and left other UI techniques aside as long as they are not essential to the application.

- Using Responsive Positioning as the interface technique for these applications surely is not the only possibility of controlling them. In some cases, it may not even be the best possible solution for a good user interface. In chapter 7.1, I reveal some thoughts on what kind of applications might be best suited for Responsive Positioning.

### 6.3.1 Interrelated Objects

The application in figure 70 shows a view of an application which can be used for project planning and budget management. The idea behind this application is to find out, which projects can be carried out and which cannot. The view in the figure consists of two sections which are indicated by dotted lines. The ‘planning’ section on the right-hand side serves two purposes. Objects can be assigned to a department as well as to the common budget by dragging them into a respective position in the ‘in budget’ region. Each project object carries an attribute which specifies its individual cost. How this attribute is visualised, determined or changed is not part of this view. Double-clicking on an object could, for example, open a separate window in which its cost can be configured.

The left-hand side section, the budget section, shows the status and allows the manipulation of the ‘amount’ of the current budget. It thus consists of an axis which refers to an ‘amount’ attribute. The object within this region represents the current budget. Its ‘amount’ attribute value and thus its spatial position in relation to the axis represents the accumulated costs of all those project objects in the ‘in budget’ region of the planning section.

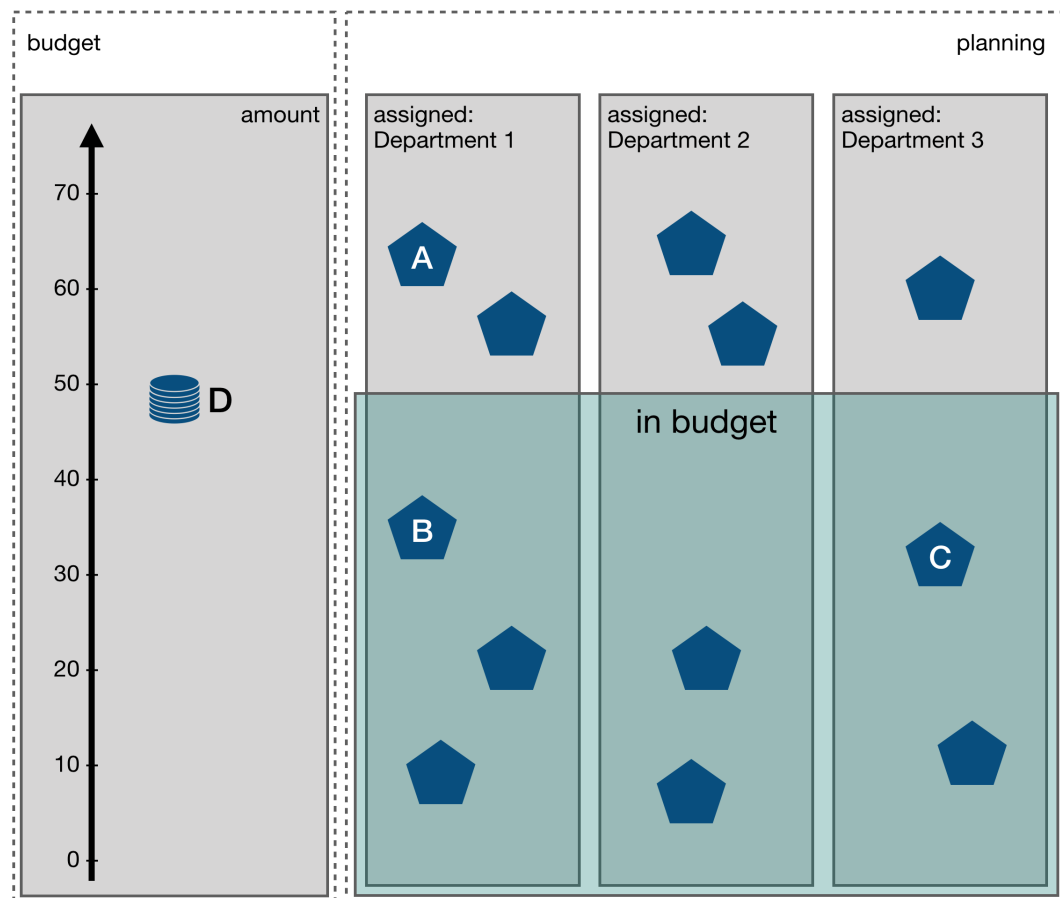


Figure 70: A view of an application for project management

The application logic determines the interrelations between the objects. In this example a spatial manipulation of almost any object, and thus any attribute manipulation, leads to attribute updates and is associated with positioning updates of other objects:

- When project object A is dragged into the 'in budget' region, and thus through Responsive Positioning, its respective attribute gets set, the application logic updates the 'amount' value of the budget indicator object D which translates into a spatial update of said object.
- When, in turn, the budget indicator object D is manipulated by moving it up or down, thus changing its 'amount' value, the budget allows more projects to be carried out or is too tight for those projects currently in the 'in budget' section. Assume the value is increased. In this case, the

application logic can choose one or more of those project objects which currently are not 'in budget' and set their 'in budget' status. It could, for example, change the respective attribute of object A which would consequently move down into the 'in budget' region. Similar evaluations are made when, by moving the budget indicator D, the 'amount' value of the budget is decreased. In this case, the application logic determines objects which cannot be carried out anymore and, for example, unsets the 'in budget' attribute of projects B and C which consequently move to a position outside the 'in budget' section.

### **6.3.2 Manipulation Restrictions**

While in the budget management example many object manipulations are possible, others would be meaningless or even misleading. In the music player example at the beginning of this chapter, some manipulations were dealt with by immediately resetting object attributes to their previous values within the application logic. In the user interface, this means that some object movements are dealt with by immediately moving them back to where they came from. Resetting invalid attribute manipulations this way is necessary, as the application logic is required to work correctly regardless of the functionality of the views.

Nevertheless, within the views, meaningless or dangerous manipulations should already be made impossible as implementing restrictions on this level allows for richer user interface responses than allowing manipulations in the views which are reset immediately afterwards. While an object is in the process of being repositioned, those areas which are no valid targets for the object could, for example, be greyed out or, while the object is hovering above these areas, the mouse pointer could change itself, indicating that a movement to this position is not possible. Should an object nevertheless be moved to a position where it is not allowed to be, the interface could react by discarding the operation and moving the object back to where it came from without ever triggering an attribution function. In the view depicted in figure 70, the budget indicator object D, for example, may not be dragged into the planning region as, for this application, it does not make any sense to assign a department or an 'in budget' property to this object. Similarly, project objects have no business being in the 'budget' section.

### 6.3.3 Multiple Contexts in Multiple Sections

Figure 71 shows a view which allows teachers to assign participants to learning groups and also allows them to assign working material to them. The view could be a part of a learning management system which makes use of a database of available learning material.

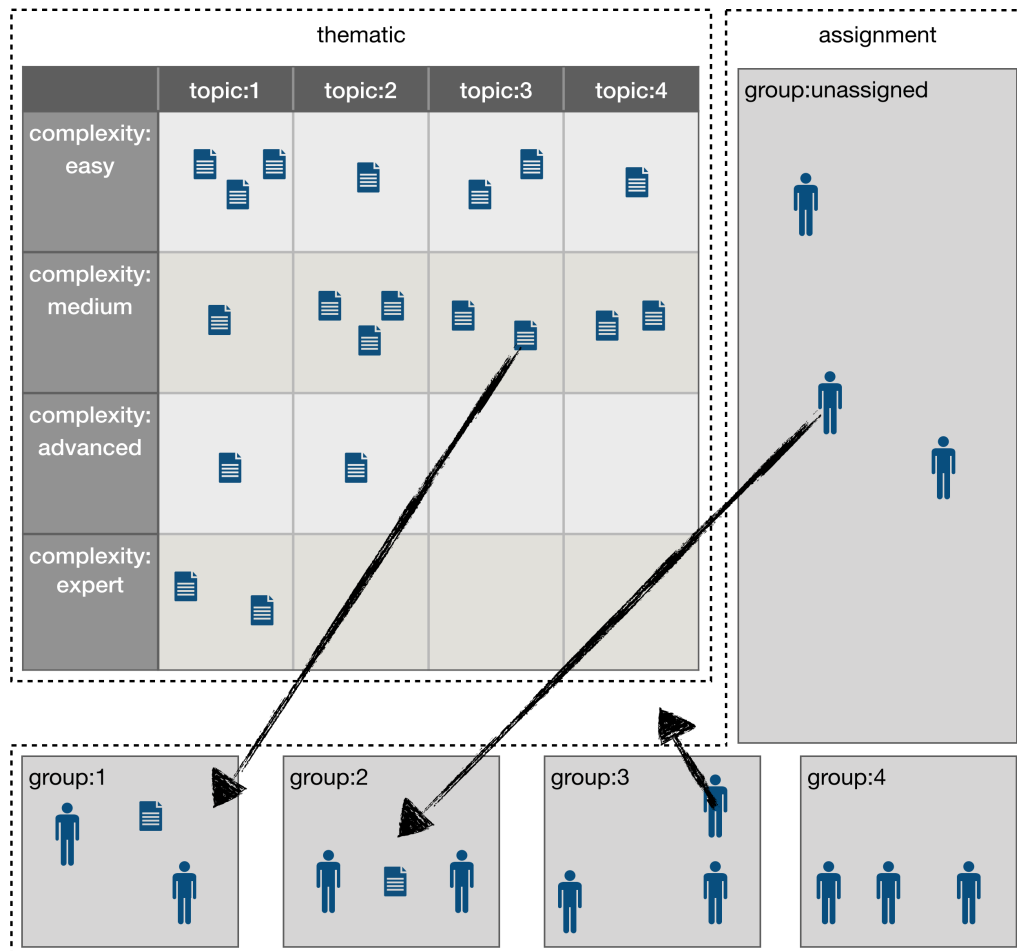


Figure 71: A group assignment view of a learning management tool.

While in the budget management example in figure 70 objects explicitly could *not* be moved between sections, this example relies on object movements between sections and the associated context changes. The view consists of two sections. A thematic section consists of a matrix on the top left of the screen using which documents are organised in relation to a topic and their degree of complexity<sup>242</sup>.

<sup>242</sup> I assume that the selection of topics and the configuration of the groups has been made in another view most likely using other user interface techniques. The objects were then fetched

The assignment section consists of four regions at the bottom of the screen as well as a large region at the top-right. All of these regions relate to the ‘group’ attribute. The lower four regions all refer to a concrete student group, here given values from 1 to 4, whereas the large region in the top right refers to an implicitly ‘unassigned’ state which is the initial value of participant objects when they first appear in the application.

Using this view, a teacher can assign participants as well as working material to the specified groups by dragging the respective objects into the group regions. The documents which represent the working material thereby change their evaluation contexts to ‘assignment’ which means that though being dragged into group regions, they keep their topic and complexity information, so when the group work is over or when the same object is used in another view of the application, it can appear in its thematic context again<sup>243</sup>.

Changing the objects back to their ‘unassigned’ state in this view can be implemented by making intelligent use of the neutral background, the void between the sections. In figure 71 a participant object is dragged from the “group:3” region into this void. Dragging it there sets its ‘context’ attribute to ‘neutral’ which triggers a contextChanged in the application logic which, in turn, can reset the object’s ‘group’ attribute to ‘unassigned’. In case of working material, a similar operation would remove any remaining group association of the object and would set its ‘context’ attribute to ‘thematic’. In both cases, the neutral background would become a reset area which, in case something is dragged onto it, sends it back to its original positions, which in case of working material is the ‘thematic’ section and in case of participants is their ‘unassigned’ group association.

#### **6.3.4 Automatic Context Switching**

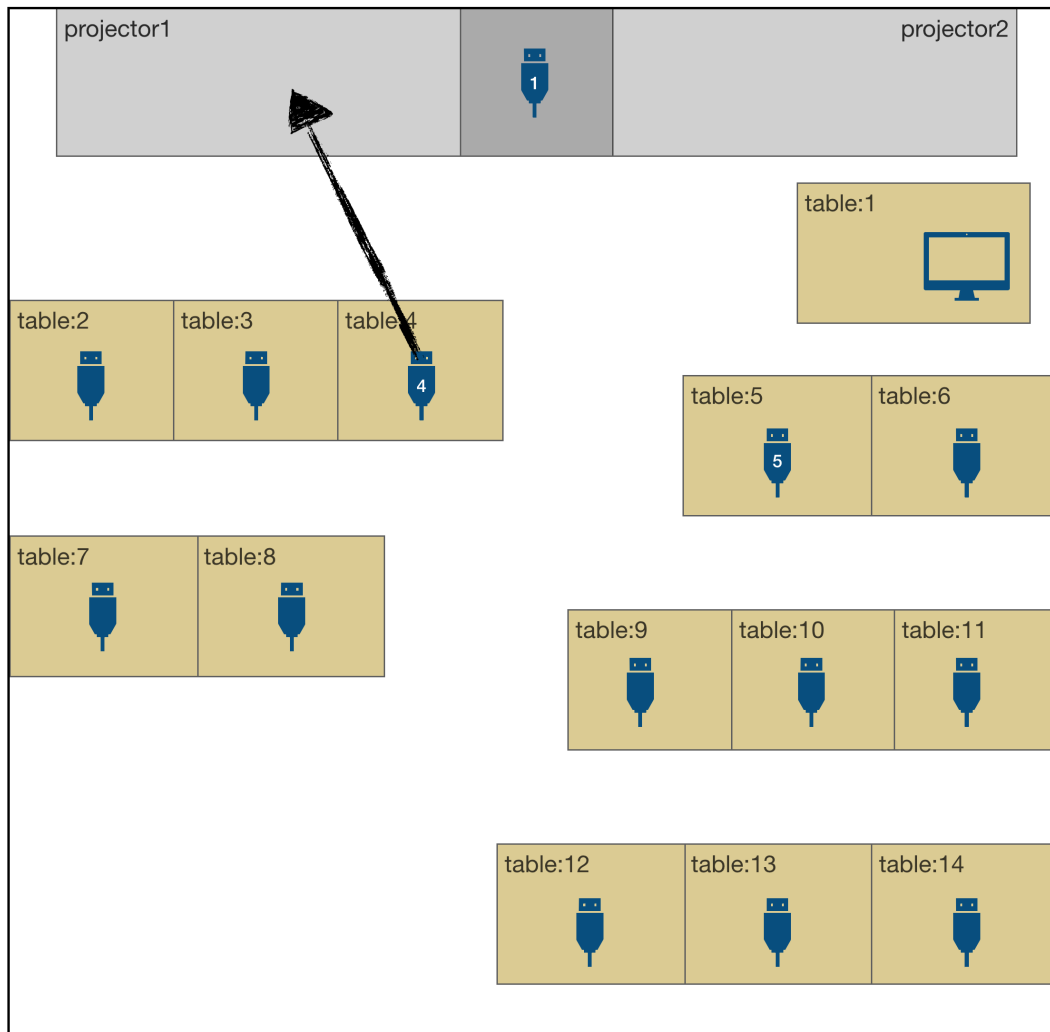
The application shown in figure 72 has a real-world origin in one of the smaller lecture rooms at the Heinz Nixdorf Institute at Paderborn University. In the room, there are tables which are equipped with power outlets as well as with VGA sockets which can be used to connect laptops. Attached to the ceiling, there are two projectors, and in the front, there is a high desk which also has a power outlet

---

from a comprehensive research database and automatically positioned accordingly.

<sup>243</sup> In such a case each of the views had to refer to a different context attribute, so the respective context attribute would have to be subject of the configuration of a view.

and a VGA connector which can be used to connect the laptop of the lecturer. In addition to that, there is a small desktop computer with a pen-based screen which serves the purposes of a virtual whiteboard. A multimedia control device is attached to the wall and provides an interface which allows the users to select the input sources. At the moment, the control device uses a classical button-based user interface.



*Figure 72: A view of an application for controlling the multimedia equipment of a lecture room*

The optical structure of the view shown in figure 72 roughly resembles the topology of the seminar room. It consists of two sections which for the sake of graphic simplicity are not explicitly indicated in this figure, yet all yellowish



regions belong to the location section whereas all grey regions belong to the presentation section. In the location section at the bottom, tables are modelled as regions which refer to a 'table' attribute. The objects in this section, except the one representing the desktop PC, represent the VGA connectors of the individual tables. The objects' 'table' attributes is set according to their location of the connectors in the room. This value cannot be changed as the connectors are hardwired, and thus their position is not subject to any manipulation. Two regions at the top of the screen constitute the 'presentation' section. They represent the assignment of an input source to one or both of the projectors in the room. If a source could always be assigned to only one projector at a time, a good solution for the upper regions would be letting them refer to the same projector attribute with one having a value of 1 and the other having a value of 2 (see figure 49 in chapter 5). A problem would arise if one wanted to let these two overlap in order to allow one input source to be presented on both projectors. Positions in the overlapping area would refer to both attribute values at the same time, which would create a conflicting spatial structure so the resulting structured background would not be suitable for Responsive Positioning anymore. There are two possibilities of implementing the region for both projectors in an RP-suitable way. One would be to use three regions or a three-part segmentation where the middle region or segment would refer to a value which would represent both projectors (e.g. 3). The other solution, which has been chosen here, is using different attributes for each projector, so the left region refers to a binary 'projector1' attribute while the right region refers to the 'projector2' attribute. Both regions overlap, so for objects in the overlapping area both attributes are set. In the current state of the application depicted in figure 72, object 1<sup>244</sup> is positioned at the intersection of the two projector regions, which means both projectors show the image of the connected source, which is most likely the laptop of the lecturer.

---

244 The object names correspond to the numbers of the tables. For simplicity reasons, only the objects manipulated in the example are labelled.

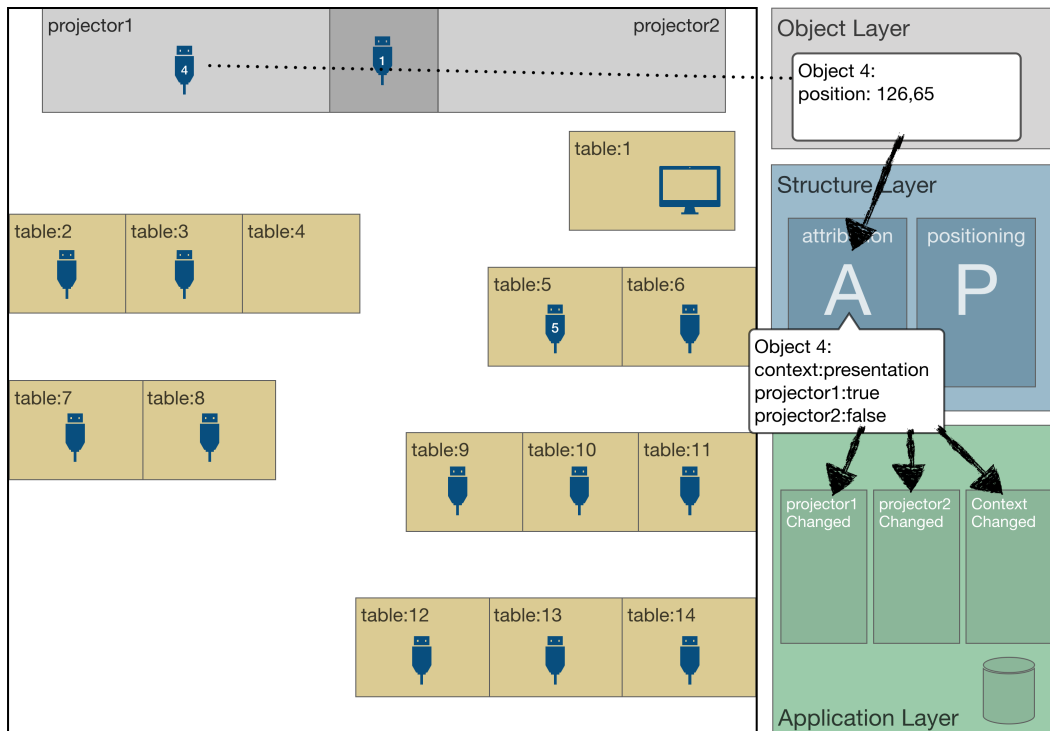


Figure 73: The state of the application after object 4 has been dragged to the 'projector1' region.

