Spatio-Temporal Parsing in Spatial Hypermedia

Ph.D. Dissertation

Thomas Schedel

Abstract

Spatial Hypertext represents associations between chunks of information by spatial or visual attributes (such as proximity, color, shape etc.). This allows expressing information structures implicitly and in an intuitive way. However, automatic recognition of such informal, implicitly encoded structures by a machine (a so-called *spatial parser*) is still a challenge. One reason is, that conventional (non-adaptive) parsers are conceptually restricted by their underlying source of information (i. e., the spatial hypertext). Due to this limitation there are several types of structures that cannot be recognized properly. This inevitably limits both quality of parser output and parser performance. We claim that considering temporal aspects in addition to spatial and visual properties in spatial parser design will lead to significant increase in parsing accuracy, detection of richer structures and thus higher parser performance.

For the purpose of providing evidence, parsers for recognizing spatial, visual and temporal object relations have been implemented and tested in a series of user surveys. It turned out, that in none of the test cases pure spatial or visual parser could outperform the spatio-temporal parser. Instead, the spatio-temporal parser was able to compensate limitations of conventional parsers. Furthermore, differences in parsing accuracy were successfully tested for statistical significance. The results indicate a non-trivial effect that is recognizable by humans. We have shown that the addition of a temporal parser shifts machine detected structures significantly closer to what knowledge workers intend to express.

Resumé

Spatial Hypertext repræsenterer associationer mellem informationsdele fra spatiale eller visuelle attributter (såsom nærhed, farve, form osv.) Dette gør det muligt at udtrykke informationsstrukturer implicit og på en intuitiv måde. Men automatisk genkendelse af sådanne uformelle, implicit kodede strukturer (ved anvendelse af en såkaldt spatial parser) er stadig en udfordring. En af grundene er, at konventionelle (ikke-adaptive) parsere begrebsmæssigt er begrænset af deres underliggende informationskilde (dvs. den spatiale hypertekst). På grund af denne begrænsning, er der flere typer af konstruktioner, der ikke kan genkendes korrekt. Dette begrænser uundgåeligt både parser output kvaliteten og parser ydeevnen. Vi hævder, at inddragelse af temporale aspekter i tillæg til spatiale og visuelle egenskaber i spatial parser design vil føre til betydelig øgning af parsing nøjagtighed samt detektering af rigere strukturer og dermed højere parser ydeevne.

Med henblik på at vise dette, er parsere for genkendelse spatiale, visuelle og temporale objekt relationer implementeret og testet i en række brugerundersøgelser. Det viste sig, at i ingen af testcasene var en ren rumlig eller visuel parser mere effektiv end den spatio-temporale parser. I stedet var den spatio-temporale parser i stand til at kompensere for de konventionelle parseres begrænsninger. Endvidere blev forskelle i parsing nøjagtighed med succes testet for statistisk signifikans. Resultaterne indikerer en ikke-triviel effekt, der kan observeres af mennesker. Vi har vist, at tilføjelse af en temporal parser rykker maskinopdagede strukturer betydeligt tættere på, hvad videnarbejdere har til hensigt at udtrykke.

Contents

Chapter 1

Introduction

Hypertext goes far beyond what we know today from the *World Wide Web.* The general idea behind and its basic concepts go much further than simply linking and following trails of webpages. Consequently, hypertext allows for much more advanced applications than we are using today in our digital lifes. This introductory chapter is intended to illustrate that.

1.1 Classic Hypertext

Literally the prefix "hyper-" means "over" or "beyond", which highlights a superiority over (written) text. So in the classic sense of the word one could understand hypertext as a kind of "super"-text. What makes such text "super" or superior over plain text is its structure (i. e., how elements of a document entity are related to each other). Unlike written text, which can be seen as a one-dimensional sequence of words, sentences or pages respectively, hypertext documents do not need to be linear. They are rather networks of document items which may be traversed in a *non*-linear fashion. This classic type of hypertext is also known as *Node-Link Hypertext.*

Node-link hypertext has got its name from its underlying model, including two core elements, *node* and *link.* Nodes are holders of content (i. e., carriers of verbal or non-verbal information) [1]. The ways in which these chunks of text or other media are interconnected are defined by links [2]. Links describe (navigable) associations between nodes and can herewith determine both (a) how to reach a certain node and (b) how to interpret node content [1]. So on one hand links are tightly associated with traversal [1] and can therefore be used as a means of navigation, but on the other hand links also can express meaning [3] and hence can be used as an instrument for giving node content a semantic context [1].

Following the node-link model, authors of hypertext documents are required to express information structures by *formal* nodes and links. Thus, writers of hypertext must use *formalisms*. This, however, has certain drawbacks.

Explicitly creating and linking nodes is something many people are not familiar with. Therefore it is not surprising that some people may find it unnatural and therefore difficult to use the formal node-link scheme [1, 4]. This is especially the case when users only have an incomplete or vague idea about what to express. Fuzzy or partial knowledge is usually of implicit nature. Formalisms however require explicit statements. Thus, tacit information which might have been left implicit in a less formal representation must be made explicit when using nodes and links [5]. This leads to an increased cognitive load and high effort of expression.

On the other hand (formal) hypertext is not an ideal representation technique for describing emerging information structure; that is, for describing information that continually evolves [6]. Emerging structures change frequently and therefore require continuous structuring and restructuring of their representation. Recurring restructuring of networks of nodes and links, however, can be cognitively difficult and therefore time consuming [6]. Due to that overhead, traditional hypertext is poorly suited for many kinds of information analysis and design tasks [2].

Both, support for lowering users' effort of expression [7] and emerging structure support [1] can be provided by interfaces allowing for *visual communication*. Visual communication is conveyance of information via visual aids (e. g., shape, color etc.) and can be differentiated from written and spoken communication by its (a) implicit; (b) informal and (c) emergent nature [8]. Visual communication is amazingly natural [8] and intuitive. Often people organize information in the form of notes on boards and desks and herewith unselfconsciously communicate in a non-verbal but visual fashion [8]. Conveying such implicitly encoded information does not require prescriptive communication rules. Users do not need to agree on a basic communication framework [9] in order to understand a visual message. This reduces communication overhead [9] and keeps the "visual language" used for communication lightweight and flexible.

Combining classic hypertext and visual communication makes it possible to convey nodes and links via visual aids and therefore to visually communicate information structure. This results in an alternative representation form of hypertext, which is natural, lightweight and flexible. This new form of hypertext is called "Spatial Hypertext".

1.2 Spatial Hypertext

Implicit As already pointed out, people may find it difficult to use the formal node-link scheme [1, 4]. This is mainly because explicitly creating and linking formal nodes is something most people are not familiar with. However, people are accustomed to arranging objects in space [1]. A good example of this is our daily office work. We build heaps or piles of related papers on desk, we arrange cards on bulletin boards, write short notes on whiteboards or use a text marker to highlight important or associated parts in a document. In other words, we use our natural ability to perceive and associate objects to manage daily office work and thereby build visually perceivable networks of information objects. One could understand that as a physical implementation of hypertext. Spatial hypertext is based on that idea. Rather than explicitly connecting nodes by (potentially typed) formal links, as it would be done in node-link hypertexts, spatial hypertext expresses associations between information units implicitly by spatial and visual attributes (e. g., proximity, color, shape etc.) [1, 7]. Thus, as opposed to classic node-link hypertext where we define structure explicitly, the focus of spatial hypertext lies on *implicit* relationships [7].

Ambiguous A desired consequence of spatial hypertext's implicit nature (that is, of not defining links explicitly) is constructive *ambiguity* [3, 9]. "Ambiguous" means in this context, that spatial hypertext allows for being interpreted in multiple ways. Visual expression created by humans can be clear and explicit but may also leave some room for interpretation; without being invalid. This allows to create information structure even when users only have a vague or incomplete idea about what to express [10]. That way spatial hypertext supports the creation of *fuzzy* or *unclear* information structures [10].

Informal Spatial hypertext permits users to express *implicit* knowledge in visual information spaces. It thereby supports intuitive interaction with a visual medium and allows for direct manipulation of structure [6]. In spatial hypertext, modification of information happens immediately by changing spatial and visual properties and does not require formalisms [11]. There is no need to prematurely commit to language conventions for expressing certain types of structure [12]. Instead spatial hypertext allows the user to create instances of structures even before naming and typing them [10]. It thereby supports explorative development of emerging structures and visual languages [10] through informal interaction [12]. This *informal* nature of spatial hypertext has a great advantage: We do not run the risk of formalising information incorrectly or inconsistently. Formalization errors are usually difficult to correct [5] and make "ill-formalized" information more difficult to use than information not formalized [2]. In spatial hypertext we do *not* have this problem.

Emergent Since spatial hypertext expressions are based on language elements that are easy to change (i. e., it is easy to change a visual property or move an object), spatial hypertext supports expression of evolving interpretations [7]. It is therefore appropriate for information-intensive tasks where information continually evolves [7]; that is, for tasks which typically include collecting, analyzing, organizing and sharing of information [13]. The documents resulting from such tasks are usually under constant modification [13] and just rarely reach the state of a final product [7]. Usually, they are more an encoding of *evolving* lightweight structures [14] or rather "freewheeling" structures [3]. Spatial hypertext is therefore not only of implicit, ambiguous and informal but also of *emergent* nature [6].

1.3 Spatial Parsing

Having encoded information in spatial hypertext, human users can easily retrieve and decode it again. For a machine, however, it is not necessarily easy to detect such implicit and idiosyncratic structures. This is solved by using so-called *Spatial Parsers*.