Figures 73 to 75 show the state of the application both in regards to what can be seen in the user interface (on the left-hand side) as well as the passing of data within the Responsive Positioning application stack (on the right-hand side). Figure 73 shows the state of the application immediately after object 4 has been dragged to the 'projector1' region. On the object layer, the object's position changes to (126,65). This triggers the attribution function of the structured background in the structure layer to calculate new attributions for the object. It first determines the new context of the object, which is 'presentation', and then evaluates the position according to the structures of the 'presentation' context resulting in an attribution set of  $\{(set, context, presentation), (set, projector1, true), (unset, projector2, true)\}$ . With this evaluation, the Responsive Positioning operation is finished, so the changed object attributes can now be processed by the application logic.

On the application layer, the attribute changes trigger functions reacting to changes of 'context' and of the 'projector1' attributes. For the purpose of

simplicity, the handling of context changes by the application can be omitted here. If they trigger a reaction at all, it could be that dragging an object onto neutral ground could be interpreted as resetting the object to its location context. A `projector1Changed` function in the application logic handles the new assignment of object 4 by setting the input of the left projector to the VGA socket of the table stored in the 'table' attribute of the object. This means that even though object 4 just switched its context to 'presentation', the application logic needs the information of the 'location' context to do its work. This is no problem, as all attributes are available regardless of the context an object is evaluated against. As a result of the input source changes, the VGA source of the teacher's desk, which was presented on both projectors previously, now is only connected to the right projector. The application logic reflects this change by unsetting the 'projector1' attribute of said object 1 or by setting it to false (see figure 74).

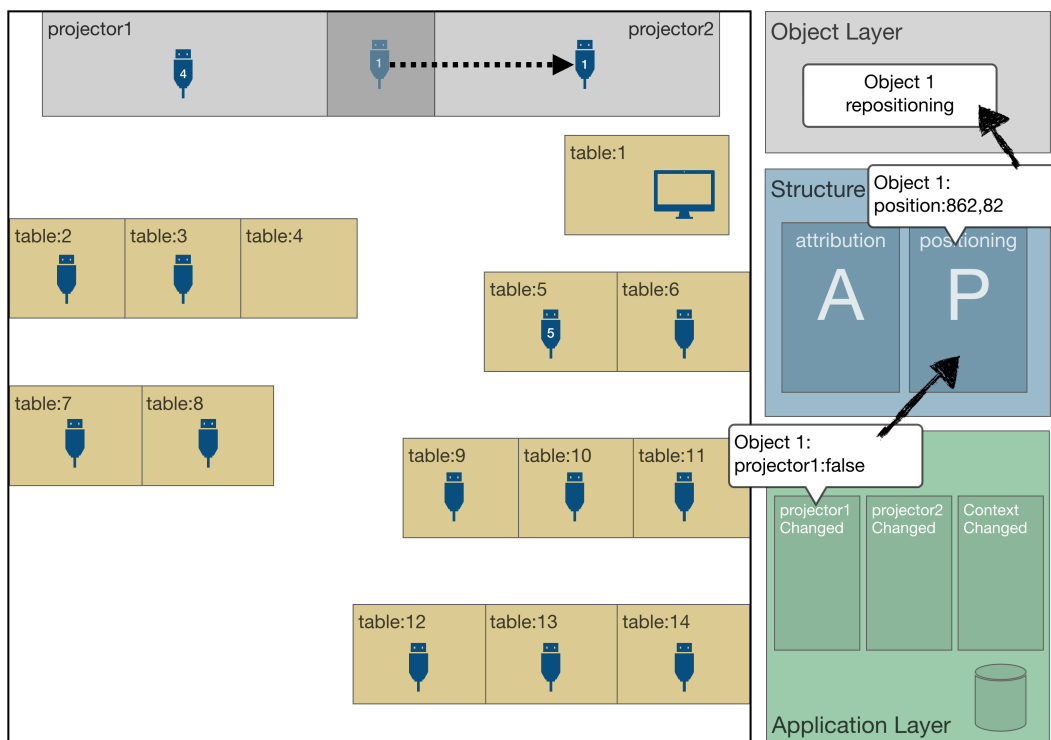


Figure 74: The evaluation by the application logic and subsequently by the positioning algorithm results in a position change.

The change in the attributes of the application object is processed by the positioning function of the structured background which determines a new spatial

position for object 1 which does not overlap with the ‘projector1’ region anymore. This positioning change is then carried out by the object layer so, at the end of the evaluation process, the positions of all objects in the Responsive Positioning user interface correspond to their assignment statuses. As all these operations happen rapidly, from a user’s point of view, the action of placing object 4 in the ‘projector1’ region is immediately reacted upon by an input channel switch of the projector and by object 1 moving to the right.

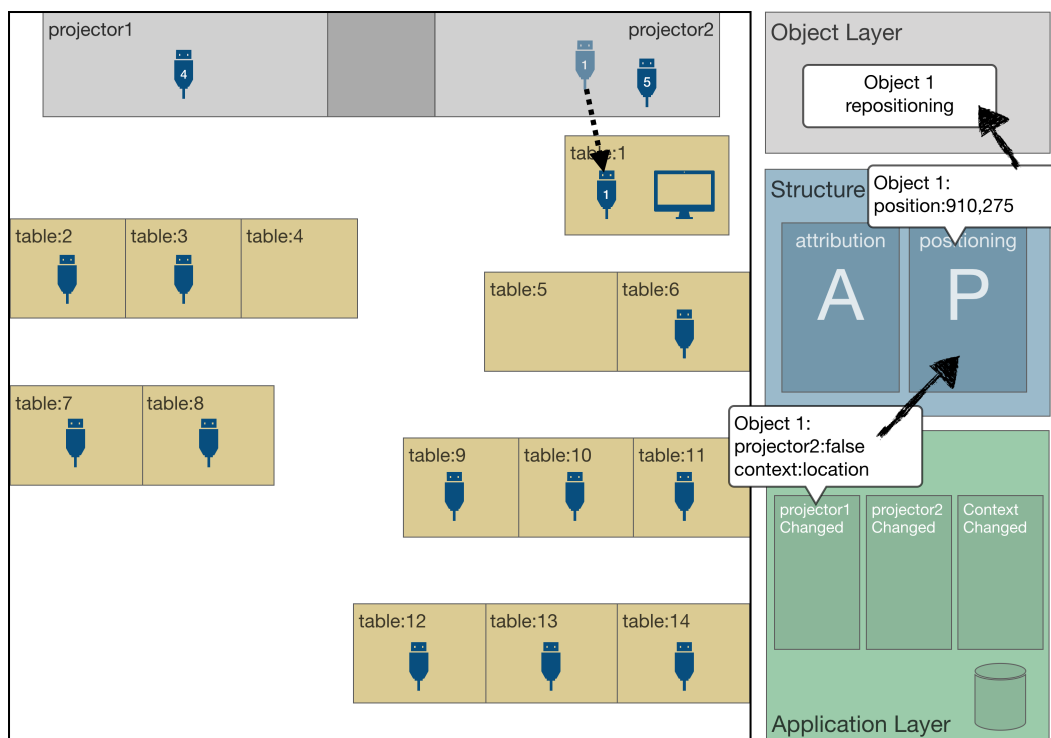


Figure 75: The evaluation by the application logic results in a context change which lets object A move back to its table location.

In a further step, assume a student wants to connect her laptop to the system and visualise it on the right projector by dragging object 5 to the ‘projector2’ region. This change is translated to attributions  $\{(set, context, presentation), (unset, projector1, true), (set, projector2, true)\}$ . Figure 75 shows that, after setting the input channel of the projector accordingly, the application logic has to reset the values for object 1, as the VGA socket of table 1 now is not presented on any of the projectors anymore. Such a reset means that both the ‘projector1’ as well as the ‘projector2’ attribute are unset or set to false and the ‘context’ attribute is reset

to ‘location’. The positioning function of the structured space reacts to these changes by creating a new position for object 1 which, as the context has changed, is its position in the location section.

In this example, the context has been used not only not to lose the location data of the object by not evaluating it when the object is in the presentation context but also as a part of the visualization of the object state itself. An object being in a section visualises whether or not a source is currently projected.

### 6.3.5 Combinations with other UI techniques

Figure 76 shows part of the user interface of a music player application. The structure on the left-hand side of the figure is used to search for songs in a database. This is done by entering text into a search bar. This text is sent to the application which processes it and generates objects which are displayed in the structure below. While this result area is a responsive positioning structure, the search bar is not. It is positioned in relation to the structured background, but in regard to Responsive Positioning, it is neither an element of the structured background nor is it a foreground object.

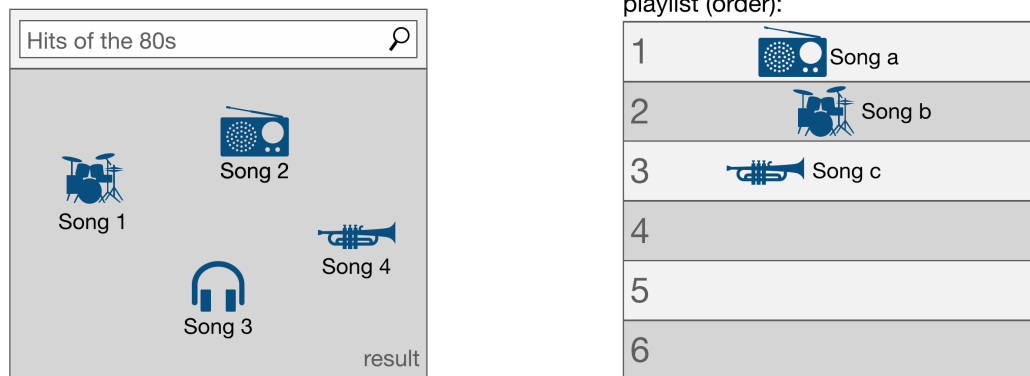


Figure 76: Parts of the user interface of a music player application

Being able to have such *foreign objects* is very important. While this thesis hopefully suggests that many applications can use Responsive Positioning within their user interfaces, there often will be parts of them where Responsive Positioning is not the right choice. Entering text, like in this case, definitely is one of those functionalities which cannot be done using Responsive Positioning. Many other foreign objects are thinkable. In the multimedia control application in

figure 72, for example, an additional “All off” button could trigger a function in the application where the projectors are turned off, and all objects return to their location section.

### 6.3.6 List Structures

The right-hand side structure of the music player user interface in figure 76 is a playlist. From a formal point of view, lists are not compatible with the definition of structured backgrounds made in chapter 4. This is because a list always specifies an object-to-object relation. Even if it is assumed that this criterion would not exist, trying to specify an attribution function for lists would in many cases prove to be problematic. Imagine a list which shows products ordered by price. Of course, it is possible to envision a positioning function for this kind of list which would order objects accordingly and which could even handle the addition of new objects to the list or removing objects from it. While the positioning direction of a correspondence between the price of the product and rank in the list is not problematic, the other direction, determining an object’s price from its position in the list, is. Imagine a scenario in which a list structure is used in order to determine prices in relation to each other by ordering them spatially. Such a scenario would have to include the need for changing the order of the objects within the lists. Imagine the consequences of dragging objects up and down in such a list. A change in an object’s position in the list would have to imply changes in the ‘price’ attribute of that object, yet it remains unclear, how such a price should be determined. The position in space does not convey the necessary information, but neither does the relative position of the object in relation to the objects above and below. This relation only provides ranges in which the price may be. It must be lower than the one above it and higher than the one below it. Of course, in a concrete scenario, this could mean that the average value is chosen, but that can hardly be a general solution for a common list structure which could be part of a Responsive Positioning semantics as this solution is not independent of concrete application semantics.

There is a type of lists in which this problem does not exist because they *only* rely on the rank of an object, which means that this rank is not coupled to anything<sup>245</sup>.

---

245 Erren (2010) suggested lists which are initially ordered by some attribute but where this attribute is implicitly changed to a rank attribute as soon as spatial manipulations among the entries take place (page 105). Such a list would not be usable in Responsive Positioning, as it

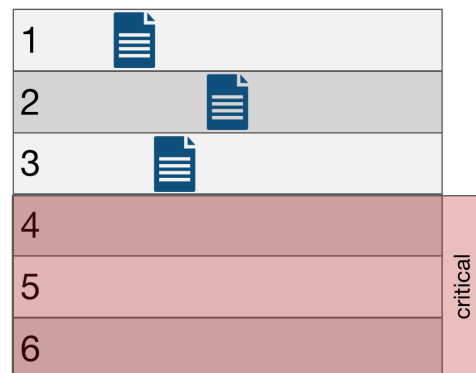
Imagine a top-20 list of one's favourite episodes of a TV show. This list contains objects with a rank attribute from 1 to 20. Dragging the object with rank 5 to the position of the object with rank 3 would cause a reordering of a number of objects. This, by the way, shows that the list is an object-to-object relation. Setting that aside, there is no problem in determining a new rank for every object. The playlist in figure 76 is another example of such a list.

While lists are not compatible with Responsive Positioning as such, under certain circumstances the problem of rank-lists can be solved on the application level, where an application could easily make other structures "listy". The structure in figure 76, for example, is a segmentation which refers to the attribute of 'order' with values from 1 to 6. Assume there are three objects in the list already and another object is dragged from the search results area to position 2 of the list right next to (or even on top of) the existing element at this position. The Responsive Positioning logic assigns an 'order' value of 2 to the newly inserted object. As Responsive Positioning does not have any concept of lists, for the semantics it does not matter that now there are two objects carrying the same order value. To sort this problem out, the application can react to the attribute assignment by recalculating the 'order' attributes of the other objects and change them accordingly. Another solution, creating similar results, would be using an axis for which an application automatically equalises the distances between elements, so when an object is dragged into a position where it gets an 'order' of 2.45, the objects at positions 1 and 2 keep their 'order' attributes while the new one gets a 3 and every object which had an 'order' value greater than 3 gets its rank increased by 1.

As mentioned before, lists are not compatible with Responsive Positioning semantics. While list structures can be "simulated" in the way described above, overlapping them with other structures can result in unwanted or unstable behaviour, so lists like this one have to be used with great caution. Figure 77 shows an example of a list structure which overlaps with a region. Placing an object into the first position of this structure would move all objects down one in rank. This automatically puts the object, which is then in rank 4, into a state where it cannot be positioned by the structure anymore.

---

would not serve the purpose of being able to perceive and manipulate a specified object attribute by manipulating the object's position.



*Figure 77: An overlay of a list and a region*

This is a somewhat schizophrenic situation as the very structure which made the object attribute change thereby puts it in a state where it cannot be positioned anymore. One might, of course, argue that such a switch means that an object, which due to the list structure has been moved to the third position in the list, automatically would set the ‘critical’ attribute, yet such an interpretation is not compatible with the semantics provided by responsive positioning either. It would require an interpretation where one structure dominates the other. The list would reposition the object, and the other structures would determine new attributes for this new position. Implementations of such interpretations are not impossible, but they are beyond the Responsive Positioning semantics of this thesis.

### 6.3.7 Views on the Same Object Ensemble

The last idea for a Responsive Positioning application presented in this chapter is a configurator for a beverage dispenser using which I can illustrate a number of concepts.

Figures 78 and 79 show two different configuration views for this beverage dispenser. Figure 78 can be used to configure which beverage has which ingredients. It consists of a clever arrangement of different regions, which allow for quite a number of tasty combinations. Figure 79 shows a view which can be used to configure the quantity of liquid for the individual beverages. In figures 78, the objects named ‘Espresso’ and ‘Double Espresso’, for example, are configured similarly, yet figure 79 reveals that they differ in the aspect of how much coffee is to be produced.



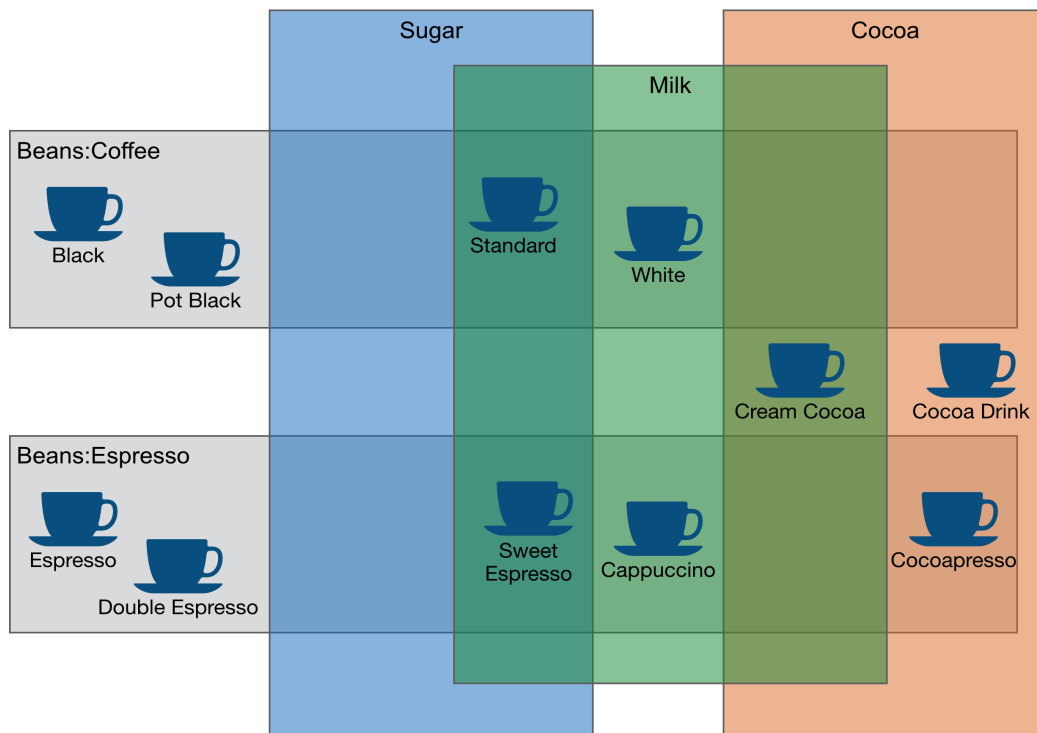


Figure 78: The ingredient configurator of a beverage dispenser

As in figures 78 and 79 the same objects are placed according to different structured spaces, they are examples of views on the same object ensemble. In accordance with the determinations made in chapter 5.5, these two views, if there is enough screen space available, should be arranged in a way they can be perceived at the same time. This allows users of the configuration application to keep both aspects in sight while making their adjustments. Highlighting objects which are selected in one view in the respective other view allows the users to find the objects in one structure and enables their modifications in another structure. While a simultaneous presentation of both views does make a lot of sense, combining both of them into a single view by defining two sections, like in the group assignment example explained before, would wrongfully suggest that the beverage objects are either in their ingredient configuration or their quantity configuration context between they would switch. This is not the case though. Beverages always both have ingredients as well as quantity. The objects are not temporarily interesting regarding only one piece of information. A one-view implementation would thus be disadvantageous. To check the full configuration of

the beverages, users would have to continuously change the contexts of the individual objects, yet they could never get all necessary information at the same time.



Figure 79: The configuration screen for the quantities of each beverage.

### 6.3.8 Objectification and “Unreal Objects”

One may have noticed that figure 78 does not allow for all combinations of ingredients. It is, for example, not possible to create a sweet cocoa, mixing cocoa with sugar. Imagine now, the machine would be even bigger and would, for example, also provide banana or strawberry drinks. With that many options, structures like the one in figure 78 are not possible anymore, so if Responsive Positioning is supposed to be used, another solution has to be found.

In all Responsive Positioning applications, the user interface designer has to consider which aspects of the application are to be interpreted as manipulable objects and which aspects are their properties and must thus be reflected as spatial

properties of the structured background. Reconsider the multimedia control example in figure 72. For this example, it has been decided to make the input sources the manipulable objects carrying their current assignment to one of the projectors as attributes.

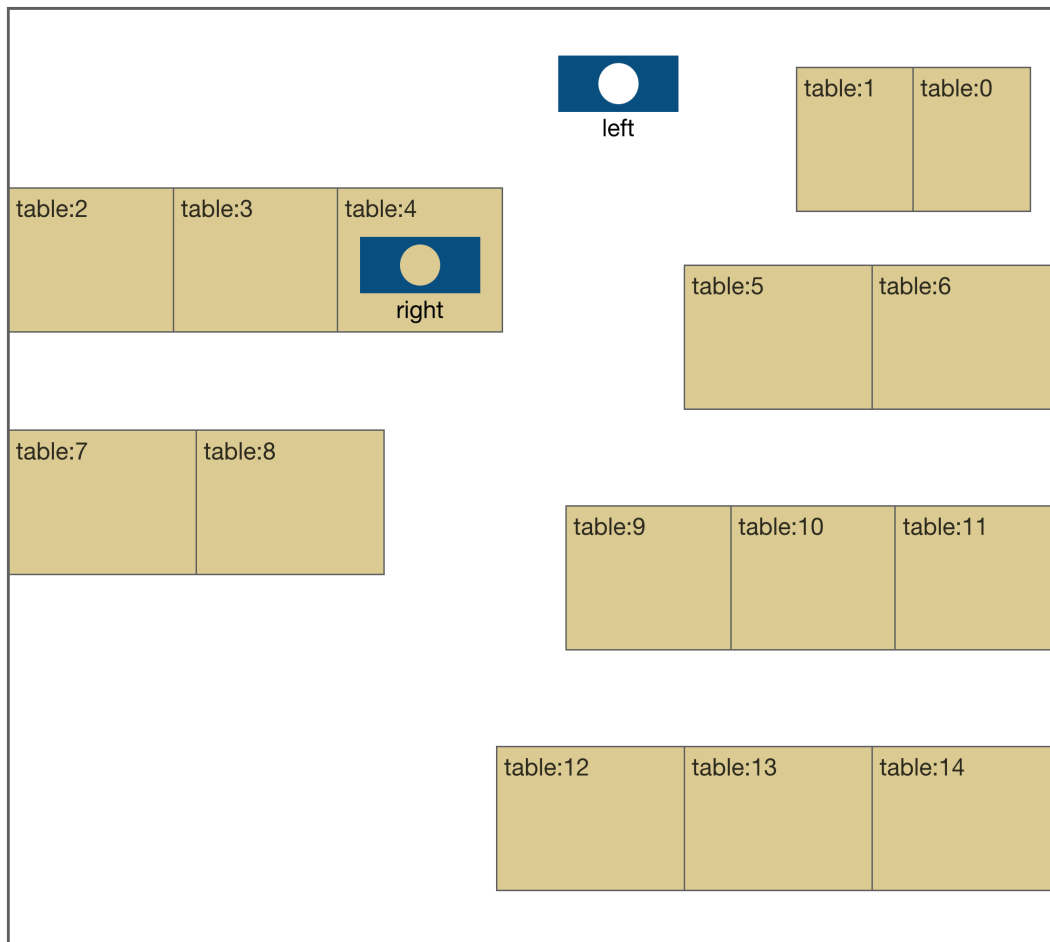


Figure 80: An inverted solution to the application view shown in figure 72

Figure 80 shows a different possible structure which could be used for the same purpose. In this case, the projectors are the objects of manipulation and can be dragged to the tables thereby changing their input channels. From a technical point of view, this version is a lot simpler. Fewer basic structures are needed, and there are only two objects which have to be positioned. There is no need for different contexts or sections, and the objects which are manipulated represent those affairs which are in fact manipulated by the application as it is the input

channels of the projectors which are manipulated when assigning a source to a projector. Despite these inherent advantages of the solution in figure 80, the solution in figure 72 has the advantage of spatially indicating where the projection of a connected device occurs. This is not the case in figure 80 where, in the depicted state, the object indicating the right projector is positioned on the left-hand side of the view as table 4 happens to be on the left-hand side of the room. Additionally, while it is true that technically the input channel of a projector is changed by the application, this does not necessarily reflect what users of the system might have in mind when they want to present their content. It is well possible that they would rather claim “I connect my device to the left projector.” instead of “I connect the left projector to my device.”<sup>246</sup>

In figure 78, a similar decision had to be made. The ingredients of the beverages were made parts of the structure while the beverages came up as manipulable objects. This design is quite obvious, as one would say “I configure cappuccino to have espresso as an ingredient,” yet hardly anyone would formulate it like “I configure espresso beans to be in cappuccino.” Additionally, creating new beverages is a purpose of such an interface while the set of ingredients is given by the machine. Thus it makes sense to put those aspects which are fixed into the background structures while making those aspects which are to be configured manipulable by “objectifying” them. As combinations of ingredients in figure 78 have to be expressed in a single position, there must be an overlap of regions for each configurable combination which, as stated, can be problematic in case of a high number of possible combinations. Leaving Responsive Positioning aside for a moment, the same assignment status expressed in figure 78 can also be expressed using the structure depicted in figure 81.

---

<sup>246</sup> In fact, when I presented a mockup solution like the one depicted in figure 80 to an audience of coworkers from other departments at my research institute, they considered the solution to be confusing and asked why I moved the target to the source instead of moving the source to the target.

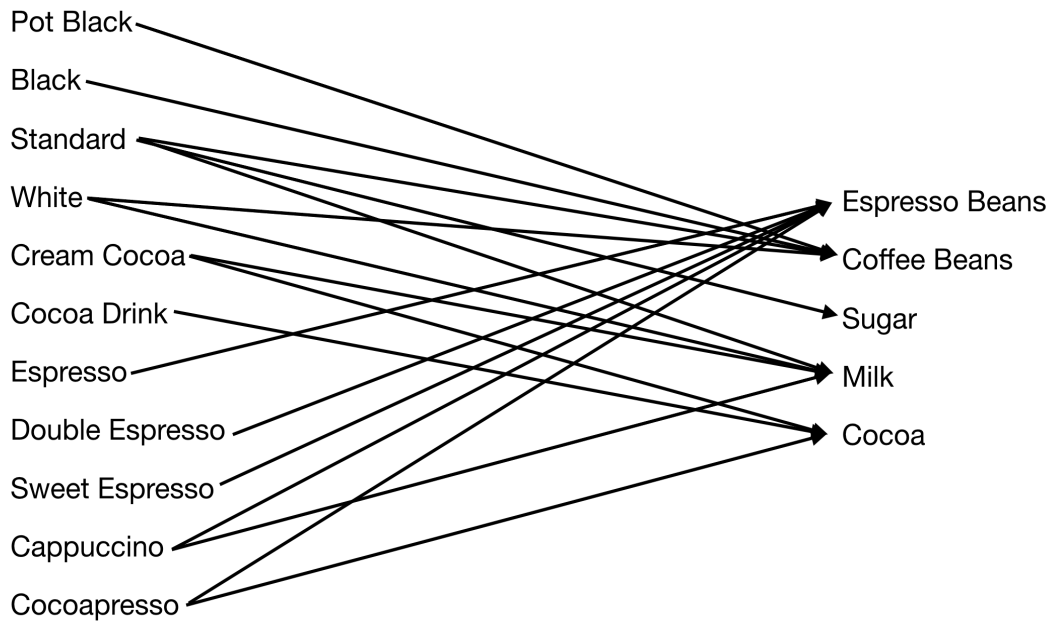


Figure 81: The mapping between beverages and their ingredients as a bipartite graph

A bipartite graph like the one in figure 81 represents a mapping between nodes of one kind and nodes of another kind. Such graphs by themselves are, of course, no structures which could be used in Responsive Positioning as they are not compatible with its semantics. Still, it makes sense to look at the graph for a moment, as it helps to illustrate the problem of having to decide, which aspects of an affair can be made objects and which aspects are object attributes and thus become spatial properties. When creating graphs, there are two answers to that question of which one is more compatible with Responsive Positioning than the other.

1. The nodes of the graph, beverages as well as ingredients, are objects which can be positioned in space. Connections are made by using special objects called edges or connectors. Described like this, connectors are part of *two* object-to-object relations at the same time. This explanation is compatible with Engelhardt's explanation of "linking by a connector"<sup>247</sup> but is, of course, not at all compatible with Responsive Positioning as these object-to-object relations cannot constitute structured space.

<sup>247</sup> Engelhardt (2002), page 40

2. The nodes of the graph are spatial regions and thereby are part of the spatial background. The connectors are objects which are positioned in the foreground in relation to these structures. This interpretation, while not being obvious, is somewhat more compatible with Responsive Positioning as in this case, connectors are positioned according to object-to-space relations. Still, in the semantics described in this thesis, I restricted Responsive Positioning to point locators which means that every object is bound to exactly one position in a structured space. A connector which connects two regions would, of course, have to be connected to two points in space at the same time.

So while general graph structures cannot be adapted to Responsive Positioning as defined here, there is a solution for the graph in figure 81 as this graph is bipartite, which means that there are two different types of nodes. All the nodes on the left represent beverages, those on the right represent ingredients. Connections can only be made between beverages and ingredients. There are no connections between ingredients, neither is it possible to connect beverages with each other. The connections of such a mapping can be expressed in an adjacency matrix like in figure 82, where the nodes of the beverages are translated into values of the beverage attribute, and the ingredient nodes are translated into values of the ingredient attribute. Each connector can be transformed into exactly one object which is bound to only one position in the matrix thereby getting assigned both attribute values. Such a structure, which is a simple matrix, is compatible with the Responsive Positioning semantics.

	Espresso Beans	Coffee Beans	Sugar	Milk	Cocoa
Pot Black		●			
Black		●			
Standard		●	●	●	
White		●		●	
Cream Cocoa				●	●
Cocoa Drink					●
Espresso	●				
Double Espresso	●				
Sweet Espresso	●		●		
Cappuccino	●			●	
Cocoapresso	●				●

Figure 82: Structure representing the connections of figure 78 using unreal objects describing connections in an adjacency matrix

I call objects like the connector objects in figure 82 *unreal objects* because they are no genuine objects of the problem domain. When speaking about a coffee machine, one speaks about the beverages and the ingredients so in general, these aspects would be candidates for application objects. One normally would not speak about “the object that makes cappuccino contain milk”, but that is exactly what the objects in figure 82 have to be interpreted like. When using unreal objects, the world is kind of flipped. The actual domain objects become properties of a spatial structure, and the objects in the action and perception space on the screen define the relations between those properties.

### 6.3.9 Views on Different Object Ensembles

If one wanted to have the coffee machine configurator to provide both the fluid amount configuration view of figure 79 as well as the ingredient configurator using the unreal objects of figure 82, the definition of a view (see chapter 5.5) must be extended, as those two views while semantically referring to the same affairs, regarding Responsive Positioning do not refer to the same set of objects. In figure 79 beverages are objects, while in figure 82 beverages are only represented as attributes of a connector object. Their configuration can thus only be determined indirectly by deriving it from the configurations of the connector objects.

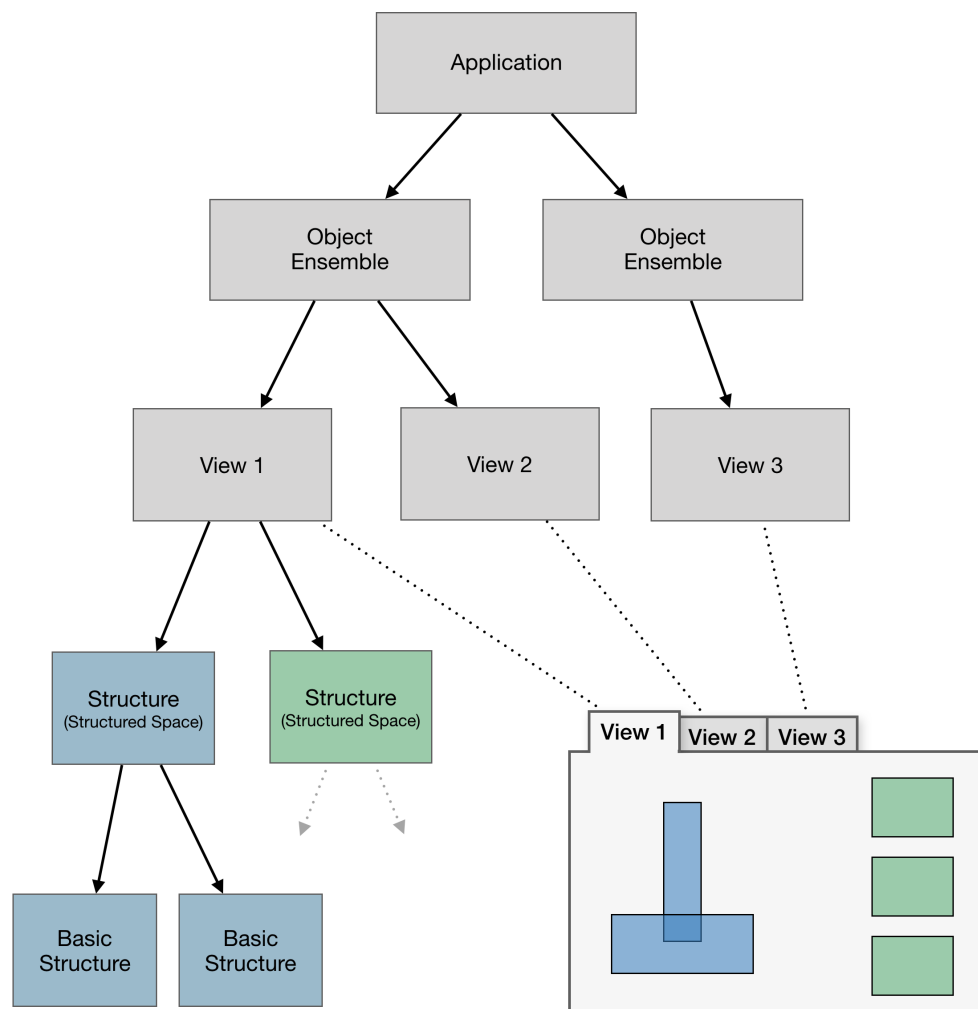


Figure 83: The relations between an application, its attribute ensembles, its views, its sections, and its structures



In order to allow both views to be part of the same application as well as in order to generally allow application views to refer to different object ensembles, I extend the concept of views so that they can refer to the same object ensemble but can also refer to another set of objects provided by the application. Taking this extension into account, I can now give a comprehensive definition of the relations between an application, its attribute ensembles, its views, its sections, and structures, as depicted in figure 83.

- Applications can provide different sets of object ensembles which can be made accessible in different views. In figure 83 the application provides two distinct object ensembles.
- For the user, the user interface of an application consists of one or more views which are used to manipulate the objects of the object ensembles provided by an application. Two of the views in figure 83 refer to the same ensemble, while the second ensemble has a view of its own.
- The application, especially if it consists of more than one view, needs additional user interface elements which makes the views available to the user by arranging them on the screen or by allowing the user to switch between them.
- A view provides a structured background in relation to which the object ensemble is presented and can be manipulated. The configuration of a view may include a definition of manipulation restrictions which allow feedback to manipulations while they are taking place.
- The structured background of each view consists of one or more structures in different spatial sections. The structures provide different evaluation contexts for the application. If there is more than one structure, the space between them is considered to be neutral space. When having several views working on the same attribute ensemble, each view has to refer to its own context attributes or the views containing the same objects have to contain the same sections.
- Each structure is created through the arrangement and overlay of one or more basic structures. This combination must be suitable for Responsive

Positioning, which means that basic structures may not be conflicting (see chapter 5).

### 6.3.10 Configurable Views

In order to be able to effectively use Responsive Positioning as a user interface for an application, one of the key tasks is, of course, the creation of well-structured spaces. If that step is done badly, the user can end up with inoperable user interfaces, which means that the necessary inputs for an application cannot be made and the output of this application cannot be perceived.

In the examples explained so far, it should have become clear that the structured space of a view often cannot be fully designed in advance, meaning there cannot be someone who “paints” the views which could then, without change, be used in Responsive Positioning applications. Those views would be static and would not be able to adapt to how the user of an application needs it to be like.