Spatial parsers are (software) components which implement highly specialized structure detection algorithms. These detection algorithms are designed for retrieving information structures which are implicitly encoded in spatial and visual arrangements of objects generated by humans. We know these arrangements already as spatial hypertext.

To avoid misunderstandings it should be noted, that analyzing spatial hypertexts is no "parsing" in the classic sense of the term. In visual language processing "spatial parsing" is defined as "the process of analysing an input picture to determine its syntactic structure" [15]. So, "parsing" is synonymous to "syntactic analysis". This requires syntax definition, for example via picture layout grammars [15] or graph grammars [16]. Spatial hypertext, however, is of informal, implicit, ambiguous and emergent nature. Which means, that we do not know beforehand how words of the language are generated. Thus, premature definition of an adequate grammar is not possible. For sure we can define "some" set of production rules. Such a formal grammar, however, does not differ among different users [17] and cannot emerge [10]. Thus it will not cover all desired structural aspects of the visual language.

It is important to note, that in spatial parsing we do not assume that we can unambiguously recover syntactic structure [3, 14]. The purpose of parsing spatial hypertexts is not to "debug" formal spatial or visual expression [3]. The point is rather to uncover *some* partially-framed structure which are hidden in human-generated spatial layouts [14].

One could also say, that spatial parsers infer implicit structural knowledge from explicit spatio-visual knowledge and thereby turn implicit information structures into explicit ones [10].

Since this is more a matter of interpretation than checking for syntactical correctness, it should be considered to rather call such structure detectors “spatial interpreter” (instead of “spatial parser”). Anyhow, to be consistent with existing literature, we will use the term “parser”.

Why do we need structure recognition? When implemented and integrated into a spatial hypermedia system, a spatial parser could be used to trigger or support various high-level services. Such features include contextual search and intelligent navigation [6], but also user notification or structure suggestion [14].

Especially to be emphasised is a spatial parser’s support in multi-user environments (i. e., when multiple users collaborate on the same visual information space). By providing feedback about what the spatial parser (and therefore the system) has “understood” as structure, spatial parsing may identify ambiguous structures and suggest a common interpretation to all users [10]. Firstly, the users would become aware of the possible interpretations of the structures; secondly, they could efficiently agree on a common interpretation. This would avoid miscommunication based on contradictory interpretation of the spatial structure. Without such a shared understanding, the dialog between users about personal reflections and understanding of information becomes a difficult task and the hypertext material themselves are only of limited utility [10].

What requirements must be met by a spatial parser in order to support such advanced functionality? In [10] three core requirements were identified: (a) spatial parsers must be able to interpret spatial and visual relationships; (b) parsers must tolerate people’s different ways of expressing structure spatially and visually (where over-interpretation should be avoided) and (c) a spatial parser should extract only *intended* structure (i. e., parsers should “understand” what human users intend to express).

Especially the last requirement poses major challenges when it comes to practical implementation. A functional spatial parser must detect only *intended* structure, even though whether something is structure or not is a subjective matter of personal opinion and therefore depends on the user’s individual perspective [10]. In many cases we neither have information about authors and their personal motivation nor about context of application. Nevertheless the parser should be able to deliver (some) meaningful results. This is probably the reason why only little work has been done on practical spatial parsing.

Designing a spatial parser includes making several decisions. The most essential ones are (a) deciding what kinds of structure should be recognized and

(b) which spatial or visual attributes should be used for doing that. Since in theory all aspects of visual appearance could be crucial for understanding spatial structure [10], one might think, that detection of exhaustive explicit structure is only possible if a parser uses *all* aspects of what is expressed visually [10]. This, however, is not feasible. The same applies to the decision of what kinds of structure should be recognized. We cannot satisfy *all* individual perceptions of what might be structure or not [10], and therefore cannot detect "everything". However, what can be done is identifying *some* structure which is accepted by *most* users. This is why spatial parsers are typically (though not exclusively) built on *heuristics* for spatial pattern recognition [10,12]. Such heuristics are *ideally* valid across different contexts of application and cultural environments.

Structures in human generated layouts are formally elusive. This is mainly because they only become manifest in the user's thoughts and therefore only exist in mind [17]. This makes it difficult to discuss them. Nevertheless, it has been tried to identify several categories of spatial and visual structures.

1.4 Structures

According to [12] there are four common ways of noting relationships visually and spatially (Fig. 1.1): (a) relationships by spatial arrangement; (b) relationships by object type; (c) relationships by collection and (d) relationships by composition.

Spatial Arrangement Relationships by spatial arrangement are expressed via spatial proximity and alignment, which can be of different form [12]. A common, basic framework of such proximity- and alignment-based structure types may look as illustrated in Fig. 1.2: Structure types are split up into two main categories, *Ordered Groupings* and *Unordered Groupings*. Ordered groupings require visual uniformity of objects (i. e., the members of such groups must be of the same type) and they require a certain alignment of objects. As for unordered groupings, the opposite is the case: They may be built up of objects of different types and do not require a certain alignment of members. They are *unordered* [10].

Ordered groupings are, for instance, *Lists*, which exist in two different layouts: aligned horizontally, called *Horizontal Lists* and aligned vertically, labeled *Vertical Lists* [10,14]. Other ordered groupings are, *Matrices* [10,14] (or *Tables* [3]), where objects are aligned both, vertically and horizontally [3]. Therefore, matrices and tables can be seen as compositions of lists [14]. Finally, so-called

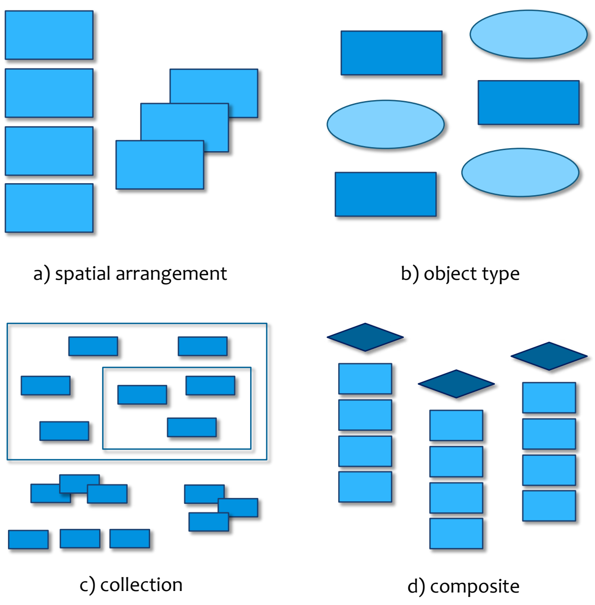


Figure 1.1: Four common ways of noting relationships visually and spatially [12]

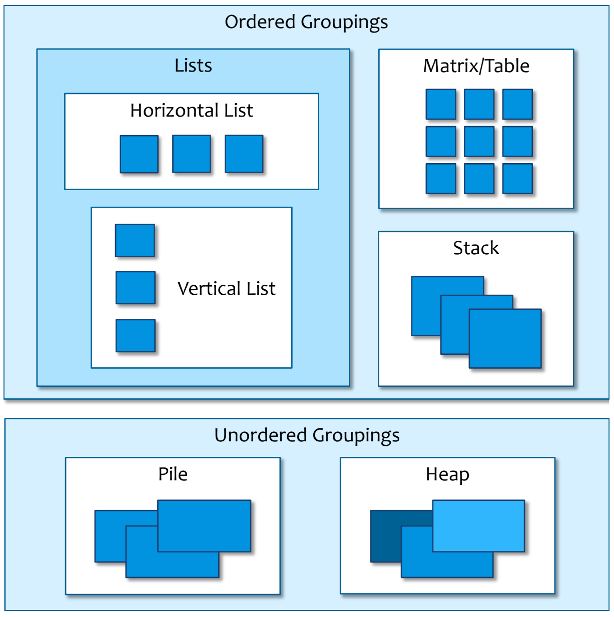


Figure 1.2: Basic spatial structure types [3, 10, 14]

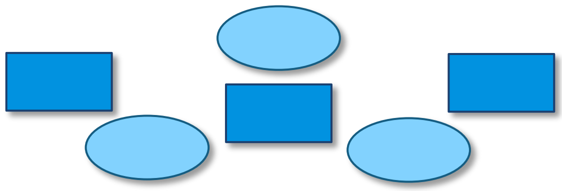


Figure 1.3: Relationships by object type [12] – sample layout with two visual categories of objects (position and alignment do not matter)

Stacks are ordered groupings as well [10, 14]. Stacks are defined as sets of significantly overlapping objects of the same type [14] that are aligned in some direction [10]. Thus, they may be considered as "compressed" lists.

Pile [10] and *Heap* [14] are unordered groupings. They do not require a certain alignment of objects [10]. However, what they do require, is, that objects are partly overlapping and have either the same type, this way they form a pile, or they have different types, which defines a heap [10, 14].

Object Type Relationships noted by object type (or by *visual* type) are categorical relationships expressed by visual similarities (see Fig. 1.3) [10, 12]. "Visually similar" refers to objects that have similar color, font, extent, shape, border width etc. – their position and alignment *do not* matter. Even if objects were randomly distributed over the screen, they still might be perceived to have some categorical relation if they share certain visual properties [10]. That way we can express "category membership, that *cross-cuts* spatial positioning" [12]. This type of structure can be seen as *Taxonomic Set*, as taxonomic sets are groupings in which all members belong to the same category [3].