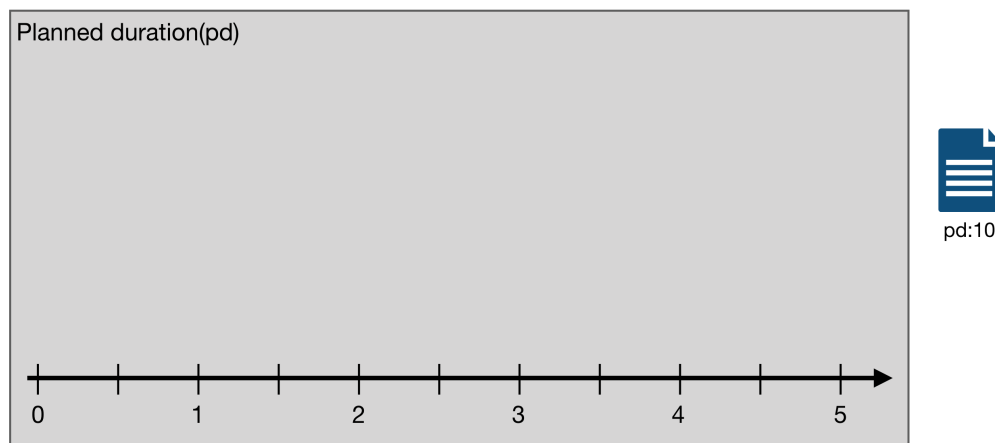


Figure 84: A structure which cannot place the object

Figure 84 shows what could happen in such a case. It shows a structure which could be a part of an application for project planning. Projects are positioned on an axis which corresponds to the number of person-days planned for the project. In this case though, there is a problem with the user interface. The object on the right has a ‘pd’ (planned duration) value of 10, but the axis only allows values from 0 to 5. The structure is not able to visualise the application state properly as it cannot place the object. The reason for such an inoperable interfaces is a discrepancy between what is modelled in the application logic and the

configuration of the structure. While at the beginning of the chapter I described application logic and spatial structure as being independent, there must, of course, be a match between them. Application and spatial structure are independent in so far as the correspondence logic within the structure has no notion of the domain model and vice versa so each component can be developed, tested and run without the other but if they are to be used together, one controlling the other and one being the output for the other, they must fit to each other.

Figure 85 shows an example of a user interface where the configuration of the spatial structure does not reflect the attribute values provided by the application, as the user interface designer was not able to assume the necessary details in advance. Someone used a database of technical inventions in order to search for Hypertext systems and is provided with a timeline which should put the resulting objects into context. The user interface designer of this application decided to let the timeline run from 1960 to 2000, maybe because initially all objects in the database were from that timespan. In the depicted case, the application created a result object, the ‘Memex’ object, which cannot be positioned using this structure as the Memex was invented in 1945, which is before the lower limit of 1960. This is not the only problem with this structure. The decision to restrict the upper boundary to 2000 may have been even less intelligent as any new technical development put into the database, of course, has a later invention year than 2000.

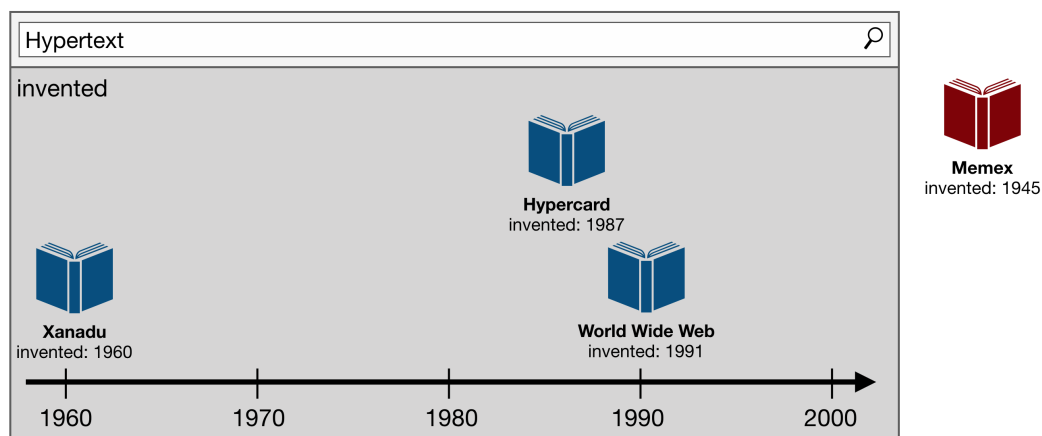


Figure 85: A search view which created a result object which is unplaceable by the spatial structure provided.

The issue in figure 85 is not that structured backgrounds can be created badly. Of course, every user interface technique can be implemented badly. It should rather

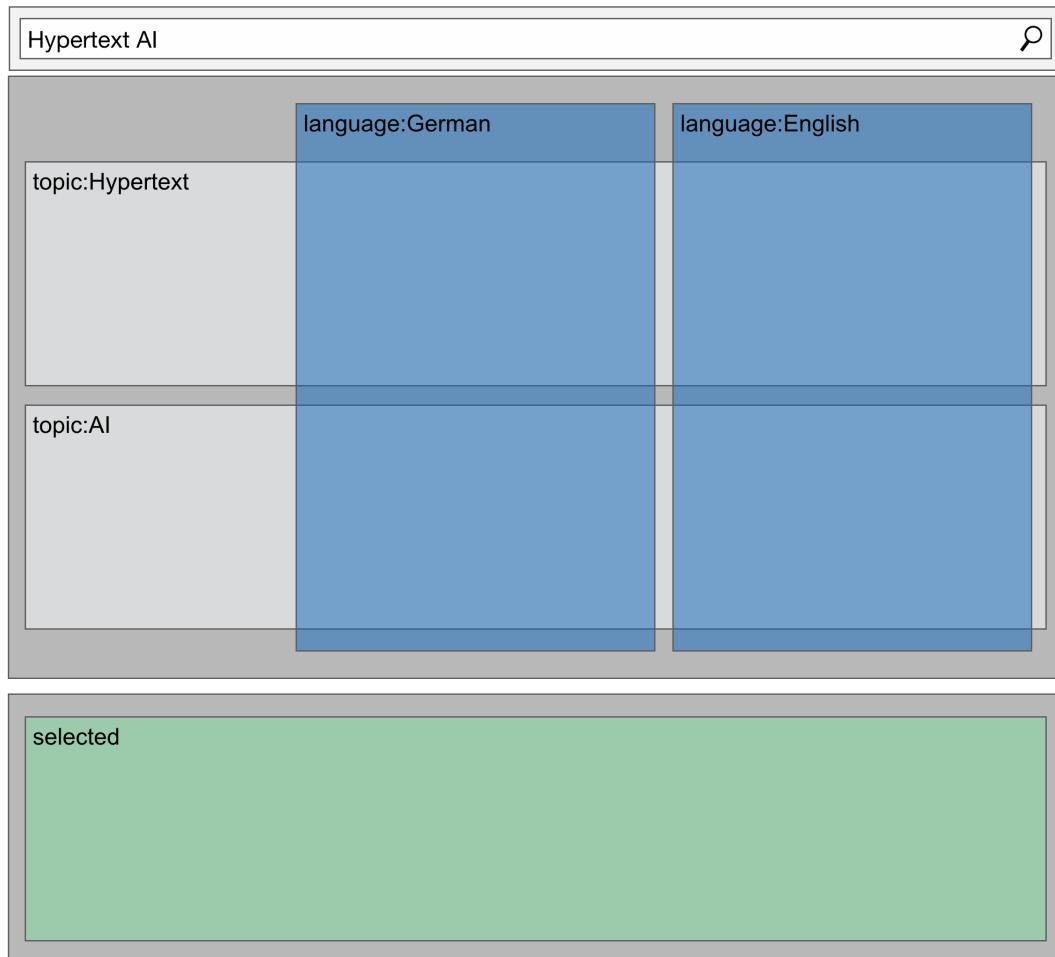
point to the problem that not every aspect of a structured background can be known in advance as providing the best possible structure for a given situation can depend on the status of the application. The best solution for the problem in figure 85 would be to configure the axis in a way that the earliest year would be either the earliest year of the results or of the earliest element in the whole database. The same is true for the upper boundary of the axis, which could, of course, as well always be set to the current year as no device can have been developed in the future. In order to provide such a flexibility within a background structure, it must be possible to configure spatial backgrounds in such a way the view can render them according to the conditions of the application state. Such a solution requires applications to be able to set the configuration of structures or needs structures to be able to gather their configuration by accessing the application model. This aspect will be covered in chapter 6.4.

### **A Draft for a Template Engine for Structured Spaces**

Figure 86 shows a view of an application which, among other things, allows users to query a database for documents by topic and by language. Its topmost element is a search bar, which is a classic UI element, which I call a *foreign object*. As mentioned before, *foreign objects* are elements of the object ensemble of a Responsive Positioning view which neither are application objects nor part of the structured background or mere decoration. Below the search bar, there is a section which represents the search results. At the bottom of the view, there is a section which consists of a single element. Users can drag objects from the result area into this area in order to assign them to their personal document selection.

In contrast to classic user interfaces, where the result area for a search might just be a generic list view which is then filled by the application, in this scenario, the structure of the space itself has to change. New structures have to be created or removed according to the number of topics which have been searched for. Those structures which are present have to be configured so that they meet the criteria of the search. This means the structure cannot be completely predesigned using purely graphical means. A possible solution which would allow a flexible background structure might be to explicitly program the creation of the

backgrounds thereby letting the applications create the spatial structures, yet this would diminish the flexibility of application development as views could not be created independently of the application logic.



*Figure 86: A Responsive Positioning view of an application whose exact configuration depends on what has been searched for*

In the following, I suggest the use of templates which describe the spatial and logical structure and provide variables according to which an actual structured space can be configured. In order to do this, the background structure must be described using a notation that can be parsed in order to create concrete instances of it. There are many user interface description languages like XUL by Mozilla and XAML by Microsoft which describe the structure of user interfaces in an

XML file. The XUL and XAML specifications<sup>248</sup> themselves are not helpful for my purposes though, as they, of course, have been designed to meet the demands of classic WIMP based user interfaces. Nevertheless an XML structure is a good candidate for creating templates. Inventing my own description language, the structure in figure 86 can be described as follows:

```
1 <view id="search_select" title="Search and Select">
2   <foreign id="search">
3     Search: <input type="text"
4               data-bind="value: searchText" />
5   </foreign>
6   <section id="results">
7     <layer id="topics">
8       <region class="topic" key="t" value="Hypertext" />
9       <region class="topic" key="t" value="AI" />
10    </layer>
11    <layer id="languages">
12      <region class="language" key="l" value="German" />
13      <region class="language" key="l" value="English" />
14    </layer>
15  </section>
16  <section id="select">
17    <region key="selected" />
18  </section>
19 </view>
```

The XML description reflects the structures which can be seen in figure 86. It consists of a foreign object (lines 2 to 5), which is the search bar, as well as two sections, one called ‘results’ (lines 6 to 15), the other called ‘select’ (line 16 to 18). For each section, the XML description specifies, which basic structures it consists of. The ‘select’ section consists of only one region (line 17) whereas the ‘results’ section consists of a total of four regions. For Responsive Positioning, an XML specification like the one given here cannot fully explain the functionality of the user interface. It can only describe which structures exist and which sections they belong to. It does not specify how the structures appear and how they are positioned spatially. In Responsive Positioning, the visual properties of user interface elements are much more important than in other user interfaces. This is especially true for spatial properties as they define what is possible with the structure. While in a classic user interface, the layout of elements may be vital to

---

<sup>248</sup> The respective specifications can be found online under [developer.mozilla.org/de/docs/Mozilla/Tech/XUL](http://developer.mozilla.org/de/docs/Mozilla/Tech/XUL) and [docs.microsoft.com/en-us/dotnet/framework/wpf/advanced/xaml-overview-wpf](http://docs.microsoft.com/en-us/dotnet/framework/wpf/advanced/xaml-overview-wpf) (accessed October 2017)

their usability and ergonomics, the general functionality of the user interface is not affected by it. In the user interface description provided here, a CSS specification is used in order to define the object placements. A CSS configuration can specify which structures are placed at which position of the screen and which structures overlap one another. To be able to create a CSS specification which is independent of the number of topics and languages, in the XML description the five structures within the ‘results’ sections are grouped by topic (lines 7 to 10) and language (lines 11 to 14) using dedicated layer tags. These layer tags makes it easier to define groups as overlapping structures and avoids the necessity to explicitly place objects by their id. The following code shows, how CSS can be used as a description of the visual and spatial properties of the structure<sup>249</sup>.

```
section {background:#ddd;}
#search {height:40px; margin-bottom:10px;}
#results {margin-bottom:10px; height: calc(100vh - 310px);}
#select {height:250px; padding:30px;}
#topics {
    position:absolute;
    top:150px; left:30px; right:30px; bottom:300px;
    display:grid; grid-row-gap: 20px;
}
#languages {
    position:absolute;
    top:90px; left:230px; right:50px; bottom:280px;
    display:grid; grid-auto-flow: column; grid-column-gap: 20px;
}

region {border:1px solid black;}
#topics region {background:rgba(0%,0%,100%,0.2);}
#languages region {background:rgba(100%,0%,0%,0.2);}
#select region {background:rgba(0%,100%,0%,0.3); height:100%;}
```

According to this CSS specification, the topic and language regions overlap. Their layers are positioned on top of each other using the absolute positioning capabilities of CSS definitions. The regions within their layers are configured using CSS grid layouts which in this case means, they are dividing the available space among themselves and resize themselves accordingly. As a consequence of the specification, languages appear as columns whereas topics appear as rows.

Due to the use of the grid layout, the CSS specification is formulated in a way it would work with any number of topics or languages as the size of the regions and

---

<sup>249</sup> I removed some of the colour and border specifications in order to keep the definition simple

their positions adapt automatically regardless of their number. The XML specification given above, though, still is static. Depending on the state of the application, which means depending on what has been searched for, the XML structure must contain more or fewer structures with differing configurations, so the XML document must be modified in a way that a parser can automatically create individual specifications from it. The syntax of the specification below is based on knockout.js<sup>250</sup> using which one is able to specify a data binding between different sections of the XML specification<sup>251</sup>.

```
1 <view id="search_select" title="Search and Select">
2   <foreign id="search">
3     Search: <input type="text"
4               data-bind="value: searchText" />
5   </foreign>
6   <section id="results">
7     <layer id="topics" data-bind="foreach: searchArray">
8       <region class="topic" key="t" data-bind="value: $data" />
9     </layer>
10    <layer id="languages">
11      <region class="language" key="l" value="German" />
12      <region class="language" key="l" value="English" />
13    </layer>
14  </section>
15  <section id="select">
16    <region key="selected" />
17  </section>
18 </view>
```

The modified XML file binds the input provided by the search bar (lines 3 and 4) to the configuration of the topics layer. Internally<sup>252</sup> the entered text is converted into an array. For each of the words provided as the input, in line 8 a region is created. This way, background structures for applications based on Responsive Positioning can be described in a way that they adapt themselves to the current needs of the applications. In this case, they change the number and the

---

250 The knockout.js specification can be found under [knockoutjs.com](http://knockoutjs.com) (accessed October 2017)

251 knockout.js, by the way, makes use of the Model-View-ViewModel pattern which I describe in chapter 6.4. In contrast to classical template engines where a template file is explicitly parsed resulting in a concrete HTML or XML file, frameworks like knockout.js use data binding which means the structure changes dynamically according to a data model without having to explicitly trigger a parser.

252 The conversion code is omitted here.



configuration of the topic structures according to the number and the configuration of the topics which have been selected for it<sup>253</sup>.

## 6.4 Model-View-ResponsivePositioningModel

Responsive Positioning views, as described in this chapter, are views in the sense of Reenskaug's original definition of Model-View-Controller as in Responsive Positioning, views contain the necessary logic which translates spatial manipulations of its application objects into changes on the application level. I have shown that these views do not only have to contain their own evaluation logic in the form of structured background but also that this evaluation logic must, at least partially, be configurable as it semantically depends on the state of the application logic. In order to avoid an architecture where the configuration of views is intertwined with the application logic, an MVC like structure is necessary, yet as described in chapter 6.2, many MVC interpretations are not compatible with Responsive Positioning as they do not provide a clear location for the Responsive Positioning logic.

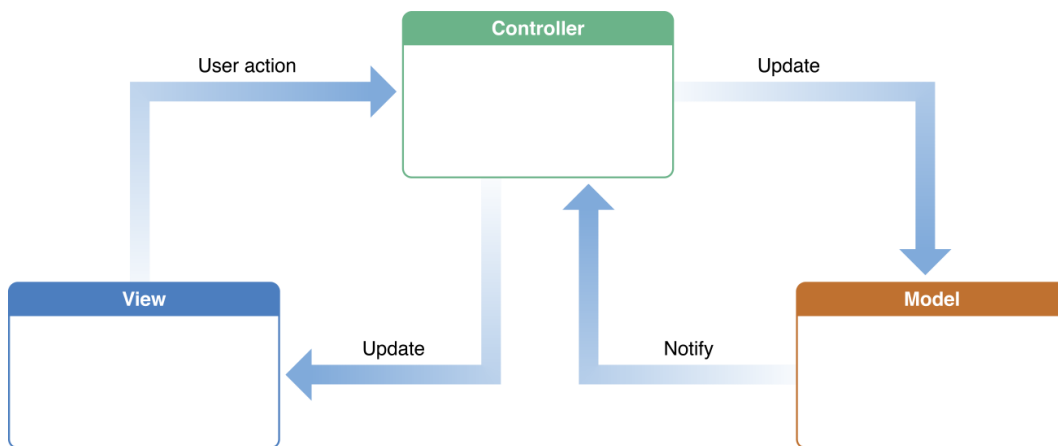


Figure 87: The Model-View-Controller interpretation in Apple's Cocoa framework (taken from Apple 2015)

A possible solution could be Apple's version of Model-View-Controller (see figure 87). In contrast to the common MVC interpretation described by Eckstein

<sup>253</sup> The way of defining structured backgrounds proposed here has the great disadvantage of being inherently non-visual. A way of creating background structures visibly yet keeping the flexibility of the described XML/CSS-specification would certainly be desirable but has not been investigated further in this thesis.

(2007), see figure 69 in chapter 6.2, there is no direct connection between a model and the view, but the controller takes the role of a mediator between the two. While Apple's describes this kind of controller as a "conduit" which suggests that changes from the view are only passed through to the model and vice versa, this MVC interpretation is technically compatible with Responsive Positioning, as it would allow the Responsive Positioning logic to be implemented in the controller. In contrast to Reenskaug, where the views themselves "know" how to interpret manipulations to them in relation to the application model, in this concept a controller would take the function of a translator between object manipulation and object rendering in the view on the one hand and the application model on the other hand.