Collections Categorical relationships between objects may also be noted through so-called *Collections* [12]. There are two different notions (see Fig. 1.4): (a) explicit and (b) implicitly defined collections.

Firstly, collections can be seen as explicit, regional selections of objects which reside in separate subspaces [10]. This notion supports the creation of subspace trees which then can express hierarchical ordering of collections [10]. This way it becomes possible to express hierarchical category structures [12]. Thus collections can be used to define taxonomies. Such distinct subspace trees, however, neither fit spatial hypertext's implicit nor informal nature. They are rather means to organizing artifacts and are therefore more an application feature than a visual language feature. Thus, one could argue whether this type of explicit structure has any relevance for spatial parsing. Nevertheless, hierarchical subspaces have become an integral part of several spatial hypermedia systems.

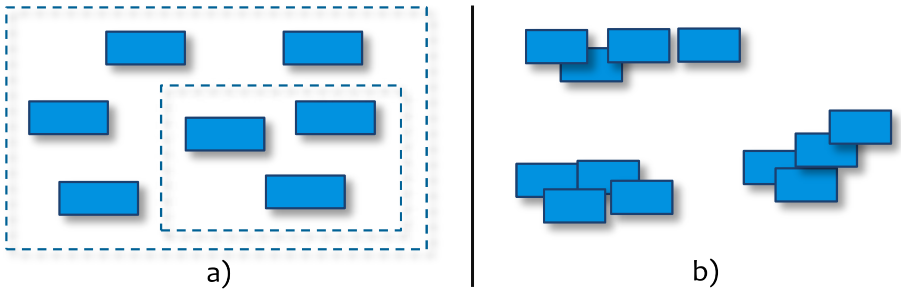


Figure 1.4: Relationships by collection [12] – two samples for explicitly and implicitly defined collections

Secondly, a collection may be seen as an implicit step by step selection of objects. When objects get moved into clearly, spatially separated clusters over time, they form distinct areas that emerge [18]. When objects located together in such an area also relate to the same task [10], they can be used to express *activity*-related relationships. This is why such implicit collections are usually called *Activity-Related Areas* [12]. One could argue now that these structures are nothing more than spatial clusters. Objects in the same area *may* refer to the same task but they do not *need* to. Just because two objects are located close to each other and are spatially separated from the rest of the hypertext, it does not necessarily mean that they refer to the same task. For sure they seem to have *some* relation, but the spatial hypertext does not "tell us" which one. The statement, that "objects in the same cluster relate to the same task" is just an assumption and nothing a spatial parser could verify. Thus from a spatial parsing perspective this notion of implicitly defined collections refers more to spatial clustering than to activity realted working areas. Spatial clusters fit perfectly into the category of *Unordered Groupings* (since they are proximity-based and do not require a certain alignment of member objects). So one can discuss whether it does not make more sense to treat collections as "relationships noted by spatial arrangement".

Composites The last category of relationships to be discussed are noted by composition (see Fig. 1.5). The key element here are so-called *Composites*. According to spatial hypertext literature composites are defined as lightweight, abstract, recurring, regular spatial patterns of different object types [14, 19]. They are spatial combinations of two or more instances of different visual types, where each object occupies a known position in the structure [3, 12]. Therefore, composites rely on both relative spatial positioning and the ability to distinguish between different types of objects [3]. For instance, labeled vertical lists [14, 19] can be defined as composites, comprising of vertical lists with header symbols (an example is illustrated in Fig. 1.5). Using composites it becomes possible to express relationships between different types of objects [10].

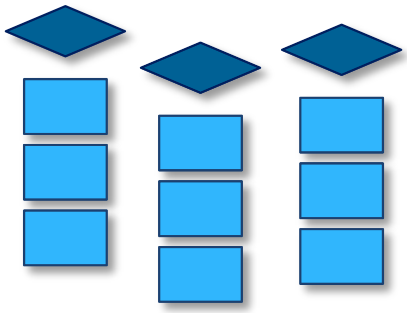


Figure 1.5: Relationships by composition [12] – sample layout with three instances of labeled lists (i. e., compositions of vertical lists and header symbols)

Primary vs. Secondary In addition to classifying structures by their spatial and visual appearance, as we did in the last section, one can also categorize them by their expressive power and their visual effects for the viewer. Depending on how easily structures may be visually separated from each other and therefore how easily they can be identified as discrete meaningful units, we can distinguish between (a) primary and (b) secondary structures.

Primary structures typically get noticed sooner by viewers than secondary structures. Thereby they get recognized as the structural "backbone" of a visual structure and "guide" the viewer through the interpretation process. Secondary structures rather build on top of primary structures and are often used for refinement only (i. e., they express nuances of primary structures [20]).

One could argue now (as the findings in [20] suggest), that spatial properties (e. g., proximity, alignment etc.) are primary means for describing structure, whereas color, shape, opacity etc. are secondary aspects of visual expression. Following this argumentation we can classify all previously defined spatial structure types (incl. lists, stacks, spatial clusters etc.) as primary structures. Taxonomic sets (expressed by visual similarity), however, overlap spatial structure and thereby link structure elements via tacit cross references [12]. Thus they can be regarded as secondary structures.

The sample layout in Fig. 1.6 illustrates that. Fig. 1.6 shows a horizontal alignment of four vertical lists; each comprising three colored rectangular objects. One can understand that table-like layout of rectangles as an expression of the primary aspects of the overall information structure. The two different colors assigned to rectangles only extend that "master"-structure by tacit cross references between list elements and thus cover secondary aspects of the structure.

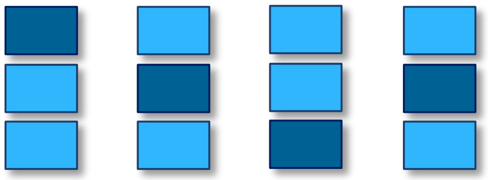


Figure 1.6: Sample layout – primary vs. secondary structures

1.5 Implementations

Pioneer implementations of spatial parsers were inspired by obervations of how people use map-centered hypertext systems. One of those systems was Aquanet, a hypertext system that allowed users to work with geometrical and textual information objects in a (shared) information space [21]. Experiences with Aquanet [18] showed, that people prefer spatial representations of hypertext over more explicit and formal models [3], and thus put particular importance on implicit expressions [14].

In order to get a better understanding of such human-generated spatial layouts, a survey was conducted in both, computational and non-computational settings [3, 14]. Within the scope of that study researchers examined several spatial hypertexts created in three different systems [3]: (1) NoteCards [22]; (2) the Virtual Notebook System (VNS) [23] and (3) Aquanet [21]. Having analyzed those sample hypertexts, each the result of a long-term information management or analysis task, it turned out that spatial layout and visual properties allow to identify specific *types* of structures (such as, horizontal and vertical lists, taxonomic sets etc.) [3].

Based on these findings, and the results of analogous studies of the way people organize materials [24–26], an early "Heuristic Structure Recognizer" [3] was developed and tested. First experiments using that prototypical recognizer led to the conclusion, that automatic detection of implicit structure is feasible, and that it is therefore a worthwhile subject of further investigation [3]. This finding was one of the main driving forces for integrating a spatial parser into the VIKI system [12].

1.5.1 Spatial Parsing in VIKI and VKB

Instead of tightly coupling functionality in a more monolithic implementation, VIKI's spatial parser realised its structure detection functionality via composition of and delegation between independent structure recognition modules [14].

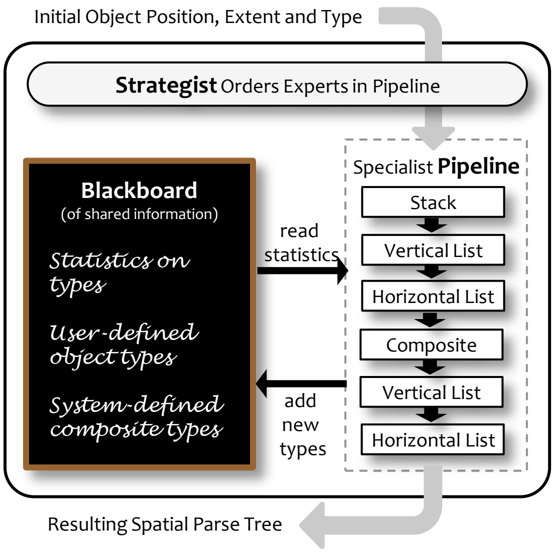


Figure 1.7: VIKI's spatial parser – architecture for spatial recognition [14]

This modular and thus flexible architecture (Fig. 1.7) comprized three core components: (1) a reconfigurable *Pipeline* of spatial structure recognizers ("specialists"); (2) a *Blackboard* of shared information and (3) a so-called *Strategist* used for setting up the pipeline [14].

The "heart" of the VIKI parser was its *Pipeline* of specialists. Specialists were heuristics-based software components, responsible for identifying particular types of structures. Those modules used the spatial display and its current parse as input and produced a new parse as output. Thus, each specialist was a small spatial parser in itself. VIKI included specialists for detecting both, homogeneous structures, such as lists (horizontal and vertical) and stacks, but also heterogeneous structures, such as heaps and composites. Attributes used for recognizing these structure types included position, extent and object type. [14]

Each specialist produced output which could be used as input for other specialists. By doing so, specialists got linked together in a chain, or a pipeline [14]. Pipelining different instances of (not necessarily different) specialists together, has two desirable effects: (a) a single specialist can operate multiple times (even though we perform only a single pass) [14] and (b) already detected structures can be re-parsed, which makes it possible to identify higher level structures [3] (i. e., the pipeline implements a bottom-up parsing strategy [14]).

In addition to communicating via input and output, specialists shared global information via the *Blackboard*. This included information, such as object types (defined by the user), composite types (defined by the system) but also statistics on the use of each type recorded in the blackboard. Specialists for heterogeneous structures created new (system-defined) types, added them to the blackboard and updated usage statistics (in order to reflect the new structures). Specialists for the detection of homogeneous structures did not create new blackboard entries and used the types of their input objects instead. [14]

The third and last component in VIKI's parsing architecture was the so-called *Strategist*. The strategist‘s main task was to detect the optimal order in which specialists should be applied (i. e., the parsing “strategy”). This was accomplished by performing some initial statistical analysis of object alignments. The resulting dominant orientation of the spatial layout was then used to determine the order of specialists in the pipeline. [14]

One assumption of VIKI's parser was the definition of user-generated types. It was assumed, that atomic information objects were given unique and meaningful types by the user. Practical experience with VIKI had shown, however, that often times users did not define such types [11]. This limited the spatial parser's effectivity, since VIKI, by default, treated untyped objects to be of the same type [7].

This problem was solved in VIKI's successor VKB [7], by patching in a visual preprocessor before the spatial parser. That preprocessor had the function to automatically assigning types to each object which did not have a user-defined one. This was realized by implementing a heuristic type assignment algorithm building on the following simple rule:
Objects which have similar visual attributes (incl. width, height, background and border color, and border width) are considered to have the same type, whereas visually dissimilar objects are considered to be of different types [7].
Using this rule to automatically assigning visual types to objects, the preprocessor did not only support the spatial parser in interpreting ambiguous layouts, but also decreased the workload for the users [11].

A more recent alternative to this very early implementation of a spatial parser is FLAPS [11], which was (among other applications) also integrated into VKB.

Unlike the spatial parser described in [14] and [7] respectively, the FLAPS-parser is a fuzzy-logics-based, multi-pass algorithm, which generates so-called “containment graphs” instead of simple parse trees [11]. Thus, what differentiats FLAPS from previous parser implementations, is (1) how parser output is represented and (2) how core structures are detected.

Previous spatial parsers tried to resolve ambiguity by determining the *best* interpretation and discarding all others [11]. This procedure resulted in distinct containment hierarchies of visual structure elements (i. e., parse *trees*). Parser output created by FLAPS, however, are graphs, where each node can have *multiple* parent nodes [11]. Such "containment graphs" may therefore include cycles. Thus, FLAPS produces no conventional parse trees as output.

Just like previous parsers in VKB, also FLAPS uses independent structure recognizers for piles, vertical lists, horizontal lists etc. The crucial difference is that recognition specialists in FLAPS are based on fuzzy-logic. FLAPS' structure recognizers consist of fuzzy rules which can be processed by a fully-functional fuzzy-logic inference engine. Object attributes analyzed when doing such inferences, include (a) proximity and regularity and (b) visual similarity of objects. Proximity and regularity are determined by alignment and relative distance between objects. Evaluation of object similarity depends on visual characteristics (e. g., color, size etc.) and object type (e. g., image, text etc.). This fuzzy-logics-based approach makes such recognizers particularly well suited for recognizing ambiguous structures. [11]

FLAPS' main parse procedure includes the following four steps: (1) recognize as many structures as possible (by passing the whole space to every recognizer); (2) automatically merge alternative interpretations into a single coherent structure; (3) repeat this procedure considering the recently detected structures and (4) keep doing this as long as further structures can be recognized [11].

For automatic merging FLAPS follows a minimalistic approach, where different merging strategies are chosen, depending on whether two structures exclude, intersect, fully overlap or are identical. When two interpretations exclude or intersect, then no special merging action is required, since FLAPS can deal with both cases. When one structure is a subset of another (i. e., when one is contained by another), then the merging operation "absorbs" the subset structure and updates the relationship graph accordingly. Finally when two structures were assigned different types but have exactly the same elements, they get joined into a new structure with a compound type (i. e., a new type automatically generated from both individual types). [11]

Both VIKI's and VKB's original spatial parsers belong to the most important and best know parser implementations in the spatial hypermedia domain. Both broke ground for a completely new way of visual analysis: *Spatial Parsing.*

Nevertheless, in addition to those pioneer implementations, there are further parsers used in other systems than VIKI and VKB. Some of them shall be mentioned in the following sections.

1.5.2 Adaptive Spatial Parsing

As already pointed out, interpreting spatial hypertexts is a highly subjective issue. Whether something is perceived as structure or not strongly depends on a person‘s individual perspective [10]. However, users' individual ways of visual thinking are usually unknown to a spatial hypermedia system. Nevertheless, a functional spatial parser still should be able to "understand" what a user intended to express by a certain arrangement of information objects. This poses great challenges for designers of spatial parsing algorithms.

One possible (though not the best) strategy for approaching this problem, is adapting a spatial parser to its users. This could be accomplished, either by tailoring a parser *manually* to particular user requirements (e. g., via customized implementation or configuration), or by adapting a parser *automatically* (e. g., via supervised or unsupervised learning). While having the disadvantage of tying together parsers and user profiles (which undesirably limits flexibility), this adaptive approach still provides a possible solution to the aforementioned "subjective perception"-problem.

The spatial parser design described in [17] is based on that idea. Building on the work presented in [3, 14], Igarashi et al. [17] developed an adaptive parser which uses explicit user feedback to self-adjust to users' expectations. Their parser has the ability to "learn" from a user's corrections of (subjectively) wrong interpretations made by the system and thereby supports automatic adaption.

Conceptually the parser comprizes of two features: (1) a *Link Model*, which forms the basis for the parsing process and (2) *Automated Parameter Tuning*, which facilitates the parser in "learning" from users [17].

The core element of the so-called *Link Model* is a graph. Each vertex in that graph symbolizes an information object and each edge represents a candidate relationship. The main task of the structure recognition procedure is to evaluate such candidates by their strength of connection and to reject those links which are not "fit" enough to be part of an intended structure. [17]

This basically works as follows: In a first step links are created between adjacent objects. This link creation process happens in three steps: (1) sorting; (2) finding candidates and (3) link creation. First, all objects get sorted by their upper left x-coordinate. Then, for each object, all proximity candidates are identified (again by comparing x-coordinates). Finally, links to the closest neighbors are created (i. e., links to the nearest candidate in all eight directions). This link creation procedure gets triggered everytime an object was moved on the screen and so the neighborhood of some objects must have changed. The following steps are executed only on demand. [17]

In the next step the strengths of links are calculated. This strength calculation process includes two phases: (1) primary strength calculation and (2) interaction process [17].

Initially, primary calculations are done for each link. This includes calculation of a strength value and a list-level value. The strength of a link is defined as a function of the distance between two objects, and the list-level (i e., the list likeliness) is a function of both, distance and the x-/y-gaps between objects. Both functions (strength and list-level) can be adjusted by the adaption process, which is discussed afterwards. In addition to these two values, each link is also assigned a type. If the previously calculated list-level value is non-zero, then the link is typed as “list-link” (horizontal or vertical respectively). Otherwise the link type is set to “cluster-link”. [17]

The second part of the strength calculation is based on interactions among links and includes two revision cycles. In the first cycle, links which probably do not link list elements together (because they are conflicting with links between surrounding objects), are assigned a decreased list-level value. For these purposes a parameterized factor is applied. In the second cycle, strength values are re-calculated through “repression and reinforcement” (i. e., adjacent links with different types repress each other and adjacent links of the same type reinforce each other). This has the effect of strengthening links between objects that are located close to each other or are regularly aligned. [17]

Having calculated how strong connections of certain types are, it becomes possible to use those strength-indiators for filtering out links which most likely belong to a structure. This is done as follows: First, the stronger (or most stable) links are selected. Then, objects connected by these links are grouped together in result candidates. The only thing that remains to be done is to decide on the (primary) type of these groupings (i. e., vertical list, horizontal list or just cluster). This decision is based on the number of links belonging to each structure type. The type that has the most links in a structure gets selected. [17]

It becomes obvious, that the link model in [17] differs fundamentally from the “pipeline of specialists”-approach presented in [3, 14]. Shipman et al.’s algorithms focus on the recognition of distinct classes of structures, and are particularly well-suited for analysing layouts which resulted from meaningful human activities, such as document triage. The link model in [17], on the other hand, targets more flexible and ambiguous layouts that do not include the notions of stacks, heaps, composites etc.

What also sets this parser apart from known solutions in VIKI and VKB, is its ability to adapt automatically to user expectations. The parser’s second essential feature besides the *Link Model* is called *Automated Parameter Tun-*

ing and allows the parser to "learn" from explicit user feedback. This tuning mechanism is based on search heuristics and works interactively. [17]

Building on genetic algorithms, the tuning mechanism operates on evolving generations of individuals. Individuals (or candidate solutions) lie in a configuration parameter space and represent potential parser configurations. Each of these candidate configurations holds a set of parameters and a score. The score indicates how well the parser would perform (in analysing user-generated training samples), if the respective configuration parameters were applied. This "fitness"-indicator is given by a fitness function (or evaluation function), which checks how close a given training parse result comes to the respective sample structure. By statistically selecting solutions that are "fit" enough and by applying operators, such as mutation and crossover, the tuning mechanism realises a search routine for finding "optimal" parser configurations. [17]

Combining this heuristic parameter tuning mechanism with an interactive user interface allows for incremental refinement of a spatial parser's configuration, without the need to define any numerical parameters. This is an attractive alternative to the cognitive complexity of tuning a set of configuration parameters manually. [17]

1.5.3 Shared Spatial Parsing

The spatial parser developed by Reinert et al. [10,27] targets a different aspect of spatial hypermedia: *Collaboration.*

Amongst other benefits, spatial parsers can be used to resolve inconsistencies in structural interpretation and can be therefore especially helpful in collaborative environments. For that reason Reinert et al. implemented an incremental parser whose parse results can be stored persistently and shared among different collaborative users. Their *shared* spatial parser was designed as an *open* parsing service provided by an open and collaborative spatial hypermedia system, called CAOS [10,27].