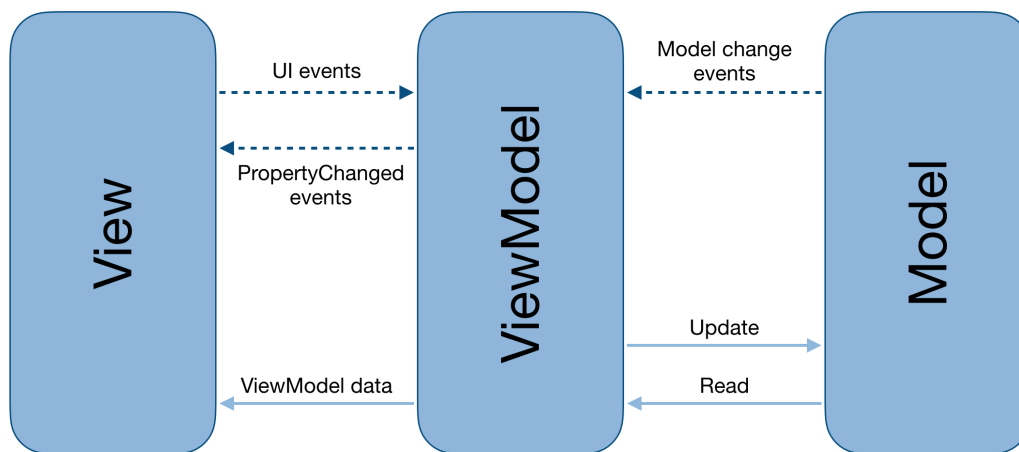


Figure 88: The Model-View-ViewModel pattern (redrawn from Microsoft 2010)

Microsoft describes a similar solution in their Model-View-ViewModel (MVVM) pattern<sup>254</sup>. In MVVM, schematically depicted in figure 88, the controller has been transformed into a ViewModel which, as the name says, is a model for the view. While technically quite similar to Apple's MVC interpretation, MVVM emphasises on a ViewModel which is a model which is independent of the application model. As a consequence, the view in the MVVM pattern is not a rendering of the application model but rather a rendering of the ViewModel. The ViewModel processes changes made to the view and updates the (application) model if necessary. This model, in turn, informs all attached ViewModels about changes in the model. The ViewModel can then fetch these changes, process them

<sup>254</sup> see Smith (2009), Microsoft (2010)

and update the view if necessary. A ViewModel, of course, does not have to be passive but can implement its own logic. It could, for example, implement a ResponsivePositioning logic.

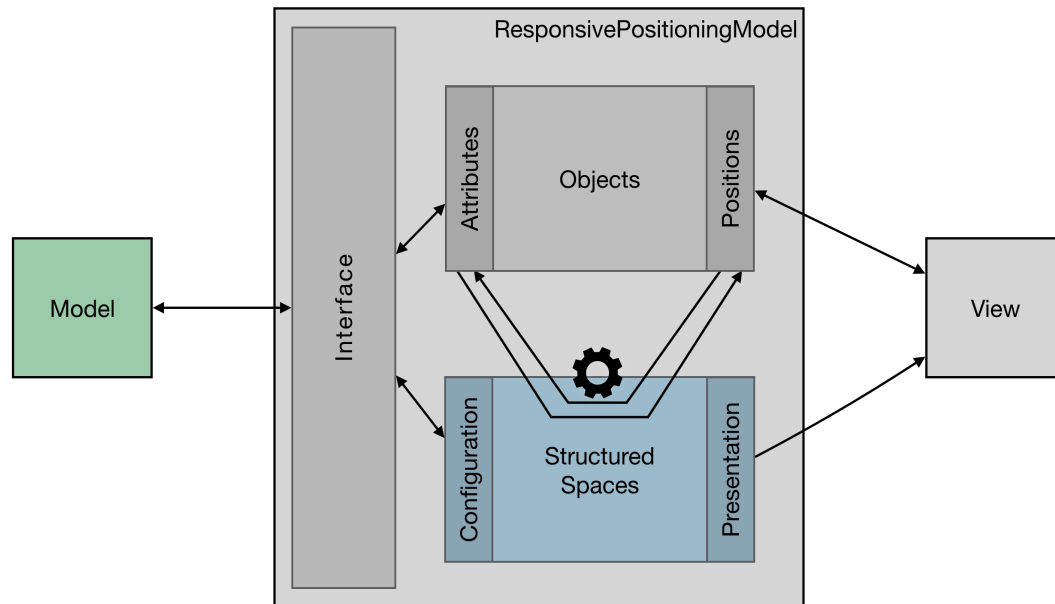


Figure 89: Model-View-ResponsivePositioningModel (MVRPM)

Figure 89 shows an adaption of the MVVM pattern which I call Model-View-ResponsivePositioningModel (MVRPM). The model, as in all MVC variations, encapsulates the application logic. The view renders both the spatial structure as well as the application objects. Both of them are stored within the ResponsivePositioningModel (RPM) which contains the Responsive Positioning logic as well as all necessary data about the objects and the background structures of a Responsive Positioning user interface. The view can only access the presentational properties. In the MVRPM pattern, the application model does not directly react to changes in the view nor can it directly change attributes of the objects within the ViewModel. As the RPM is a model of its own, the objects and the structures only exist within this model and its rendering in the view. Just as the ViewModel in MVVM, the ResponsivePositioningModel processes those data provided by the model and creates its internal data, which in this case represents the objects and the structures. In case of a change in an object attribute, the RPM processes this change and performs the appropriate model actions if necessary, so in the ResponsivePositioningModel, all connections between presentation data in

the PRM and application data in the model are channelled and processed through an interface.

The MVRPM pattern described above not only allows applications to interact with object attributes but, as motivated before, also allows the structured spaces to be configurable.

### **6.5 WebArena – An Implementation and New Potentials**

Great parts of the Responsive Positioning semantics described in chapters 5 and 6 have been prototypically implemented<sup>255</sup> in a web-based software called WebArena which I developed in the course of creating this thesis.

The concepts behind WebArena are based on two main factors of influence. On the one hand, WebArena is based on many of the ideas of a knowledge work concept called virtual knowledge spaces while, on the other hand, WebArena was explicitly created to meet the spatial requirements of an experimental course concept which has been tested at Paderborn University. By sketching both concepts, it will become clear that the concepts very much influence each other and that both concepts influenced the concept of Responsive Positioning.

Further on, I motivate how the combination of concepts like dynamically attributable objects (from the knowledge work concept), the possibility of being able to dynamically create structured backgrounds (from the course concept) and the evaluation potentials provided by Responsive Positioning allow for new ways of thinking about the user interface, which would not be possible if the concepts were only considered separately. In an exemplary scenario, I motivate, how Responsive Positioning within a WebArena allows for user interfaces which are not pre-constructed, like in the examples provided so far, but in which users can dynamically create the interface when they need them and for as long as they need them.

---

<sup>255</sup> See Sens (2015)

### 6.5.1 Virtual Knowledge Spaces

Many of the concepts behind WebArena are based on the concepts of *virtual knowledge spaces* as defined by Hampel (2001)<sup>256</sup> who described them as virtual locations on the network in which knowledge work can take place. The concept is quite extensive, so the following description is highly selective and focuses on those parts of the concept which WebArena is based on and which are interesting in terms of positioning techniques<sup>257</sup>. Hampel based his knowledge space concept on the ideas of Multi-User Domains (MUDs) and Multi-User Object-Oriented Environments<sup>258</sup> (MOOs). These virtual environments started as text-based games but over time developed into online meeting places. According to Curtis (1992), they can be described as “a kind of virtual reality”.

“A MUD is a software program that accepts ‘connections’ from multiple users across some kind of network (e.g., telephone lines or the Internet) and provides to each user access to a shared database of ‘rooms’, ‘exits’, and other objects. Each user browses and manipulates this database from ‘inside’ one of those rooms, seeing only those objects that are in the same room and moving from room to room mostly via the exits that connect them. A MUD, therefore, is a kind of virtual reality, an electronically-represented ‘place’ that users can visit.”

Based on this “shared database of ‘rooms’, ‘exits’ and other objects”, Hampel defined rooms to be the key concept for providing an internet-based cooperative learning environment.

“The room structure enables positions to be defined for objects, representing people, documents, tools and services that allow a knowledge space to be cooperatively established and, at the same time, responsibilities, rights, or competences to be mapped

---

256 See also Hampel&Keil-Slawik (2001)

257 An important aspect of the concept which is not covered here, for example, is a rights management which allows the self-administration among knowledge workers (see Hampel&Keil-Slawik (2001) Section 6.4).

258 Hampel&Keil-Slawik (2001), section 5.1

structurally. To this extent, rooms constitute a key metaphor for virtual learning communities.”<sup>259</sup>

Hampel’s virtual rooms<sup>260</sup>, of course, were not to be understood as photorealistic simulations of real-world rooms, but rather have to be interpreted metaphorically, which means that some properties of a real-world room are transferred to virtual rooms, while some characteristics cannot be transferred and others are genuinely digital and thus cannot be observed in real-world rooms. Rooms in virtual knowledge spaces are described as places in which knowledge workers can find and organise their working materials as well as the place where they can get into contact with each other. Like in real-world rooms, a virtual room has an inventory which means it contains objects which, in this case, represent the artefacts of knowledge work (e.g. documents, images, weblinks, or soundbites). Gates or exits<sup>261</sup> were adopted from the MUD concept. Using a gate, a user can move from one virtual room to another one, assuming he has sufficient rights to visit this room. Quite in contrast to real-world rooms, virtual rooms can contain other virtual rooms. By featuring both subrooms as well as gates, Hampel’s virtual knowledge space concept allowed both hierarchical as well as hypertext-like connections between rooms. Many objects in virtual knowledge spaces, e.g. documents or images, have a content which can be made visible when needed. In addition to that, all objects carry a number of attributes<sup>262</sup>. Despite some attributes which have a rather technical nature, like an id, an object type and a name, object attributes are either registered specifically for an object type, such as a destination attribute for a weblink object, or can be used freely just by assigning values to them.

---

<sup>259</sup> Ibid., section 5.2

<sup>260</sup> Due to an ambiguity of the German word “Raum”, which can refer both to “room” as well as to “space”, Hampel used the word “Virtueller Wissensraum” both for the structure of interconnected, self-administrated rooms as well as for the rooms themselves. In order not to have to call the latter a “virtual knowledge room” while calling the former a “virtual knowledge space”, a term which is established in English publications about the concept, I decided just to speak about “room”, as did Hampel&Keil-Slawik (2001), or “virtual room” in order to distinguish them from real-world rooms.

<sup>261</sup> Both terms were used in different stages of the concept.

<sup>262</sup> See Hampel (2001), pages 142-144.

According to Hampel, virtual rooms can be represented in a number of ways<sup>263</sup>, so virtual rooms as spatial entities in which objects can be arranged and manipulated by dragging them around, are but one possible representation of a room. When Hampel and his colleagues put the concept of virtual knowledge spaces into realisation in form of the sTeam<sup>264</sup> server, the basic access method they thought of was a web browser accessing a web server. Web user interfaces at the beginning of the century were strictly asynchronous and lacked many of the graphical features they provide today. Users who accessed a virtual room using a web browser basically got a list view of the objects in the room. Changes to those objects remained invisible unless the user manually reloaded the page.

### 6.5.2 Visual Clients for Virtual Knowledge Spaces

An interface called COAL (Client Object Access Layer)<sup>265</sup> allowed for the development of external clients which not only allowed synchronous access to virtual knowledge spaces but also made a spatial access to virtual rooms possible. Büse (2001) and Niehus (2006) created Java-based clients which allowed the spatial arrangement of objects within virtual rooms (see figure 90)<sup>266</sup>. In order to not only be able to arrange the existing objects, but also to provide a graphical environment for them, Büse and Niehus extended the list of possible object types by graphical elements such as rectangles, circles, arrows as well as text elements which could be used as captions or for textual explanations of what can be seen in the room. This allowed knowledge workers to create graphical structures and arrange documents, rooms, exits, and other objects according to it.

Through the graphical clients, Hampel's concepts of virtual spaces whose declared purpose was to provide virtual places where collaborative knowledge work can take place, was extended by features which allowed virtual rooms to be places not

263 He explicitly mentioned "two-dimensional, three-dimensional, as a tree structure, or a textual description" (Ibid.).

264 Hampel (2001), pages 189-240

265 Hampel&Keil-Slawik (2001), section 6.3

266 While being quite similar in their basic functionality, they differed in the technologies they used. Büse created a software called *sTeam shared whiteboard* which could be embedded into a website as a Java Applet, which at the time was a common technology used in order to overcome the drawbacks of the contemporary web technology. Niehus instead focused on modern rich client technology and thereby decided to use Eclipse frameworks for his implementation (Niehus 2006, page 24ff), which was initially called *Mediarena* but later was renamed to *Mediarena Composer*.

only for the organisation but also for the presentation of knowledge. This new potential allowed for a new experimental teaching scenario called the *Jour Fixe Concept*<sup>267</sup> in which, instead of preparing for a written or oral exam on the contents of a lecture, participants of a course had to prepare and present a graphical knowledge space for the topic assigned to them. For doing so, they had to arrange objects in such a way it allowed a visitor to find relevant information about the given topic by exploring the spatial virtual room. The knowledge structures thus resembled a kind of exhibition hall covering a given topic.

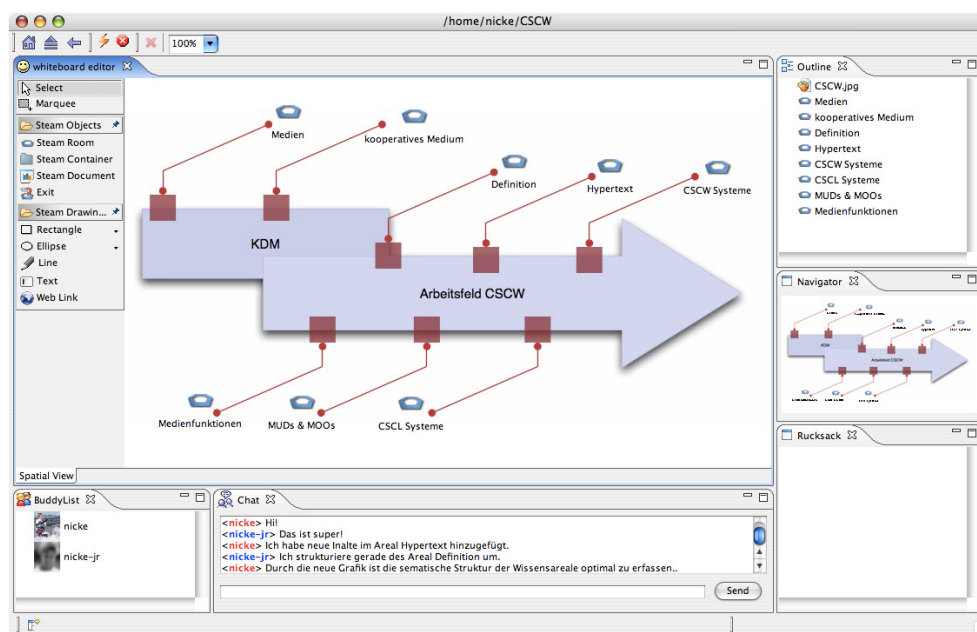


Figure 90: A graphical representation of a virtual room of a course in the Mediarena Composer (image taken from Niehus 2006)

### 6.5.3 WebArena

While WebArena, in great parts, is a reimplementation of the aforementioned concepts and clients, it also is a reinterpretation. While in the software described above, graphical space was used in order to provide a common place for the organisation of knowledge work as well as for the presentation of knowledge work results, the focus of the WebArena lies on using the spatiality of a virtual room as a knowledge work tool itself. While WebArena is not technically based

<sup>267</sup> See Hampel et al. (2003). The name Jour Fixe was chosen because the process included several meetings in which the working groups presented the current state of their work to the rest of the course in order to get responses early on.



on the sTeam server, both share some core ideas of Hampel's virtual knowledge space concepts. Just as described by Hampel, WebArena uses a room metaphor. Real-world rooms and WebArena rooms share a number of common characteristics which differed somewhat from the characteristics of virtual rooms in Hampel's virtual knowledge spaces.

*Spatiality:* In WebArena a room always consists of a 2-dimensional space in which objects can be arranged, so a room is not only a metaphorical place for objects but provides the non-metaphorical space in which objects can be arranged.

*Separation:* Virtual rooms are separated from each other. Every object is located either in one room or in another room. While the standard WebArena client always shows one room at a time, but allows users to switch between them by changing tabs, experimental versions of WebArena can display more than one room at a time and allow objects to be dragged back and forth between them<sup>268</sup>.

*Purpose and Structure:* An important characteristic of WebArena is that different rooms serve different purposes. The purpose finds its expression in the interior design of a room. Rooms thus are not only containers for their inventory but can provide an interior design which gives them structure and makes the object positions inside of them meaningful. This functionality is provided by being able to define an explicit background structure for a room in relation to which the inventory of the room can be positioned.

Figure 91 shows the WebArena software as it appears in a current web browser. The black bar at the top contains icons which can be used to create new objects or structures, reach the environment of the current room, log out of the system, etc. The grey panel on the right-hand side of the figure is an object inspector. It allows the users to edit registered attributes of the currently selected objects. It also contains a button to switch between the mode in which the background of a room can be edited and the mode in which only the foreground objects are editable. The main content part on the left shows a structured space consisting of two regions and one axis which provides a structure for two pdf documents.

---

<sup>268</sup> See Petertonkoker (2013).

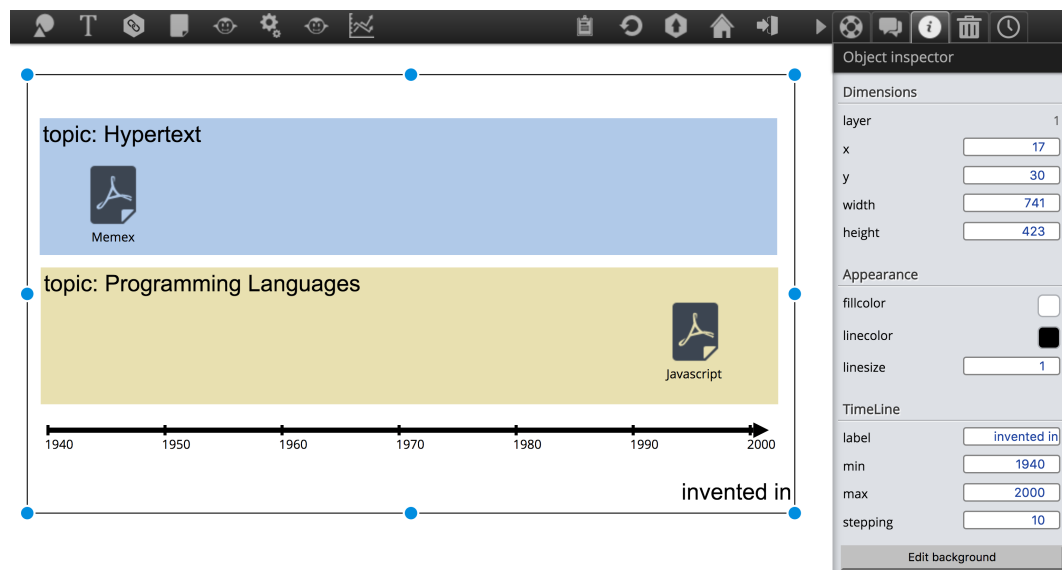


Figure 91: WebArena showing a room structured by an axis and two regions

WebArena's focus on using the spatiality of a room as a tool of expression or as a means of differential experience was the result of an evolution of the aforementioned Jour Fixe concept. A new concept called MediaThing<sup>269</sup> kept some of the characteristics of the Jour Fixe concept, such as regular meetings and the final goal of creating a virtual room, yet instead of asking the students to provide a virtual knowledge space using which a given topic can be explored to its full extent, students were taught to formulate a research hypothesis regarding their topic which they then had to verify or falsify by placing knowledge artefacts in relation to a well-designed structured space which reflected their hypothesis. While the WebArena version used for the experimental courses so far did not allow any automated evaluations, students were nevertheless expected to apply positioning semantics similar to those of Responsive Positioning while creating their evaluation structures as well as while positioning their knowledge objects in relation to these structures. This means that students were required to be able to justify the positions of the objects in relation to the structured space.