Conceptually the CAOS-parser builds on VIKI (Sect. 1.5.1) and the algorithm discussed in the last section (Sect. 1.5.2). In addition to recognizers for lists and heaps the CAOS-parser also implements heuristics for the detection of matrices; which is (semantically) not supported by VIKI and Igarashi et al.'s implementation. Stacks, composites and clusters are not part of the design. [10]

Measures used for inferring these types of structures include indicators for proximity, alignment and extent, but not for visual similarities. As for proximity there is only a single measure: the absolute distance between two objects in two-dimensional space. When used together with a distance threshold that indicates how far apart objects can be from each other to still have a structural relation, this measure allows for identifying spatially separated structures. That thresold value is defined by two configuration parameters (one parameter for each dimension). Alignment, on the other hand, is decided by analysing position deviations in both dimensions. The position used for this is the upper left corner of an object's bounding box. The maximum allowed variances in both directions are given by configuration parameters again. [10]

The CAOS-parser implements an event-driven, bottom-up algorithm. It is event-driven insofar as the parsing procedure gets triggered by editing events (i. e., manipulation of objects). Thus, single parser runs are not performed on demand but whenever spatial structures might have changed. When started, parsing happens in a bottom-up fashion. This means that the algorithm tries to build more complex structures out of simpler ones. By doing this in a repetitive fashion the parser can identify high-level structures (e. g., lists of lists). [10]

Like VIKI also the CAOS-parser uses configurable structure experts to recognize certain types of structures. There are four experts called in the following order: (1) heap expert; (2) horizontal list expert; (3) vertical list expert and (4) matrix expert [10].

The heap expert is invoked first. During execution, the following happens: Firstly, all objects in the information space are sorted according to their x-coordinate. Once that is done, all pairwise combinations of objects are checked for overlapping. When two objects overlap, the ratio between overlapping area and object extent is calculated for both objects. If one of those values exceeds a given limit, the heap expert treats the two objects as heap elements. Here the expert pursues the strategy of making heaps as big as possible. [10]

Detection of heaps is followed by list recognition. The expert for recognizing horizontal lists basically proceeds as follows: In a first step all objects get sorted by their upper left x-coordinate. This allows for horizontal iteration over all objects in the information space. In a second step, each object is assigned an itersection box. Position and extent of such a box depend on list alignment and proximity parameters. In the horizontal case an intersection box touches the upper right corner of an object. The expert then iterates through all objects from left to right and performs intersection checks between intersection boxes and upper left corner points. When two objects intersect they can be regarded as list elements. In this case the expert either builds a new list starting with these two objects or a previously detected list gets extended (where lists are build as long as possible). Objects identified as list elements are skipped in further passes. Basically, the vertical list expert does the same. The only differences lie in the relative positioning of intersection boxes (i. e., in the vertical case they would touch the bottom left corner of an object) and the coordinate used for sorting (i. e., in the vertical case it would be the upper left y-coordinate). [10]

The matrix expert, which is called last, turns lists of lists into matrices and therefore depends on the list experts. Thus, list experts always must be called before the matrix expert. Its working principle can be described by the following three steps: (1) look for matrix candidates; (2) verify candidates as matrices and (3) substitute spatial objects. Firstly the matrix expert iterates through all objects in the information space and identifies potential matrices. Such a matrix candidate is either a horizontal list of vertical lists, or a vertical list of horizontal lists. In both cases element lists must be of the same length. In the second step candidates are reparsed to check for correct alignment. This is done by checking, whether a vertical list of horizontal lists could also have been parsed as horizontal list of vertical lists. When this is the case, the expert substitutes the respective list of lists for a new spatial object that represents a matrix structure. [10]

One basic assumption of the CAOS-parser is, that spatial hypertext simulates desk work and is therefore conceptually limited to vision in two-dimensional space. The same applies to Igarashi et al.'s adaptive spatial parser (Sect. 1.5.2) and parser implementations in VIKI and VKB (Sect. 1.5.1). This, however, does not necessarily have to be the case. Spatial hypertext implementations may go far beyond the classic "deskwork-paradigm". Working with spatial hypertext can be more than simply arranging papers on a 2d-canvas.

1.5.4 Three-Dimensional Spatial Parsing

Nielsen and Ørbæk [28, 29], for instance, have explored spatial hypertext in three-dimensional spaces. In [28,29] they described a spatial parser for TOPOS, a prototypical information management system which supports informal groupings of information objects in a three-dimensional space.

Nielsen and Ørbæk argued, that there is only little difference between subjective and objective placements of objects in a two-dimensional space. Which means, that the difference between perceived structure and physical structure is fairly small when the information space is limited to two dimensions. The reason for that is, that spatial hypermedia systems that implement two-dimensional information spaces allow users to view spatial structures from a fixed perspective only. Although the information space can be navigated via zooming and panning, the viewing angle is always the same. [28, 29]

This is not the case in a three-dimensional environment. There, camera position and orientation can have significant influence on how proximity and proportions of objects are perceived. In other words, depending on the viewing angle, groupings of objects may look completely different. For this reason it does not make much sense to parse such a space from an objective global view point. The resulting structures would much too often confuse users. Instead three-dimensional parsers should analyze the scene from a user's (subjective) viewpoint. This can be accomplished by providing the parser with the user's current viewing frustum. Nielsen and Ørbæk's parser implementation is based on that idea. [28, 29]

Although being designed for usage in a three-dimensional context, the core parsing algorithm was mainly inspired by VIKI's (Sect. 1.5.1) and Igarashi et al.'s spatial parser (Sect. 1.5.2). For that reason the TOPOS-parser uses Igarashi et al.'s link model (in a slightly modified and extended form, called "proximity model") combined with Shipman et al.'s concept of configurable structure experts (for recognizing clusters and arbitrarily oriented lists). Automated parameter tuning via evolutionary algorithms, however, was not implemented. [28, 29]

Contrary to parser implementations in VIKI and VKB, the TOPOS-parser does not search the space for well-structured layouts and rather focuses on the recognition of (rough) groupings of objects [28, 29]. It therefore operates on a higher level of abstraction. While being conceptually valid, this approach is rather inappropriate in more specialized contexts of application, such as linear document authoring or movie editing. This is why more specialized parsers typically do not build on such "general purpose"-algorithms. Instead they implement algorithms which are tailored to specific application purposes and thus operate on a lower abstraction level. Good examples for that can be found in the work by Yamamoto et al. [30–32].

1.5.5 Specialized Spatial Parsers

Yamamoto et al. [30–32] developed a series of spatial hypermedia systems (labeled as "ART"-systems), that support early stages of information authoring. Such information authoring includes writing research articles or books, preparing multimedia presentations or editing movies. Systems in the "ART-family" use spatial representations not as a medium for representing final artifacts but as a means for authoring linear, hierarchical, and network structures. [31, 32]

Spatial parsers can be used in ART-systems to convert such structures into sequential texts, slides or video clips. Thus, the focus of an ART-parser is not on identification of certain structure types (such as, stacks, clusters etc.), but on *translation* of spatial information into more explicit textual or multimedial artifacts which can be used in an ongoing authoring process. [30–32]

Currently there are no parser implementations available for translating hierachical arrangements and network structures [32]. However, ART-systems support the conversion of spatially aligned objects to linear text documents. Thus, parsing of linear structure is possible [30].

The respective spatial parser simply scans all objects on the screen sequentially from top-to-bottom or from left-to-right (depending on its configuration) and thereby serializes their textual content [30, 31]. Other spatial and visual features, such as shape, size, color etc., have no influence on the parser's decisions [31]. Although being an apparently simple approach to spatial parsing this linear structure recognizer still fulfills its application purpose.

Finally, it should be noted, that expressing structure via arbitrary arrangement of objects, as known from other systems, is not a use case in ART. Unlike other information spaces, the two-dimensional ART-space has pre-assigned semantics and is therefore more a feature interface than a "free" information space [31]. This can be largely ascribed to the working principle of the spatial parser: If a user intends to use the parser for auto-generating documents from spatially aligned text snippets, then he must follow predefinded language conventions. In concrete terms, an object's relative position in 2d-space is always interpreted as its position in the respective document sequence. Interpretation of *arbitrary* arrangements of objects is not supported. Thus, it can be argued whether the "ART-family" of authoring systems really implement spatial hypertext. Nevertheless, we regard the ART-parser as specialized spatial parser.

A last example for a specialized spatial parser is given in [33], where spatial neighborhood detection in *Web Squirrel* is presented. Neighborhoods in Web Squirrel are groups of items clustered around labels. Such labeled clusters can be nested inside each other and thereby support the creation of hierarchical category structures. The size of such neighborhoods is decided by a configurable spatial parser which implements the following algorithm:

First, the neighborhood boundary of each neighborhood label M is set to the label's *Vicinity*. The *Vicinity* of an object is defined as the object's bounding rectangle, scaled by a given factor (e. g., 1.5). Then, for each object whose *Vicinity* intersects a neighborhood boundary, and for each neighborhood label having a smaller fontzise than M the following is done: that item or neighborhood is added to M and M's neighborhood boundary is updated to the union of former bounds and the *Vicinity* of the newly added object. This gets repeated either until all items have been assigned to some neighborhood, or until the *Vicinities* of all remaining objects lie outside of any neighborhood. [33]

Chapter 2

Formal View

The last chapter was supposed to serve as both, an introduction into the field(s) of spatial hypermedia and spatial parsing and as an overview of relevant literature. It was shown how spatial hypertext and spatial parsing are informally defined (Sect. 1.2 and Sect. 1.3) and which practical solutions to the parsing-problem were developed (Sect. 1.5).

In summary, we can define spatial hypertext as an alternative representation form of traditional hypertext, which is natural, lightweight and flexible. In contrary to classic node-link hypertext, spatial hypertext expresses associations between information objects *implicitly* by spatial and visual attributes [1, 7], supports constructive *ambiguity* [3, 9], does not require premature (*formal*) language definition [12], and supports expression of *evolving* lightweight structures [7, 14]. In short, spatial hypertext is of implicit, ambiguous, informal and emergent nature (Sect. 1.2). Information structures which are implicitly encoded in such human-generated arrangements of objects are detected by heuristics-based (software) components, so-called spatial parsers. Spatial parsers infer implicit structural knowledge from explicit spatio-visual knowledge and thereby turn implicit information structures into explicit ones. In short, the process of parsing spatial hypertexts can be defined as retrieving hidden information structures from human-generated layouts (Sect. 1.3).

Although being accepted within the spatial hypermedia community, this view on spatial hypertext and spatial parsing is purely *informal.* Nobody has tried yet to define both spatial hypertext and spatial parsing *formally.* Which means that there is *no* common theoretical, mathematical basis yet, which could be used to approach the parsing problem more systematically. This chapter is intended to change that.

2.1 Spatial Hypertext Languages

As pointed out already, spatial hypertext can be seen as an alternative representation form of traditional hypertext. Rather than making information structure explicit by defining formal nodes and links, spatial hypertext targets at expressing and communicating implicit structure via visual aids (e. g., shape, color etc.). Visual expression and communication of implicit structural knowledge requires an appropriate visual notation. Such a visual language must be compatible with the core characteristics of spatial hypertext (Sect. 1.2) and therfore has to be lightweight, flexible and intuitive. We denote such visual notations as *Spatial Hypertext Languages.*

Spatial hypertext languages are informal, visual languages for implicit structure representation. Expressions in such a language are collections of visual features that describe structure but, as opposed to visual programming languages, do not specify behaviour. Spatial hypertext languages are not intended for specifying dynamics and are rather used for describing static networks of information units. In a general sense, spatial hypertext is structure expressed in a visual representation language.

From a formal point of view, one can see a language as a *set* of (meaningful) structures, which are called *words.* Words are constructed from *symbols* and the set of symbols from which the words of the language may be formed is known as *alphabet.* This simple, set-theoretical definition applies not only to string-based languages, where words are viewed as one-dimensional sequences of symbols, but also holds true for visual languages. Also visual languages can be defined as sets of words constructed from symbols. The crucial difference, however, is that (1) "visual symbols" are graphical rather than textual and (2) "visual words" are diagrams rather than strings. A good example for that are languages defined by picture layout grammars [15].

Following this idea, we can define spatial hypertext languages as sets of spatial hypertext *artifacts* (i. e., words), where each artifact is nothing more than a flat collection of spatial hypertext *symbols.* We use the term "artifact" due to the emergent character inherent to spatial hypertexts (Sect. 1.2): that is, arrangements of information objects that emerge in the course of an information analysis task are rather intermediate results (or steps) of a development process than just self-contained expressions in a particular language. For this reason, we consider the term "artifact" as being more appropriate than simply calling elements of a spatial hypertext language "words". Thus we define spatial hypertext languages as sets of spatial hypertext artifacts. Such artifacts are sets of spatial hypertext symbols, and symbols are collections of spatial or visual properties (such as, position, shape, color etc.). Symbol properties are represented as instances (or elements) of attribute *types*, which are sets of attribute *values*. The definition of such types forms the visual character of a spatial hypertext language.

Spatial examples for such attribute types include position, size, orientation, but also shape. We define “shape” as all the geometrical information of an object that is invariant to changes of location, orientation and size. Visual attributes, on the other hand, may include line settings (such as line style, color, thickness etc.), surface properties (such as fill color, textures etc.), but also opacity or transparency values. The concrete definition of such spatial and visual attribute types strongly depends on the visual user interface of a spatial hypermedia system. One system, for example, might allow for arbitrary placement of objects in $\mathbb{R}^2$, whereas another one holds objects in a discrete two-dimensional grid. Although both systems support spatial arrangement of objects they still implement different definitions of *position*: the first one builds on real-valued vectors, whereas the latter one assignes objects to cells in a grid. Thus, in theory we can identify several *classes* of attribute types which most likely belong to a spatial hypertext language (such as: position, size, orientation, shape, color etc.). The exact definition of these sets of attribute values depends on concrete system implementation.

Another point to be noted, relates to the definition of well-formed words (or well-formed artifacts). As pointed out already in the last chapter (Sect. 1.3), we cannot know how words of a spatial hypertext language are generated *before* the hypertext gets modeled and the visual language has emerged. Thus premature definition of an adequate formal grammar is not possible. We therefore need an alternative approach to grammatical construction in order to formally define spatial hypertext.

Here we benefit from the set-theoretical view on formal languages, that was menioned before: If we define a formal language L as *some* set of words over an alphabet Σ and if we denote the set of *all* words over the same alphabet as Σ^*, then L effectively is a subset of Σ^* (i. e., $L \subseteq \Sigma^*$). Such a subset could be defined now in two ways: either by explicitely listing or describing all elements that belong to the set of well-formed words (e. g., by using a formal grammar, an automaton etc.) or by *excluding* all words from Σ^* which do *not* belong to the language. The latter approach allows for an easy, formal approximation to languages with an unclear syntactic structure. Spatial hypertext languages are good examples for that. This will become clear from the following details:

As designers of spatial hypermedia systems we may not be able to predict exactly which artifacts might be regarded as being well-formed, but we usually know which hypertexts are uncommon or simply not possible. When designing spatial hypermedia interfaces two fundamental decisions must be made, (1) which spatial or visual properties should be used and (2) which constraints should be put on visual information objects.

First of all, the designer must specify relevant spatial and visual attribute types. It must be defined which classes of properties should be supported by the visual interface (e. g., position, shape, size etc.), and it must be decided how to implement these features. This implicitly defines the set of all visual symbols that can be formed; that is, the spatial hypertext "alphabet".

Secondly, the designer must decide on potential constraints on symbols and symbol combinations. It must be defined whether to prohibit certain attribute value combinations (e. g., via respective rules in a symbol editing dialog), or if particular symbol combinations in the workspace should be avoided (e. g., by activating or deactivating gui components that are used for symbol creation). This way the system designer determines which symbols are allowed and may be put together in the same artifact. The designer thereby limits the number of artifacts which can be built and effectively defines the spatial hypertext language that is available via the user interface.

That language can be further refined by the user of the system. In the course of creating, modifying and deleting symbols, users select subsets of attribute type definitions (e. g., by choosing only black and white as fill-color) and tend to avoid certain attribute value and symbol combinations (e. g., black spheres are never used together with white rectangles). By adding such additional constraints on symbols and attributes, users implicitly refine an interface language into "their language" (which is the language currently in use).

So, in practical implementations both can limit the range of a spatial hypertext language, system designers and users. Designers impose restrictions on visual interfaces and users of such interfaces choose those features that might be appropriate for solving a certain problem. Insofar we do not fully agree with [34] stating that such a visual language is "not prescriptive or restrictive". To some extent *every* spatial hypertext language is restrictive. Spatial hypertext languages are always bound to system implementations and are therefore limited by system design. In other words, there is *no* spatial hypertext implementation which allows for "total freedom" in expression.