Early runs of the course used the visual clients for visual knowledge spaces described before. When they were reimplemented as a web-based software instead of a Java client, the WebArena was tailored to the needs of the new scenario, e.g.

<sup>269</sup> See Keil (2009) and Jakoblew et al. (2015). The word "thing" in this context does not refer to an object but to an old Northern European form of assembly. It is pronounced like "ting".

by allowing objects to be fixed to the background, thereby making them, at least in some aspects, a background structure. In later versions, the software was extended by matrix and axis structures and with explicit foreground and background modes which allow users to edit the background independent of the foreground when in background mode, and to experience the background as a coherent, static structured space when in foreground mode.

Sens (2015) used WebArena to prototypically implement Responsive Positioning (see the example in figure 91). His implementation relies on a feature which WebArena inherits from Hampel's virtual knowledge spaces. Like in Hampel's virtual rooms, objects in a WebArena room can carry a vast number of properties. Within WebArena every object provides an id, a name, an object type and a spatial position. Depending on the type of the object, of course, more attributes are necessary. A graphical square, for example, also carries at least width, height, fill colour and line colour. The important aspect for Responsive Positioning is that objects are freely attributable, which means that any attribute can be assigned to any object at runtime regardless of whether or not it was defined before. This feature is a necessary prerequisite for using WebArena as a platform for Responsive Positioning, as in Responsive Positioning objects gain attributes which are not specified for these objects, but which result from their positions in relation to a structured background, which cannot have been known at the design time of an object.

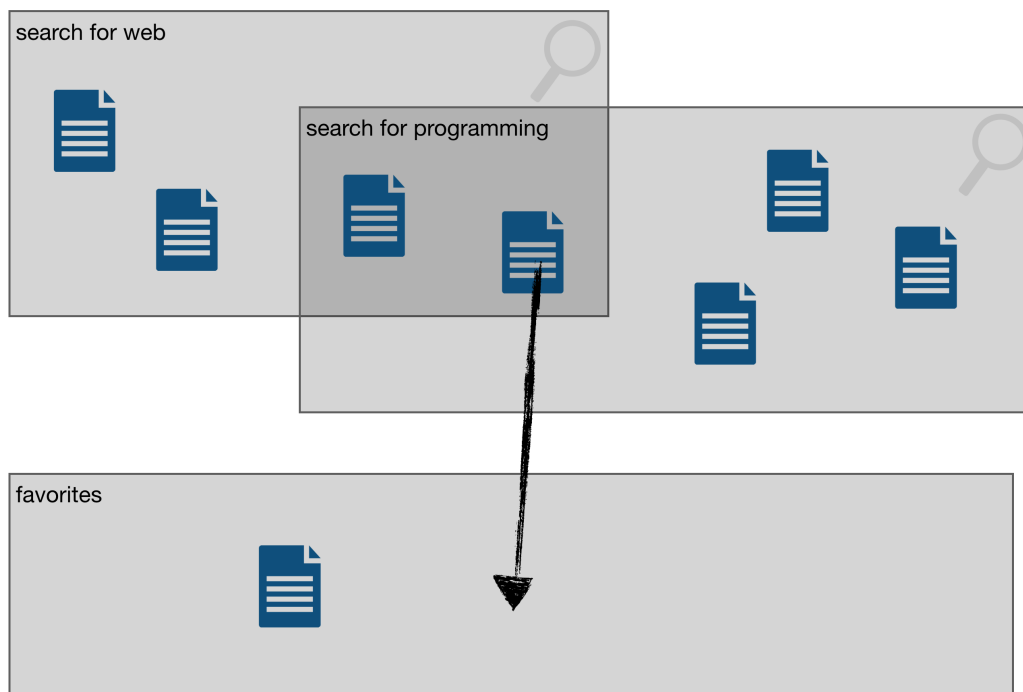
#### **6.5.4 Responsive Positioning in the WebArena Scenario**

Having a Responsive Positioning implementation in WebArena allows positions against a structured background to result in attribute assignments. Making changes on a background or moving an object into another room which contains a structured background of its own resembles a switch in the evaluation context of that attribute. As current experimental versions of WebArena do not only provide Responsive Positioning but are also equipped with an application layer in which applications can access and manipulate object attributes and are triggered when object attributes change, WebArena can also be used as a testing ground for Responsive Positioning applications.

Apart from such test cases, the combination of the flexibility of objects which can be freely attributed (a characteristic inherited from virtual knowledge spaces), the

own potential of allowing for background structures which can be easily created and manipulated by the users of the software, and an application layer which allows applications to access object attributes as well as create or remove objects, facilitates a new way of thinking about spatial user interfaces which goes beyond what has been described so far. The examples I gave so far have shown that Responsive Positioning can be used as a user interface for a number of applications. As different as their graphical structures are, what they all have in common is that their graphical structure is predetermined to a great extent by the application itself. Users of the applications can place objects in relation to the structures or can influence the structures, e.g. by specifying different queries, but they never directly create or modify the structured backgrounds themselves.

The combination of free attribution, manipulable structured backgrounds and applications working on object attributes and structure properties allows for scenarios in which users can create user interfaces dynamically. One possible scenario could be using applications as connectors to external data sources. Users could, for example, dynamically create background structures which represent a query request for a database application running in the background.



*Figure 92: A database search defined by specialised background structures*

Figure 92 shows an example of how a WebArena could be used to access a document database. Assume this database to be a pool of knowledge work material which contains a number of documents which are tagged according to their topics. Within the database, each of the documents can have multiple tags attached to them. In order to trigger a database search within the database, in the scenario depicted in figure 92 a knowledge worker created a structured background which consists of structures which the application can interpret as input for a search query. The two elements at the top of the background in figure 92 are supposed to be specialised structures used for database access<sup>270</sup>. In terms of the Responsive Positioning semantics, they work like regions of which one refers to a boolean ‘web’ attribute while the other refers to the ‘programming’ attribute respectively. After creating such a structure, an application running in the background is triggered to perform the respective search, so objects which fit to the query would automatically appear. Users can use such a room in order to have a look into the documents and drag those that interest them into the ‘favourite’ area. When finished, they can either delete the search structures or modify them in order to trigger another search. When doing that, the application logic, of course, not only has to create new objects but also has to remove those objects from the room which still happen to be within the scope of the two regions.

Figure 93 shows a similar room structure, yet this time only one of the structures depicted invokes a search while the other two are mere “setters”, so the structured space depicted can be used to query the database for documents concerning ‘programming’ in order change or extend their tagging. After the dragging operation, the leftmost document is tagged with ‘programming’, ‘web’ and ‘artificial intelligence’, the one in the middle is tagged with ‘programming’ and ‘artificial intelligence’ and the rightmost one is tagged with ‘artificial intelligence’ only, thereby losing its ‘programming’ tag.

---

<sup>270</sup> In WebArena, the elements which constitute the structured background are themselves objects, so the structure element can carry additional attributes which specify their respective roles for the application.

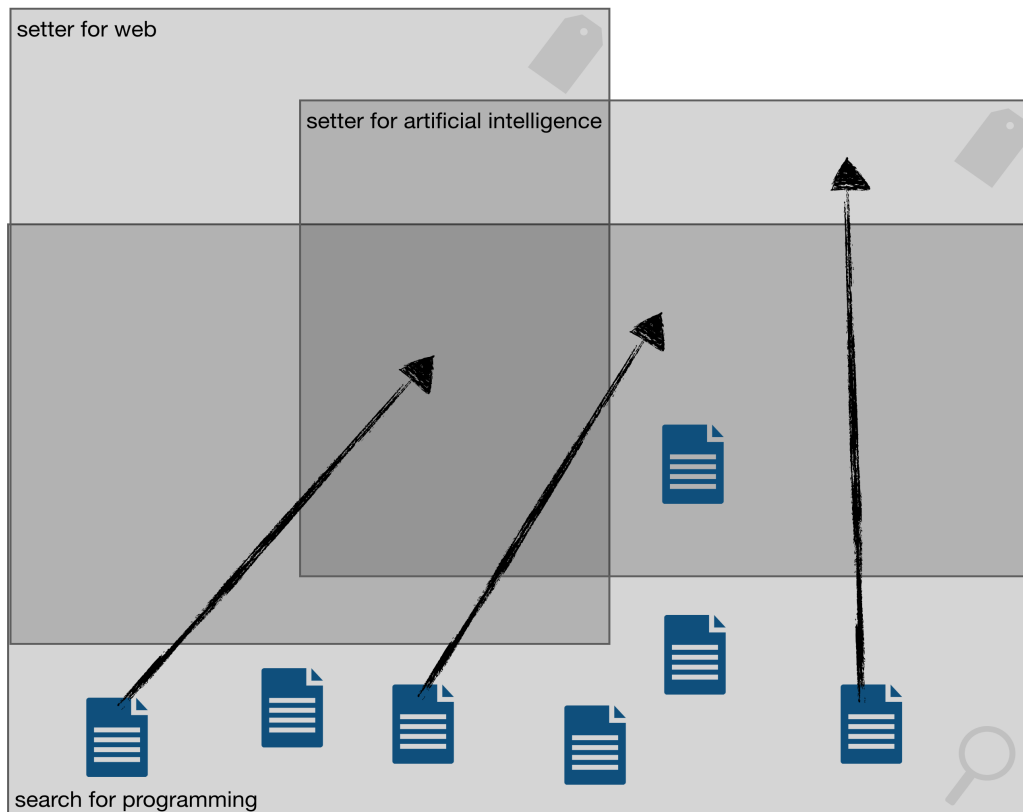


Figure 93: Extended database operations using WebArena

The structures shown here are just two simple examples for one possible application which should show that in a WebArena the interactivity of a user interface can be understood on a different level than in classical user interfaces or even in those applications explained in chapter 6.3. Instead of being “just” an interface for an application, in WebArena the interface can itself be built interactively according to one’s needs, making the interfaces themselves be subject to user manipulation. The scenario I just described is only an idea at the moment. It has not been implemented and its potentials for dynamic user interfaces in knowledge work contexts and beyond have not yet been looked into. Performing such an investigation is one of the many possibilities of future work based on the concepts of this thesis.

In this chapter, I have presented a number of applications which can use Responsive Positioning as their user interface. The exemplary applications were used not only as a proof of concept but also to further extend the concepts of the

technique. One major challenge which can be identified when creating Responsive Positioning user interfaces is that of identifying objects and properties in the application domain. The example of alternative interpretations for the multimedia control application in this regard has shown that, depending on which aspects of an application are perceived as manipulable objects and which aspect are properties, different structured backgrounds have to be created and thus a completely different interface logic results. Strategies on how an application domain can be analysed in order to find objects and structured backgrounds have to be created, as they do not exist as of yet. They remain as a task for future research on the user interface technique.





## 7 Summary and Conclusions

In this thesis I defined, explained and substantiated Responsive Positioning as a new user interface paradigm in which object attributes can be manipulated by manipulating object positions in relation to a purposefully structured space.

As a first step of deducing the potential for a new user interface paradigm based on spatial structures, in *chapter 2* I systematically determined the technical potentials of interactive user interfaces. A systematic derivation of these potentials does not only enable the description of current user interfaces from a technical perspective but also provides a theoretical foundation on which a hypothesis for new user interface potentials can be based. After considering existing definitions of the concept of interactivity and having to recognise that these definitions do not provide the necessary technical perspective, I chose to take my own approach at determining the technical potentials by considering milestones in the history of user interfaces. By describing these milestones, I could determine which technical developments were necessary to make them possible and could thereby describe their technical potentials regarding interactive technology. When assessing current user interfaces against this framework of technical potentials, on the one hand, one can recognise that the role of spatiality increased since the early command-line-based interfaces which already provided virtual objects, but on the other hand, it becomes clear that spatial positions, especially positions in relation to structured spaces, have not systematically been considered as relevant means of user input or output.

The use of spatial properties in current interactive user interfaces is compared with the usage of space as a means of differential experience in knowledge work scenarios in *chapter 3*. After defining differential experience and means of differential experience as key concepts of knowledge work, I examined the role of media for knowledge work processes. I argued that structuring one's environment, and thus using space, is a necessary prerequisite for differential experience in general and for knowledge work in particular. In different knowledge work context, different qualities of spaces played a role. In some cases, space is a mere medium which is used to arrange objects in order to allow their comparison or having them in view. In other contexts, space allows artefacts to be meaningfully

positioned in relation to each other. In this case, positions in space become meaningful while space as a medium, as a background, still remains unstructured. Ultimately, space can also be structured in itself. Artefacts positioned within such a space gain their meaning by their position in relation to the embedded structure. In the context of this thesis, I focused on the latter role of space, namely the provision of a structure for objects placed in relation to it, as it can be successfully related to the requirements of user interfaces. Relating structured spaces in knowledge work to the requirements of user interfaces allowed for the formulation of the hypothesis that positioning objects in relation to a structured space can be adapted to be a user interface technique, as the general purpose, allowing the perception of otherwise invisible object properties and providing means of manipulating these properties, is not only common in knowledge work but also is a common user interface problem.

In *chapter 4* I examined an existing theoretical foundation for describing information graphics, which are spatial structures created for the purpose of depicting semantic relations as perceivable graphical relations. I adapted a comprehensive model for such information graphics for my own definition of structured space. The distinction between object-to-object relations and object-to-space relations helped me to be able to distinguish which kinds of spatial arrangements constitute structured space and which are merely structures which expand in space using space as their medium. As part of my adaption, I identified two basic types of spatial structures, regions and axes, using which a multitude of specific structured backgrounds can be created. I defined semantic positioning as positioning objects in relation to such structured backgrounds. The way I introduced semantic positioning, it must be understood as an umbrella term for a multitude of possible evaluations which are sensible in different areas of knowledge work and beyond.

Based on the concept of structured space and semantic positioning, in *chapter 5* I developed semantics for Responsive Positioning. This semantics is tailored to meet the requirements of a user interface technique which could be derived from my investigations in chapter 2. For object positions in relation to a structured background to become a user interface, object attributes and object positions must correspond to each other not only in an unambiguous but also in a responsive manner. For the user of a Responsive Positioning user interface, the position of

the objects must at every time correspond to the attributes of the objects. This way object positions and object attributes virtually become the same. A further part of the chapter defined algorithms which describe how a correspondence between object positions and attributes can be reached in complex combined background structures. I formally determined, under which circumstances such a combined background structure can be considered a valid structured background for the Responsive Positioning technique.

In *chapter 6*, I investigated how the Responsive Positioning technique can be used as a user interface for applications. Besides a number of examples, which I used both as proof of concept, as well as to demonstrate important characteristics of Responsive Positioning user interfaces, in a great part of the chapter I investigated how user interfaces based on Responsive Positioning relate to different interpretations of the Model-View-Controller concept. I consider it to be surprising that Reenskaug's 38-year-old initial concept is one of the best fitting interpretations of Model-View-Controller for a user interface concept in which the spatial manipulations of objects in a structured view are imperative. Only modern concepts like the Model-View-ViewModel pattern provide an equally good basis as a design pattern for Responsive Positioning semantics. I thus extended Model-View-ViewModel by defining a ResponsivePositioningModel which encapsulates all the Responsive Positioning semantics and which, when implemented, should allow for a strong separation of the application logic and the user interface, a feature which in today's software development can be considered as very important. Two important sections of the chapter dealt with how structured backgrounds are defined. In a more classical approach, where the user interface is provided by the applications, these applications must be able to modify and configure aspects of structured backgrounds, for which I presented a concept. As an innovative idea, I presented the possibilities of Responsive Positioning in a context which allows the dynamic manipulation of structured backgrounds by the users and thus allows them to dynamically create and adapt their user interfaces on the fly.

This chapter, *chapter 7* mainly deals with open research questions and provides some final thoughts about the thesis and how it came about.

### 7.1 Open Research Questions

In this thesis, I developed Responsive Positioning and presented it as a possible new user interface paradigm. Every new concept and every new solution, of course, brings up many questions. In many stages of the thesis, alternatives were neglected, simplifications were made, or solutions were described as not being final but of a rather suggestive nature. In chapter 5, for example, I explained the problem of the current positioning semantics producing non-aesthetic or even unergonomic results without providing a solution for the problem. In chapter 6, I described a general idea for a description language for structured backgrounds which allows applications to create instances of a template, yet this solution has the disadvantage of having to describe spatial relations with textual means and, aside from that, has not been implemented and thus has not been tested yet. Each of those points, among many others, of course, deserves further thoughts and further experimentation.

Aside from such issues which already came up in the course of the thesis, in this section, I bring up some of the more existential questions about what the concept is useful for, how it can be combined with other techniques or to what extent it is better than existing user interface approaches. While each of these questions deserves an in-depth evaluation, I nevertheless want to give some initial thoughts on which direction an answer might go in. The possible research questions mentioned here are, of course, not complete and even less so are the initial ideas for their answers.

#### **Characteristics of applications using Responsive Positioning**

In the course of the thesis, I frequently claimed that Responsive Positioning can hardly be the right user interface technique for every kind of application. The question, which applications it can be successfully used for and for which it finally proves to be a better or worse choice than existing user interface solutions can, of course, only be answered after some implementations have been made and evaluated. Yet, there certainly are tasks where it is clear that Responsive Positioning *cannot* be the right choice of user interface. These are, for example, all those tasks which are associated with editing content. One could, for example, not perform fine-grained text editing using Responsive Positioning.

Without having hard evidence provided by a usability test or by another kind of evaluation of an application using Responsive Positioning, I can already formulate some criteria an application must meet in order to use Responsive Positioning as its interface:

- An “objectification” of parts of the application logic must be possible. This means that one must be able to find objects which can be visualised, selected and manipulated. At this point, it might make sense to revisit the argumentation put forward by the protagonists of direct manipulation such as Shneiderman (1983) who claimed one needs to find “an appropriate representation or a model of reality”<sup>271</sup>. Such a model might help finding the necessary objects of manipulation. The focus on “reality” though must not lead to the impression that objects for Responsive Positioning user interfaces necessarily have to be objects in real life or that operations in Responsive Positioning have to literally translate to real-world operations. The operations in the multimedia control application, for example, do not resemble “real-world” operations (one does not drag anything to a projector) and the connector objects in the extended coffee machine configurator do not at all represent any real-world objects.
- The operations performed on these objects must be the manipulation of attributes which are meaningful in the context of the application. Among the applications described in chapter 6, for example, documents were provided with an attribute describing their topic, objects representing projects were attributed with information about whether or not they are ‘in budget’, or beverage objects were provided with information about their amount of fluid. There are applications which consist of manipulable objects for which this description is not true. While objects in a word processor, the letters for example, do carry some attributes such as the formatting, typical text editing is not about assigning attributes to the letters, but about bringing them into a certain order, which dominates the task. Dragging letters into a region to make them bold might technically make those letters bold, but it would not serve the task of effective text-editing.

---

<sup>271</sup> Shneiderman (1983), page 66. See the argumentation in chapter 2.4.3.