Although this might sound negative, it still allows us to define sets of spatial hypertext artifacts which most likely do *not* belong to a certain spatial hypertext language. By excluding these words from the full set of artifacts, we can *approximate* a spatial hypertext language as follows:

$$L_{SH} = \Sigma_{SH}^{*} \setminus X_{SH} \tag{2.1}$$

Here, L_{SH} is a symbolic placeholder for a single spatial hypertext language (i. e., a set of spatial hypertext artifacts), Σ_{SH}^{*} is the set of *all* artifacts which can be build from given attribute type definitions, and X_{SH} is an e$\underline{\text{X}}$clusion set comprising all spatial hypertext artifacts not part of the language L_{SH}. According to this definition, X_{SH} must be a subset of Σ_{SH}^{*}. We decided that

X_{SH} should be a proper subset, and therefore may not be equal to Σ^*_{SH}. The reason for this is, that we consider empty spatial hypertext languages (such as $L_{SH} = \left(\Sigma^*_{SH} \setminus \Sigma^*_{SH}\right) = \emptyset$) as being of no practical relevance. One could even argue, that such an empty set of artifacts is *no* spatial hypertext language anymore, since the spatial aspect (in the form of spatial arrangements of objects) is completely missing. Only expression of space (even if it is just an empty information space) forms a spatial language. Otherwise it would be nothing more than just *some* empty language. Thus we define:

$$X_{SH} \subset \Sigma^*_{SH} \tag{2.2}$$

We also assume, that the empty spatial hypertext artifact H_ε (which is just an empty set of spatial hypertext symbols), is not an element of the exclusion set X_{SH}. This is intended to guarantee, that the empty artifact H_ε does never get excluded and is therefore an element of *any* spatial hypertext language. Thus we can be sure that H_ε is always a well-formed artifact. Given that $H_\varepsilon = \emptyset$, we can extend our definition from Eq. 2.2 to:

$$X_{SH} \subset \Sigma^*_{SH},\ H_\varepsilon \notin X_{SH} \tag{2.3}$$

The second fundamental component besides the exclusion set X_{SH} is Σ^*_{SH}. Σ^*_{SH} is defined as the set of all spatial hypertext artifacts which can be generated based on a given spatial hypertext "alphabet", which we denote as Σ_{SH}. Even though this definition was inspired by the *Kleene*-hull Σ^* over an alphabet Σ, it is mathematically not exactly the same. This will become clear from the following details:

Assume that we have $n \geq 1$ Attribute types $A_0, A_1, \ldots, A_{n-1}$. Each A_i shall be defined as a *non-empty set* of attribute values (i. e., $A_i \neq \emptyset$ for $0 \leq i \leq n-1$). Then we can define the *non-empty* set of *all* spatial hypertext *symbols* which may be formed from elements out of $A_0, A_1, \ldots, A_{n-1}$ as:

$$\Sigma_{SH} = (A_0 \times A_1 \times \ldots \times A_{n-1}) \tag{2.4}$$

According to this definition, the "alphabet" Σ_{SH} represents the set of *all* attribute value combinations (i. e., n-tuples of attributes) which could be formed from n predetermined attribute types. Our definition of Σ^*_{SH} builds on this n-ary cartesian product as follows:

$$\Sigma^*_{SH} = \{H | H \subseteq \Sigma_{SH}\} = 2^{\Sigma_{SH}} \tag{2.5}$$

Σ^*_{SH} is the set of all Hypertext artifacts H (i. e., flat collections of symbols) that are subset of or equal to the spatial hypertext "alphabet" Σ_{SH}. Thus, Σ^*_{SH} is the set of all subsets of Σ_{SH} (incl. $H_\varepsilon, \Sigma_{SH} \in \Sigma^*_{SH}$) and is therefore the powerset of Σ_{SH}.

Given these definitions we can finally rewrite $L_{SH} = \left(\Sigma^*_{SH} \setminus X_{SH}\right)$ to:

$$L_{SH} = \left(2^{\Sigma_{SH}} \setminus X_{SH}\right) = \left(2^{\left(A_0 \times A_1 \times \ldots \times A_{n-1}\right)} \setminus X_{SH}\right) \tag{2.6}$$

Even though this is an apparently straightforward approach to formalization, it still allows us to talk about spatial hypertexts in a uniform way (something which had not been possible before). This will proof very helpful in what we are going to do in the following chapters.

In summary we can formally define a spatial hypertext language as follows:

$$\begin{aligned}
L_{SH} &= \Sigma^*_{SH} \setminus X_{SH} \\
&\quad \Sigma^*_{SH} = \{H | H \subseteq \Sigma_{SH}\} = 2^{\Sigma_{SH}} \\
&\quad\quad \Sigma_{SH} = (A_0 \times A_1 \times \ldots \times A_{n-1}),\ n \geq 1 \\
&\quad\quad\quad A_i \neq \emptyset\ (0 \leq i \leq n-1) \\
&\quad X_{SH} \subset \Sigma^*_{SH},\ H_\varepsilon \notin X_{SH},\ H_\varepsilon = \emptyset \\
\Rightarrow L_{SH} &= \left(2^{\Sigma_{SH}} \setminus X_{SH}\right) = \left(2^{(A_0 \times A_1 \times \ldots \times A_{n-1})} \setminus X_{SH}\right) \qquad (2.7)
\end{aligned}$$

Here it becomes obvious that spatial hypertext languages are mainly determined by two components: Σ_{SH} and X_{SH}. This allows for a parameterized definition of such languages in a "template-like" notation (which will play an important role later in this thesis):

$$L_{SH}\langle \Sigma_{SH}, X_{SH} \rangle := \left(2^{\Sigma_{SH}} \setminus X_{SH}\right) \qquad (2.8)$$

So, instead of $\left(2^{\Sigma_{SH}} \setminus X_{SH}\right)$ one could also use the synonymous expression $L_{SH}\langle \Sigma_{SH}, X_{SH} \rangle$. This does not change the original meaning of Eq. 2.7 and is merely used for simplification.

The following example shall help us to better understand Eq. 2.7:

Let us assume, that we are looking for the smallest spatial hypertext language possible. Following Eq. 2.1, every spatial hypertext language L_{SH} can be defined as $L_{SH} = \left(\Sigma^*_{SH} \setminus X_{SH}\right)$, where $X_{SH} \subset \Sigma^*_{SH}$ (Eq. 2.2). Thus, the size of L_{SH} is determined by the size of both Σ^*_{SH} and the exclusion set X_{SH}. The larger Σ^*_{SH} and the smaller X_{SH} the greater the size of L_{SH}, and, vice versa, the smaller Σ^*_{SH} and the larger X_{SH} the smaller the size of L_{SH} will be. Consequently, if one intends to reduce the size of L_{SH} to a *minimum*, then Σ^*_{SH} must be *minimized* and X_{SH} must be *maximized*. This is exactly what we are doing now:

Firstly, we need an adequate definition of attribute types. According to Eq. 2.4, both is required (a) there has to be at least *one* attribute type defined and (b) attribute types may never be empty. Both requirements can be met, if we have only a *single* attribute type, including only a *single* attribute value. The semantics of the type, however, do not matter in this example (typically it would be *position*; since we are primarily interested in spatial expressions).

Here, such a *minimal* attribute type shall be defined as:

$$A_{min} = \{a\}$$

When we apply now the rule implied by Eq. 2.4 on A_{min} we get:

$$\Sigma_{SH_{min}} = \big\{(a)\big\} \tag{2.9}$$

So, the *minimal* set of spatial hypertext symbols (which we denote as $\Sigma_{SH_{min}}$) includes only a single element with only one attribute. When we take this definition of a *minimal* spatial hypertext "alphabet" and apply the rule implied by Eq. 2.5 on it, then we get the following *minimal* set of spatial hypertext artifacts:

$$\Sigma^*_{SH_{min}} = 2^{\Sigma_{SH_{min}}} = 2^{\big\{(a)\big\}} = \Big\{\big\{(a)\big\}, \emptyset\Big\} = \Big\{\big\{(a)\big\}, H_\varepsilon\Big\} \tag{2.10}$$

This *minimal* set includes just two elements: an artifact with only a single symbol – the unary tuple (a) – and the empty spatial hypertext artifact $H_\varepsilon = \emptyset$ (which is by default included in the power set).

So, we know already both *minimal* sets, $\Sigma_{SH_{min}}$ (Eq. 2.9) and $\Sigma^*_{SH_{min}}$ (Eq. 2.10). What still needs to be determined is an adequate, *maximized* exclusion set (which we will denote as X_{SHmax}).

According to Eq. 2.3, an exclusion set X_{SH} must be a proper subset of Σ^*_{SH} and may not inlude H_ε. In our case this means, that $X_{SH_{max}} \subseteq (\Sigma^*_{SH_{min}} \setminus \{H_\varepsilon\})$, which can be rewritten as follows:

$$\begin{aligned} X_{SH_{max}} \subseteq \Big(\Sigma^*_{SH_{min}} \setminus \{H_\varepsilon\}\Big) &\Leftrightarrow X_{SH_{max}} \subseteq \left(\Big\{\big\{(a)\big\}, H_\varepsilon\Big\} \setminus \{H_\varepsilon\}\right) \\ &\Leftrightarrow X_{SH_{max}} \subseteq \Big\{\big\{(a)\big\}\Big\} \end{aligned}$$

The only two assignments of the variable $X_{SH_{max}}$ which could fulfill the constraint $X_{SH_{max}} \subseteq \{\{(a)\}\}$, are $X_{SH_{max}} = \emptyset$ and $X_{SH_{max}} = \{\{(a)\}\}$. Since we are searching for an exclusion set, that should be *maximal* in size, we can reject $X_{SH_{max}} = \emptyset$. We therefore set:

$$X_{SH_{max}} = \Big\{\big\{(a)\big\}\Big\} \tag{2.11}$$

Knowing now both required components $\Sigma^*_{SH_{min}}$ (Eq. 2.10) and the maximal exclusion set $X_{SH_{max}}$ we can finally determine the minimal language $L_{SH_{min}}$:

$$\begin{aligned} L_{SH_{min}} &= \Sigma^*_{SH_{min}} \setminus X_{SH_{max}} \\ &= \Big\{\big\{(a)\big\}, H_\varepsilon\Big\} \setminus \Big\{\big\{(a)\big\}\Big\} \\ &= \{H_\varepsilon\} \end{aligned}$$

$$L_{SH_{min}} \Leftrightarrow L_{SH}\left\langle \Sigma_{SH_{min}}, X_{SH_{max}} \right\rangle \tag{2.12}$$

From this (Eq. 2.12) it can be concluded, that the smallest spatial hypertext language possible is a language that comprizes only of a single element, which is the empty spatial hypertext artifact (H_ε). Thus, the least we can express with a spatial hypertext language is an empty information space (i. e., white screen). Spatial hypertexts do not fall simply from sky, but are always the result of a development process that has to start from somewhere (typically from an empty workspace). In this regard our definition makes perfect sense.