- It must be possible to create meaningful background structures for the respective attribute changes. Due to spatial limitations both in its size as well as in its number of dimensions, building suitable structured backgrounds is limited, which implies that there is a limit on both the number of attributes as well as the number of distinct values which can be considered at the same time. These limits, as well as “tricks” how they might be circumvented (like the adjacency matrix in the coffee machine example), need further consideration.
- Ideally, the objects of manipulation and the objects of responses should be the same in the sense that objects which are perceived should, at least in most cases, also be the objects which are manipulated. It would, of course, be possible to use distinct input and output objects, yet that would be a degenerated kind of Responsive Positioning, effectively resulting in two different views on the application data, where one view would be input only while the other view would be output only.

Further research in this regard should provide guidelines which better indicate in which cases Responsive Positioning should be implemented and in which cases it cannot be considered. If possible, techniques should be developed which describe how structures and objects can be derived from descriptions of the application domain.

### **Integration with conventional user interfaces**

As Responsive Positioning cannot be used as a user interface for every kind of application, it is unlikely that it can ever be the only user interface for complex applications. This raises the question of how Responsive Positioning can be sensibly combined with other means of input or output. Building on the concepts of chapter 6, there are two main possibilities of how Responsive Positioning can be combined with other user interface techniques.

One of these possibilities is integrating standard user interface elements into Responsive Positioning user interfaces. Chapter 6 already contained some examples where elements like text input areas were used in a Responsive Positioning view as so-called foreign objects within the structured background. Further examples of such foreign objects could be buttons, which can be pressed in order to invoke functions, or switches, which may be used to configure the

evaluation semantics of a structured background. Apart from these foreign objects, standard user interface techniques can also be coupled with the foreground objects. In the context of the budget planning application, for example, I suggested that double-clicking an object could open a separate configuration view. Similarly, right-clicking such an object could trigger a standard pop-up-menu to appear.

An interesting combination can be one of Responsive Positioning structures with sensitive areas which are implemented in the WebArena prototype. Sensitive areas, though visually similar, are not a positioning technique as semantic positioning or Responsive Positioning. They neither position objects nor do they evaluate object attributes. A sensitive area rather determines, whether an object has been dragged onto it or away from it and triggers an application function accordingly.



*Figure 94: The combination of Responsive Positioning structures with sensitive areas*

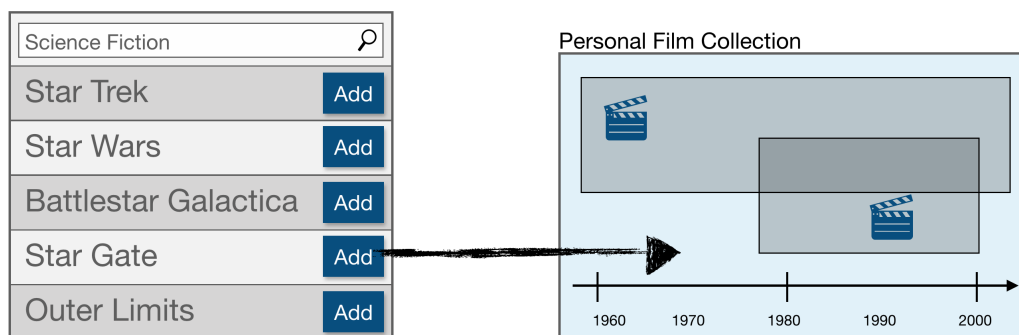
Figure 94 shows a combination of Responsive Positioning user interface elements with such sensitive areas. In the structure on the left, documents are positioned in relation to a matrix according to Responsive Positioning semantics. These objects can be dragged out of that structure and be put, for example, onto a print area. This area is not part of the structured background of Responsive Positioning. Putting an object there does not assign any attributes to it. It merely is a sensitive area which reacts to the object being dragged onto it, which in this case would mean invoking a print operation. In regard to Responsive Positioning semantics,

## 7 Summary and Conclusions

---

dragging an object onto a sensitive area is as “meaningless” as dragging the object to neutral ground, yet assuming that, regarding to the Responsive Positioning semantics, an object dragged out of the matrix changes its context attribute, such a printing option could automatically send the object back to where it came from after the printing is finished.

An alternative to integrating standard user interface elements into Responsive Positioning views is, of course, doing the opposite, thus integrating Responsive Positioning views into an application which otherwise uses conventional user interface techniques.



*Figure 95: An embedded Responsive Positioning view*

Figure 95 shows an application which contains a dedicated and embedded Responsive Positioning view. The assumed purpose of this application is the management of a personal selection of films in a graphical structure. In order to add a film to the arrangement, a database is queried using a standard user interface consisting of a search bar and a result list. In order to add a new film object to the Responsive Positioning view, an add button within the result display has to be pressed. For such solutions, a framework based on the Model-View-ResponsivePositioning-Model pattern described in chapter 6.4 should prove to be a good starting point. It supports integrated Responsive Positioning views as it encapsulates the whole view together with all the Responsive Positioning semantics. This way it should easily be connectable to other parts of the user interface by implementing a proper interface within the ResponsivePositioning-Model.

Both considering the integration of standard user interface elements into Responsive Positioning views as well as integrating a Responsive Positioning



view into a standard user interface needs further investigation both from a usability point of view as well as from a technical point of view as, up to this point, there is no implementation of the Model-View-ResponsivePositioning-Model pattern which would allow the integration suggested above.

### **Ergonomics of Responsive Positioning User Interfaces**

While the examples within this thesis show that Responsive Positioning can be used as a user interface, it remains to be seen if user interfaces based on it are “better” in the sense that they are more ergonomic or more usable. Some indications regarding these questions might be derivable from theoretical considerations. The fact that in Responsive Positioning there is only one possible manipulation, the movement of an object in relation to the background, for instance, makes the user interface easy to grasp. It may still be difficult to understand the consequences of object manipulations, but it is not difficult to determine which objects are manipulable and how that manipulation is performed. In the end, though, the questions of usability and ergonomics can only be answered by performing user tests. Finding empirical evidence using such tests might not prove to be easy, though. User tests would require comparing two user interfaces of which one uses Responsive Positioning while the other one does not. Such a test can surely be conducted, yet there are some pitfalls which have to be avoided.

1. Despite the different techniques used, the two user interfaces must be equally well designed. Comparing a well designed Responsive Positioning user interface with a poorly designed form-based user interface, for example, will surely result in the Responsive Positioning solution performing better.
2. When comparing the user interface techniques, the aspect of familiarity or the lack thereof will influence the outcome (likely) to the disadvantage of the unknown and therefore unfamiliar Responsive Positioning interface.
3. Having found out that for a certain user interface and a certain user group, Responsive Positioning is the better (or worse) solution, this of course does not allow the conclusion that Responsive Positioning user interfaces are equally good (or bad) in other cases or among other user groups.

These pitfalls can partially be avoided. Problem number 3 leads to the question mentioned above, namely which kind of task Responsive Positioning is a good technique for. This question and its possible dependence on the target group, can only be answered by performing a great number of tests among different target groups and with different types of applications. It therefore depends on an adoption of the technique by software manufacturers. As for problem number 2, the problem of differences in the familiarity, the ideal solution of finding people who have never used either technique proves to be virtually impossible today, the problem can only be reduced by training the test subjects on the new technique, so that they will not be completely unfamiliar with the user interface. Last of all, for the problem of considering equally good user interfaces in order to allow a comparison, the scientist conducting the tests must be aware of possible implicit biases when creating the test cases. Test results should be questioned in this regard in all cases and, if possible, the creation of the tests should be decoupled from those people having an interest in their outcomes.

Many more research questions come up when considering to extend the concept. One could, for example, extend objects to refer to more than a single point. Such a change would, without question, imply profound changes in the Responsive Positioning semantics. Similarly fundamental changes would have to be made when extending the attribute semantics from the simple key-value concept used in this thesis to something which allows multiple values for an object (like, for example, a person having a ‘research interests’ attribute which can have values like ‘hypertext’ and ‘ergonomics’ at the same time).

## 7.2 Epilogue

In this thesis, I developed a new graphical user interface paradigm as a technique which in contrast to other established user interface techniques makes use of the structurability of space.

The idea of developing Responsive Positioning as a user interface technique evolved over time as a convergence of two scientific interests. Even as a student, I took part in lectures based on the Jour Fixe and the MediaThing concepts<sup>272</sup>. Later, as a research assistant, I helped to organise courses based on the MediaThing concepts. During this time, I learned to understand the expressiveness of spatial positioning in knowledge work. In lectures or seminars based on the MediaThing concept, students have to build knowledge structures. These consist of a background structure according to which objects have to be positioned. The objects often represent concepts or technologies but, depending on their scientific question and the approach taken, can also represent key players or even scientific texts. The object placements according to such a structure have to be justifiable. One could also say, they have to be formally correct, which means they have to correspond to some measurable or at least determinable property of the object or what it stands for. When the background structure is well chosen, it can verify or falsify a previously formulated hypothesis and if it does, it reveals this verification of falsification in an instant.

Another research interests of mine are the ergonomics of user interfaces. Recognizing that a certain user interface has some issues is easy. Almost everyone can do that. Being able to explain why this is the case is somewhat more difficult, and being able to tell what has to be done about it can be very difficult, especially having the aspiration of coming up with a set of rules on how to avoid running into certain issues in the first place. A part of the theory behind the concept we present within our lectures is the insight that what the designer of a user interface does is nothing but designing an object arrangement on the user's screen. The design includes not only how the objects look but also where they are placed and how they can be manipulated.

---

<sup>272</sup> See chapter 6.5

the structured background and the meaningful object placements in relation to it, is virtually unheard of in the field of the design of user interfaces. Of course, within some specialised applications, often where the spatial design was transferred from analogue counterparts such as in a digital calendar, space is structured—as a matrix in this case—yet as a general technique, spatial structures play only a minor role. Of course, space in general does play an important role in user interfaces. In a well designed input form, for example, user interface elements are grouped spatially. Spatial operations where objects are moved around the screen also have existed for quite some time, especially in the form of drag&drop within the desktop metaphor, but dragging an object from one location and dropping it on top of another object or within another window may be a convenient method of performing object operations which, of course, has its advantages compared to its alternatives, but by dragging and dropping one does not assign attributes, let alone meaning, to an object. A desktop may facilitate the use of a computer and may help users to keep track of their documents and folders, but it does not allow for anything like the structured backgrounds we have within the knowledge structures in the MediaThing scenario. This led to the idea of bringing both concepts together and as a result of this idea, over time I came up with what you have read in this thesis.

Looking back at the line of argumentation I made in this thesis, I based my considerations on theoretical foundations of technical potentials of interactive user interfaces as well as on concepts of differential experience and knowledge work. When following the argumentation, one might get the impression that a complex concept like the one presented in this thesis could be derived from such theoretical considerations. This is, of course, not the case, or, to be scientifically correct, it was not the case in this thesis. Nevertheless, the approach has proven to be valuable for the formulation of a sound research hypothesis on which the subsequent argumentation could be built. I am well aware that whether or not the technique described in this thesis will be successful in the end is not very much a question of its scientific foundation or if the argumentation in this thesis is sound, but to a great extent depends on whether people jump on the train and implement Responsive Positioning in their user interfaces or even get inspired for own research endeavours. In this I hope, that with this thesis, I have given them enough reason to do so!

## References

- Agarawala (2006):** Agarawala, Anand; Balakrishnan, Ravin. *Keepin' It Real: Pushing the Desktop Metaphor with Physics, Piles and the Pen*. In: CHI 2006 Proceedings – Interacting with Large Surfaces. ACM, pp. 1283-1297 (2006)
- Apple (1983):** Apple Computer. *Lisa Owner's Guide*. Apple Computer, Cupertino (CA) (1983)
- Apple (2015):** Apple. Model-View-Controller. In: Cocoa Core Competencies. Published on <https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html> (2015)
- Arnheim (1969):** Arnheim, Rudolf. *Visual Thinking*. University of California Press, Berkeley, Los Angeles, London (1969)
- Astrahan&Jacobs (1983):** Astrahan, Morton; Jacobs, John. *History of the Design of the SAGE Computer – The AN/FSQ-7*. In: Annals of the History of Computing, Vol. 5 No. 4, pp. 340-349 (1983)
- Brodbeck et al. (2009):** Brodbeck, Dominique; Mazza, Riccardo; Lalanne, Denis. *Interactive Visualization – A Survey*. In: Lalanne, Denis; Kohlas, Jürg (eds.) Human Machine Interaction: Research Results of the MMI Program. Springer, Berlin, Heidelberg pp. 27-46 (2009)
- Budd (2004):** Budd, John W. *Mind Maps As Classroom Exercises*. In: The Journal of Economic Education, Vol. 35 No.1, pp. 35-46 (2004)
- Burks et al. (1946/1987):** Burks, Arthur W.; Goldstine, Herman H.; von Neumann, John. *Preliminary Discussion of the Logical Design of an Electronic Computing Instrument*. In: Aspray, William; Burks, Arthur (eds.) Papers of John von Neumann on Computing and Computer Theory. MIT Press, Cambridge (MA), pp. 97-142 (1946/1987)
- Büse (2001):** Büse, Daniel. *Konzeption und prototypische Umsetzung eines Shared Whiteboard für eine kooperationsunterstützende Lernumgebung*. Diploma thesis, Paderborn University (2001)
- Buzan (2002):** Buzan, Tony. *How to Mind Map*. Thorsons, London (2002)

## References

---

- Buzan&Buzan (2003):** Buzan, Tony; Buzan, Barry. *The Mind Map Book*. BBC Worldwide, London (2003)
- Ceruzzi (1998):** Ceruzzi, Paul. *A history of modern computing*. MIT Press, Cambridge (MA), London (1988)
- Ceruzzi (2012):** Ceruzzi, Paul. *Computing – A Concise History*. MIT Press, Cambridge (MA), London (2012)
- Christensen (2010):** Christensen, Ole. *Functions, Spaces, and Expansions: Mathematical Tools in Physics and Engineering*. Birkhäuser, Boston, Basel, Berlin (2010)
- Commodore (1982):** Commodore Business Machines. *Commodore 64 User's Guide*. Commodore Computer (1982)
- Corbató et al. (1962):** Corbató, Fernando; Merwin-Daggett, Majorie; Daley, Robert. *An Experimental Time-Sharing System*. In: Proceedings of the May 1-3, 1962, spring joint computer conference, pp. 335-344 (1962)
- Curtis (1992):** Curtis, Pavel. *Mudding: Social Phenomena in Text-Based Virtual Realities*. In: Ludow, Peter; Godwin, Mike (eds.) *High noon on the electronic frontier: Conceptual issues in cyberspace*. MIT Press, Cambridge (MA), pp. 347-374 (1992)
- Davies (2011):** Davies, Martin. *Concept mapping, mind mapping and argument mapping: what are the differences and do they matter?* In: Higher Education, Vol. 62 No. 3, pp. 279-301 (2011)
- Degn et al. (1973):** Degn, Christian; Eggert, Erwin; Kolb Albert. *Seydlitz für Gymnasien, Band 3B, Amerika Ozeane Polargebiete*. Ferdinand Hirt, Kiel, and Hermann Schroedel, Hannover (1973)
- Dewey&Ratner (1939):** Dewey, John; Ratner, Joseph. *Intelligence in the Modern World*. Random House of Canada, Toronto (1939)
- DiGiano et al. (2006):** DiGiano, Chris; Tatar, Deborah; Kireyev, Kirill. *Learning from the Post-It: Building collective intelligence through lightweight, flexible technology*. Published on <https://groupscribbles.sri.com/publications.html> (2006)

- Dijkstra (1970):** Dijkstra, Edsger W. *Structured Programming*. In: Buxton, John N.; Randell, Brian (eds.): *Software Engineering Techniques*. Nato Science Committee, Brussels, pp. 84-88 (1969)
- Drucker (1959):** Drucker, Peter F. *Landmarks of Tomorrow*. Harper and Brothers, New York (1959)
- Drucker (1973):** Drucker, Peter. *Management – Tasks, Responsibilities, Practices*. Truman Talley Books, E.P. Dutton, New York (1973)
- Eckstein (2007):** Eckstein, Robert 2007. *Java SE Application Design With MVC*. In: Oracle Technology Network, published on <http://www.oracle.com/technetwork/articles/javase/index-142890.html> (2007)
- Engelhardt (1999):** Engelhardt, Yuri. *Meaningful space*. In: *If/Then: Design implications of new media*. Netherlands Design Institute, Amsterdam, pp. 72-74 (1999)
- Engelhardt (2002):** Engelhardt, Yuri. *The Language of Graphics – A framework for the analysis of syntax and meaning in maps, charts and diagrams*. Dissertation, Faculteit der Natuurwetenschappen, Wiskunde en Informatica, Universiteit van Amsterdam (2002)
- Engelhardt (2006):** Engelhardt, Yuri. *Objects and Spaces: The Visual Language of Graphics*. In: Burker-Plummer, D. (editor) *Diagrams 2006: Diagrammatic Representation and Inference*. Springer, Berlin, Heidelberg, pp. 104-108 (2006)
- Engelhardt (2007):** Engelhardt, Yuri. *Syntactic Structures in Graphics*. In: *Computational Visualistics and Picture Morphology*, Vol. 5, pp. 23-35 (2007)
- Erren (2010):** Erren, Patrick. *Semantic Positioning*. Dissertation, Paderborn University (2010)
- Erren&Keil (2006):** Erren, Patrick; Keil, Reinhard. *Semantic Positioning as a Means for Visual Knowledge Structuring*. In: Neijdl, Wolfgang; Tochtermann, Klaus (eds.): *EC-TEL 2006, LNCS4227*. Springer, Berlin, Heidelberg, pp. 591-596 (2006)
- Esposito (1993):** Esposito, Elena. *Der Computer als Medium und Maschine*. In: *Zeitschrift für Soziologie*, Vol. 22 No. 5, pp. 338-354 (1993)

## References

---

- Floyd (1981):** Floyd, Christiane. A Process-Oriented Approach to Software Development. In: Systems Architecture, Proceedings of the 6th ACM European Regional Conference. Westbury House, pp. 285-294 (1981)
- Floyd (1987):** Floyd, Christiane. *Outline of a Paradigm Change in Software Engineering*. In: Bjerknes, Gro; Ehn, Pelle; Kyng, Morten (eds.) Computers and Democracy, A Scandinavian Challenge. Dower Publishing, Avebury, pp. 185-202 (1987)
- Floyd (1992):** Floyd, Christiane. *Software development as reality construction*. In: Floyd Christiane; Züllinghoven, Heinz; Budde Reinhard; Keil-Slawik Reinhard (eds.) Software development and reality construction. Springer, Berlin, Heidelberg, pp. 86–100 (1992)
- Franchi (1975):** Franchi, P.. *Interactivity, Sharing and Virtual Machines*. In: Thomas Massam (editor) Computer Frontiers. Editions Georgi, St. Saphorin, pp. 121-136 (1975)
- Frutiger (1989):** Frutiger, Adrian. *Designs and Symbols – Their Design and Meaning*. Van Nostrand Reinold, New York (1989)
- Green (1985):** Green, Mark. *Report on Dialogue Specification Tools*. In: Günther E. Pfaff (editor) User interface management systems. Springer, Berlin, Heidelberg, pp. 9-20 (1985)
- Galilei (1638/1914):** Galilei, Galileo, *Discorsi e Dimostrazioni Matematiche Intorno a Due Nuove Scienze*. (1638) Translation by Crew, Henry; De Salvio, Alfonso. *Dialogues Concerning Two New Sciences*. Dover, New York (1914)
- Gibson (1986):** Gibson, James J. *The Ecological Approach to Visual Perception*. Psychology Press, New York, Hove (1986)
- Goldberg&Robson (1983):** Goldberg, Adele; Robson, David. *Smalltalk 80 – The Language and its Implementation*. Addison-Wesley, Boston (1983)
- Grier (1996):** Grier, David. *The ENIAC, the Verb "to program" and the Emergence of Digital Computers*. In: IEEE Annals of the History of Computing, Vol. 18 No. 11, pp. 51-55 (1996)
- Hampel (2001):** Hampel, Thorsten. *Virtuelle Wissensräume*. Dissertation, Paderborn University (2001)



- Hampel&Keil-Slawik (2001):** Hampel, Thorsten; Keil-Slawik, Reinhard. *sTeam: Structuring Information in Teams – Distributed Knowledge Management in Cooperative Learning Environments*. In: ACM Journal on Educational Resources in Computing, Vol. 1 No. 2, article #3 (2001)
- Hampel et al. (2003):** Hampel, Thorsten; Keil-Slawik, Reinhard; Eßmann, Bernd. *Jour Fixe – We Are Structuring Knowledge – Collaborative Structuring of Semantic Spaces as a Didactic Concept and New Form of Cooperative Knowledge Organization*. In: E-Learn: World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education. Association for the Advancement of Computing in Education, Charlottesville (VA), pp. 225-232 (2003)
- Henderson&Card (1986):** Henderson, D. Austin; Card, Stuart. *Rooms: The Use of Multiple Virtual Workspaces to Reduce Space Contention in a Window-Based Graphical User Interface*. In: ACM Transactions on Graphics, Vol.5 No.3, pp 211-243
- Hollan et al. (2000):** Hollan, James, Hutchins, Edwin, Kirsh, David. *Distributed Cognition: Toward a New Foundation for Human-Computer Interaction Research*. In: ACM Transaction of Computer-Human-Interactions, Vol. 7 No. 2, June 2000, pp. 174-196 (2000)
- Hutchins et al. (1985):** Hutchins, Edwin; Hollan, James; Norman, Donald. *Direct Manipulation Interfaces*. In: Human-Computer Interaction, Vol. 1 No. 4, pp. 331-338 (1985)
- Ifrah (1981/1985):** Ifrah, Georges. *From one to zero*. Viking Penguin, New York (1985)
- ISO 9241-110 (2006):** ISO Technical Committee 159. *ISO 9241 Part 110 - Dialog principles*. International Organization for Standardization (2006)
- Jakoblew et al. (2015):** Jakoblew, Marcel; Keil, Reinhard; Winkelnkemper, Felix. *Forschendes Lernen durch semantisches Positionieren*. In: Schwill, Andreas; Schubert, Sigrid (eds.) HDI 2014: Gestalten von Übergängen, pp. 109-124 (2015)
- Janlert&Stolterman (2017):** Janlert, Lars-Erik; Stolterman, Erik. *The Meaning of Interactivity – Some Proposals for Definitions and Measures*. In: Human-Computer Interaction, Vol. 32 No. 3, pp. 103-138 (2016)

- Janlert&Stolterman (2015):** Janlert, Lars-Erik; Stoltermann, Erik. *Faceless Interaction – A Conceptual Examination of the Notion of Interface: Past, Present, and Future*. In: Human-Computer Interaction, Vol. 30 No. 6, pp. 507-539 (2014)
- Johnson et al. (1989):** Johnson, Jeff; Roberts, Theresa; Verplank, William; Smith, David C.; Irby, Charles; Beard, Marian; Mackey, Kevin. *The Xerox Star: A Retrospective*. In: IEEE Computer, September 1989, pp. 11-29 (1989)
- Johnson&Roberts (1989):** Johnson, Jeff; Roberts, Teresa L. *Direct Manipulation*. In: IEEE Computer, September 1989, pp. 14,15 (1989)
- Keil (2007):** Keil, Reinhard. *Medienqualitäten beim eLearning: Vom Transport zur Transformation von Wissen*. In: Bibliothek Forschung und Praxis, Vol. 31 No. 1, pp. 41-50 (2007)
- Keil (2009):** Keil, Reinhard. *Medi@Thing – Ein didaktischer Ansatz zum ko-aktiven Lernen*. In: Commentarii informaticae didacticae: (CID). Universität Potsdam, pp. 9-24 (2009)
- Keil (2010):** Keil, Reinhard. *Das Differenztheater*. In: Bublitz, Hannelore; Marek, Roman; Steinmann, Christina; Winkler, Hartmut (eds.): *Automatismen*. Wilhelm Fink, München, pp. 205-229 (2010)
- Keil (2011):** Keil, Reinhard. *Hypothesengeleitete Technikgestaltung als Grundlage einer kontextuellen Informatik*. In: Breiter, Andreas, Wind, Martin (eds.) *Informationstechnik und ihre Organisationslücken. Soziale, politische und rechtliche Dimensionen aus der Sicht von Wissenschaft und Praxis*. LIT, Berlin pp. 165-184 (2011)
- Keil et al. (2006):** Keil, Reinhard; Fleigel, Lars; Geissler, Sabrina. *MOBiDig: Manipulierbare Objekte in digitalen Systemen*. Technical report, Universität Paderborn (2006)
- Keil-Slawik (1990):** Keil-Slawik, Reinhard. *Konstruktives Design – Ein ökologischer Ansatz zur Gestaltung interaktiver Systeme*. Habilitation, TU Berlin (1990)

- Keil-Slawik (2000):** Keil-Slawik, Reinhard. *Zwischen Vision und Alltagspraxis: Anmerkungen zur Konstruktion und Nutzung typographischer Maschinen*. In: Boehnke, Klaus; Voss, G. Günter; Holly, Werner (eds.) *Neue Medien im Alltag: Begriffsbestimmungen eines interdisziplinären Forschungsfeldes*, Leske+Budrich, Opladen, pp. 190-220 (2000)
- Kidd (1994):** Kidd, Alison. *The Marks are on the Knowledge Worker*. In: CHI'94 Celebrating Interdependence, pp. 186-191 (1994)
- Kiousis (2002):** Kiousis, Spiro. *Interactivity: a concept explication*. In: *new media & society*, Vol. 4 No. 3, pp. 355-383 (2002)
- Kirsh (1995):** Kirsh, David. *The intelligent use of space*. In: *Artificial Intelligence*, Vol. 73, pp. 31-68 (1995)
- Klemme et al. (1998):** Klemme, Michael; Kuhnert, Ralf; Selke, Harald. *Semantic Spaces*. In: Höök, Kristina; Munro, Alan Jo; Benyon, David (eds.): *Workshop on Personalised and Social Navigation in Information Space*. SICS Technical Report. Vol. 98, pp. 109-118 (1998)
- Krämer (1988):** Krämer, Sybille. *Symbolische Maschinen – Die Idee der Formalisierung in geschichtlichem Abriß*. Wissenschaftliche Buchgesellschaft, Darmstadt (1988)
- Krämer (2014):** Krämer, Sybille. *Mathematizing Power, Formalization, and the Diagrammatical Mind or: What Does “Computation” Mean?* In: *Philosophy & Technology*, Vol. 27 No. 3, pp. 345–357 (2014)
- Krasner&Pope (1988):** Krasner, Glenn; Pope, Stephen. *A cookbook for using the model-view controller user interface paradigm in Smalltalk-80*. In: *Journal of Object-Oriented Programming*, Vol. 1 No. 3, pp. 26-49 (1988)
- Kuhn (1957):** Kuhn, Thomas. *The Copernican Revolution*. Harvard University Press, Berkeley (CA) (1957)
- Kuhn (1962/1970):** Kuhn, Thomas. *The Structure of Scientific Revolutions*. International Encyclopedia of Unified Science. The University of Chicago Press, Chicago, London (1962, 2<sup>nd</sup> edition 1970)
- LaLonde&Pugh (1991):** LaLonde, Wilf; Pugh, John. *Inside Smalltalk Volume II*. Prentice-Hall International, New Jersey (1991)

## References

---

- Lansdale (1988):** Lansdale, Mark. *The psychology of personal information management*. In: Applied Ergonomics, Vol. 19 No. 1, pp. 55-66 (1988)
- Laurel (2014):** Laurel, Brenda. *Computers as Theatre. Second Edition*. Addison-Wesley, Boston (2014)
- Leroi-Gourhan (1964):** Leroi-Gourhan, André. *Gesture and Speech*. MIT Press, Cambridge (MA) (1964, translation 1993)
- Looi et al. (2008):** Looi, Chee-Kit; Lin, Chiu-Pin; Liu, Kuo-Ping. *Group Scribbles to Support Knowledge Building in Jigsaw Method*. In: IEEE Transactions of Learning Technologies, Vol. 1 No. 3, pp. 157-164 (2008)
- Mahoney (2005):** Mahoney, Michael S. *The Histories of Computing(s)*. In: Interdisciplinary science reviews, Vol. 30 No. 2, pp. 119-135 (2005)
- Malone (1983):** Malone, Thomas W. *How Do People Organize Their Desks? Implications for the Design of Office Information Systems*. In: ACM Transactions on Office Information Systems, Vol. 1 No. 1, pp. 99-112 (1983)
- Maloney et al. (2010):** Maloney, John; Resnick, Mitchel; Rusk, Natalie; Silverman, Brian; Eastmond, Evelyn. *The Scratch Programming Language and Environment*. In: ACM Transactions on Computing Education. Vol. 10 No. 4, article 16 (2010)
- Mander et al. (1992):** Mander, Richard; Salomon, Gitt; Wong, Yin Yin. *A Pile Metaphor for Supporting Casual Organization of Information*. In: CHI'92 – Proceedings of the SIGCHI conference on Human factors in computing systems, ACM, pp. 627-634 (1992)
- McMillan&Wang (2002):** McMillan, Sally; Hwang, Jang-Sun. *Measures of Perceived Interactivity: An Exploration of the Role of Direction of Communication, User Control, and Time in Shaping Perceptions of Interactivity*. In: The Journal of Advertising, Vol. 31 No., pp. 29-42
- Mattes (2002):** Mattes, Wolfgang. *Methoden für den Unterricht*. Schöningh, Paderborn (2002)
- Microsoft (2010):** Microsoft patterns & practices Developer Center. *Implementing the Model-View-ViewModel Pattern*. Published on <https://msdn.microsoft.com/en-us/library/ff798384.aspx> (2010)

- Naylor (1974):** Naylor, Ronald. *Galileo and the Problem of Free Fall*. In: The british journal for the history of science, Vol. 7 No. 26 (1974)
- Niehus (2006):** Niehus, Dominik. *Konzeption und Umsetzung einer Rich Client-basierten kooperationsgestützten Lern- und Arbeitsumgebung*. Diploma thesis, Paderborn University (2006)
- Nielsen (2012):** Nielsen, Jakob. *Browser and GUI Chrome*. Published on <https://www.nngroup.com/articles/browser-and-gui-chrome> (2012)
- Norman (2013):** Norman, Donald. *The design of everyday things. Revised and expanded edition*. Basic Books, New York (2014)
- Nowaczyk (2005):** Nowaczyk, Olaf. *Explorationen: Ein Ansatz zur Entwicklung hochgradig interaktiver Lernbausteine*. Dissertation. HNI Verlagsschriftenreihe, Band 171 (2005)
- Novak&Cañas (2008):** Novak, Joseph; Cañas, Alberto. *The theory underlying concept maps and how to construct them*. Florida Institute for Human and Machine Cognition. Published on <http://eprint.ihmc.us/id/eprint/5> (2008)
- Perkins (1993):** Perkins, David. *Person Plus - A Distributed View of Thinking and Learning*. In: Salomon, Gavriel (editor) *Distributed cognitions: Psychological and educational considerations*. Cambridge University Press. pp. 88-110 (1993)
- Petertonkoker (2013):** Petertonkoker, Jan. *Flexible Coupling of Graphical Structured Knowledge Spaces*. Master thesis. Paderborn University (2013)
- Prante et al. (2004):** Prante, Thorsten; Streitz, Norbert; Tandler, Peter. *Roomware: Computers Disappear and Interaction Evolves*. In: IEEE Computer, December 2004, pp. 47-54 (2004)
- Purvis (1983):** Purvis, Keith. *The teacher as moderator: a technique for interactional learning*. In: ELT Journal, Vol. 37 No. 3, pp. 221-228 (1983)
- Quiring&Schweiger (2008):** Quiring, Oliver; Schweiger, Wolfgang. *Interactivity: A review of the concept and a framework for analysis*. In: Communications – European Journal of Communication Research, Vol. 33 No. 2, pp. 147-167 (2008)

## References

---

- Rafaeli (1988):** Rafaeli, Sheizaf. *Interactivity – From New Media to Communication*. In: Sage Annual Review of Communication Research: Advancing Communication Science, Vol. 16, pp. 110-134 (1988)
- Reenskaug (1979a):** Reenskaug, Trygve. *Thing-Model-View-Editor*. Published on <https://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html> (1979)
- Reenskaug (1979b):** Reenskaug, Trygve. *Models, Views, Controllers*. Published on <https://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html> (1979)
- Richards (2002):** Richards, Clive. *The Fundamental Design Variables of Diagramming*. In: Diagrammatic Representation and Reasoning. Springer, London, pp. 85-102 (2002)
- Robertson et al. (1998):** Robertson, George; Czerwinski, Mary; Larson, Kevin; Robbins, Daniel; Thiel, David; van Dantzich, Maarten. *Data Mountain: Using Spatial Memory for Document Management*. In: UIST '98. ACM, pp. 153-162 (1998)
- Robinson (2008):** Robinson, Anthony. *Design for Synthesis in Geovisualization*. Dissertation, Pennsylvania State University (2008)
- Rubin (1983):** Rubin, Lisa. *Syntax-Directed Pretty Printing – A First Step Towards a Syntax-Directed Editor*. In: IEEE Transactions on Software Engineering, Vol. SE-9 No. 2, pp. 119-127 (1983)
- Salomon&Perkins (2011):** Salomon, Gavriel; Perkins, David. *Do Technologies Make Us Smarter? Intellectual Amplification With, Of, and Through Technology*. In: Sternberg, Robert J.; Preiss, David D. (eds.) *Intelligence and Technology – The Impact of Tools on the Nature and Development of Human Abilities*, Routledge, pp. 71-86 (2005)
- Schelhowe (1997):** Schelhowe, Heidi. *Das Medium aus der Maschine – Zur Metamorphose des Computers*. Campus, Frankfurt, New York (1997)
- Schelhowe (2007):** Schelhowe, Heidi. *Interaktion und Interaktivität – Aufforderungen zu einer technologiebewussten Medienpädagogik*. In: Jahrbuch Medien-Pädagogik 6, Springer VS, Verlag für Sozialwissenschaften, Wiesbaden, pp. 144-160 (2007)

- Sens (2015):** Sens, Christoph. *Konzeption einer Auswertungslogik für semantische Positionierung und prototypische Umsetzung in WebArena*. Master thesis, Paderborn University (2015)
- Shneiderman (1983):** Shneiderman, Ben. *Direct Manipulation: A Step Beyond Programming Languages*. In: IEEE Computer, August 1983, pp. 57-69 (1983)
- Shneiderman (1996):** Shneiderman, Ben. *The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations*. In: VL '96 Proceedings of the 1996 IEEE Symposium on Visual Languages, pp. 336-343 (1996)
- Skagestad (1998):** Skagestad, Peter. *Peirce, virtuality, and semiotic*. In: Proceedings from The Twentieth World Congress of Philosophy, pp. 10-15 (1998)
- Smith (2009):** Smith, Josh. *Patterns-WPF Apps With The Model-View-ViewModel Design Pattern*. In: Microsoft Magazine, Vol. 24 No. 02, Published on <https://msdn.microsoft.com/en-us/magazine/dd419663.aspx> (2009)
- Smith et al (1982):** Smith, David C.; Irby, Charles; Kimball, Ralph; Verplank, Bill L., *Designing the Star User Interface*. In: Byte Magazine, April 1982, pp. 242-282 (1982)
- Spencer et al. (2003):** Spencer, Liz; Ritchie, Jane; Lewis, Jane; Dillon, Lucy. *Quality in qualitative evaluation: A framework for assessing research evidence*. National Centre for Social Research, Government Chief, Social Researcher's Office, UK (2003)
- Stotz (1964):** Stotz, Robert. *Man-Machine Console Facilities For Computer-Aided Design*. In: Simulation, April 1964, pp. R3-R7 (1964)
- Streeter (1974):** Streeter, Donald N. *The Scientific Process and the Computer*. Wiley&Sons, New-York, London, Sydney, Toronto (1974)
- Sutherland (1964):** Sutherland, Ivan. *Sketchpad – A Man-Machine Graphical Communication System*. In: Simulation, May 1964, pp. R3-R20 (1964)
- Terfloth (2016):** Terfloth, Lutz. *Unterstützung des Unterrichtsdiskurses mit Hilfe verteilter Aufgabenstellungen und Auswertung auf Basis von WebArena*. Bachelor thesis, Paderborn University (2016)

## References

---

- Tognazzini (1991):** Tognazzini, Bruce. *TOG on Interface*. Addison-Wesley, Boston (1991)
- Tognazzini (1995):** Tognazzini, Bruce. *Tog on Software Design*. Addison-Wesley, Boston (1995)
- von Neumann (1945):** von Neumann, John. *The First Draft of a Report on the EDVAC*. Moore School of Electrical Engineering (1945)
- Wegner (1997):** Wegner, Peter. *Why Interaction is More Powerful Than Algorithms*. In: Communications of the ACM, Vol 40. No 5, pp. 80-91 (1997)
- Wieser (1983):** Wieser, C. Robert. *The Cape Cod System*. In: Annals of the History of Computing, Vol. 4 No. 4, pp. 362-369 (1983)
- Williams (1990):** Williams, Michael. *Early Calculation – Mechanical Calculating Machines*. In: Aspray, William (editor) *Computing before Computers*. Iowa State University Press, Ames (IA) pp. 34-58 (1990)
- Wittgenstein (1953/1958):** Wittgenstein, Ludwig. *Philosophical Investigations*. Basil Blackwell, Oxford (original 1953, translation 1958)
- Wöpking (2010):** Wöpking, Jan. *Space, Structure, and Similarity. On Representationalist Theories of Diagrams*. In: Pombo, Olga; Gerner, Alexander (eds.) *Studies in Diagrammatology and Diagram Praxis*. College Publications, King's College London. pp. 39-56 (2010)

All web sources provided here were valid in October 2017