So far we have discussed which artifacts (or words) belong to a spatial hypertext language. In addition to this "syntactic view", if you want to call it that, there is still another aspect, that is even more important: *semantics*

As we have already seen (Sect. 1.4), there is only little concrete structure in human-generated layouts. Although in spatial hypertext research one has tried to identify several categories of spatial and visual structures (such as lists, stacks, piles, heaps etc.) they still should be treated with caution. There is *no* guarantee that these spatial patterns are always valid or that their recognition brings any benefit. The main reason for this is, that implicit structures typically emerge individually in mind and depend on the context in which they are used. This makes premature definition of default-structures extremely difficult (or even impossible). In other words, there is not too much we can say about spatial hypertexts *before* users start working on them.

To put it in one sentence:

Spatial hypertext structures result from creative work
and therefore cannot be predicted!

Not only did this influence our definition of spatial hypertext languages (as given in Eq. 2.7), but has also affected our understanding of "spatial parsing":

Rather than checking the canvas against predefined structural patterns (which is common practice today), spatial parsers should *imitate* humans in the way they perceive structure. In other words, instead of searching hypertexts for supposedly universal structures (such as stacks, tables, heaps etc.), spatial parsers rather should perceive structure through the lense of a human observer (i. e., spatial parsers should "interpret" spatial hypertexts).

As system designers we cannot with any certainty predict which structures are typically created by users. That is not possible. In this respect spatial hypertext offers just too much creative leeway. The one thing we can be sure of, however, are the basic principles by which human observers (and thus hypertext authors) perceive spatial structure. Good examples include the so-called Gestalt-principles or rather Gestalt-heuristics (such as the laws of similarity, proximity, or symmetry etc.). By an intelligent combination of such heuristics, spatial parsers could provide structural interpretations which might come quite close to what author(s) had in mind when they were creating a hypertext. This way one could detect *intended* structure for a large number of people in many different contexts of application, even though it is unknown *who* created the hypertext and for *what* purposes. Our parsing algorithms (that will be discussed later on) build on this idea. They take spatial hypertext artifacts as parser *input*, analyse them with basic, common sense heuristics and finally generate *output* in the form of so-called *interpretations*, which are discussed in the following section.

2.2 Spatial Hypertext Interpretations

As already pointed out in Sect. 1.2, spatial hypertext is of *ambiguous* nature. This allows for reading spatial hypertext artifacts in different ways. Both, humans and parsers may have different views on spatial hypertext and its implicitly encoded information. Depending on the angle from which one tries to understand the spatial and visual dependencies displayed on screen (or provided via other interfaces), there will typically be more than only one way to read the hypertext. Depending on your previous knowledge in mind and where you start with your analysis, different aspects are treated with different priority, so that some structures will be preferred over others. This results in multiple (alternative) *interpretations*.

Interpretations are encodings of how spatial hypertext can be understood (by both humans or parsers) and thus form the *semantic complement* to spatial hypertext artifacts (Sect. 2.1). In concrete terms, this means that they are formal descriptions of how visual symbols and (even more important) symbol relations can be interpreted. Typically, symbols are treated as visual placeholders for information objects, and object relations are implicitly given by symbol properties. Insofar interpretations only vary in object relations but not in the number of objects.

We express such interpretations as collections of information structures, or rather as networks of information objects (remember that spatial hypertext is actually an alternative representation form of node-link hypertext; Sect. 1.1). However, to distinguish clearly between the node-link model of classic hypertext and our formal interpretation model, we will use the terms “information unit” and “association” instead of “node” and “link”:

Let X_I be a set of k (alternative) interpretations I_i (where $0 \leq i \leq k-1$):

$$X_I = \{I_0, I_1, \ldots, I_{k-1}\}\,;\ |X_I| = k \tag{2.13}$$

$$(U, A) = \left(\begin{matrix} \{u_0, u_1, u_2\}, \\ \left\{ \begin{matrix} \{u_0, u_1\}, \\ \{u_0, u_2\}, \\ \{u_1, u_2\} \end{matrix} \right\} \end{matrix} \right) \Rightarrow$$

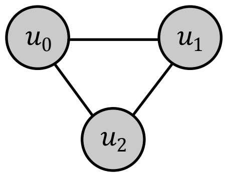


Figure 2.1: Example of a complete, undirected interpretation graph (U, A), including three information units u_0, u_1 and u_2, which are connected by three associations $\{u_0, u_1\}$, $\{u_0, u_2\}$, and $\{u_1, u_2\}$.

Each interpretation I_i $(0 \leq i \leq k-1)$ included in X_I (Eq. 2.13), shall be defined as a triple (U, A, w_i), where U is a set of n information <u>U</u>nits, A is a set of <u>A</u>ssociations, and w_i represents a <u>w</u>eighting function that is specific to interpretation I_i:

$$I_i := (U, A, w_i) \tag{2.14}$$

Provided that Ω is the universal set of *all* theoretically possible information units (i.e., the basic set of all carriers of information), we can define U (which is *some* collection of information units) as an arbitrary subset of Ω and herewith as an element of 2^Ω:

$$U = \{u_0, u_1, \ldots, u_{n-1}\} \in 2^\Omega \; ; \; |U| = n \tag{2.15}$$

Since in theory *all* elements of U could be associated with each other and as we are dealing here only with *undirected* associations, we can define A as the set of (binary) subsets $\{u, u'\}$ of U, which can be written as $\binom{U}{2}$:

$$A = \binom{U}{2} = \Big\{ \{u_h, u_l\} \;\Big|\; 0 \leq h < l \leq n-1 \Big\} \tag{2.16}$$

The number of associations $|A|$ is given here by a triangular number:

$$|A| = \left| \binom{U}{2} \right| = \binom{|U|}{2} = \binom{n}{2} = \frac{n!}{2!(n-2)!} = \frac{n(n-1)}{2} = \frac{n^2 - n}{2} \tag{2.17}$$

Together U and A apparently form a complete undirected graph, where vertices (or nodes) represent *information units* and edges are regarded as *associations.* Fig. 2.1 illustrates a simple example with three information units u_0, u_1, u_2.

With this tool at hand we can express that in theory all carriers of information $u \in U$ that are encoded in a spatial hypertext artifact could be associated with each other in *some* way. Modeling full interpretations, however, requires more than only describing plain connections between objects. Only specifying that all units might be related to each other is not sufficient. In addition we must be able to parameterize associations between information units. This means, it must be possible to define properties of object relations (if there are any). This is why the tuple (U, A) gets extended by the binary relation w_i.

w_i shall be a weighting function assigning weights (i.e., parameters) to edges (i.e., associations). More precisely w_i is a total function mapping A to a non-empty set of association $\underline{\text{P}}$arameterizations, which we simply call P:

$$\begin{aligned} & w_i : A \rightarrow P, \\ & \quad P = (P_0 \times P_1 \times \ldots \times P_{m-1}) \cup P_\varepsilon \ , \ m \geq 1 \\ & \qquad P_j \neq \emptyset \, (0 \leq j \leq m-1) \\ & \qquad P_\varepsilon \neq \emptyset, \ P_\varepsilon \subset P \end{aligned} \tag{2.18}$$

A key role here is played by P_ε. P_ε is a (non-empty) set of so-called *empty parameterizations*. When assigned to an edge $\{u, u'\} \in A$, such an empty parameterization $p \in P_\varepsilon$ may indicate either that there is *no* specific relationship between u and u', or that such a relation is *unknown* (there might be one, but we are note sure about it). Given that interpretations are collections of structures and structures are networks of meaningfully connected information units, we can use elements of P_ε as structure delimiters. In concrete terms, elements of P_ε can be used to separate discrete structures from each other.

Mathematically expressed, using P_ε we can define the jth discrete information structure included in an ith interpretation $I_i := (U, A, w_i)$ as a connected sub-graph I'_{ij} which has the following properties:

$$\begin{aligned} & I'_{ij} := (U_{ij}, A_{ij}, w_{ij}) \\ & \qquad U_{ij} \subseteq U; \ \left|U_{ij}\right| \geq 2 \\ & \qquad A_{ij} = \left\{ a \,\middle|\, a \in \binom{U_{ij}}{2} \wedge w_i(a) \notin P_\varepsilon \right\} \Rightarrow A_{ij} \subseteq A; \ \left|A_{ij}\right| \geq 1 \\ & \qquad w_{ij} : A_{ij} \rightarrow P \setminus P_\varepsilon, \ \forall a \in A_{ij} : w_{ij}(a) = w_i(a) \end{aligned} \tag{2.19}$$

According to this (Eq. 2.19), we can define information structures as connected sub-graphs I'_{ij} of interpretation graphs I_i. Such sub-graphs comprize of at least two vertices (i. e., information units $u \in U_{ij}$) which are connected by undirected edges (i. e., associations $a \in A_{ij}$) that are decorated only with values $\notin P_\varepsilon$.

We have chosen $\left|U_{ij}\right| \geq 2$ and so the constraint $\left|A_{ij}\right| \geq 1$ in Eq. 2.19 because we firmly believe that only *relations* create structure. Single information units are nothing more than *atomic* objects; their internal composition (i. e., “payload”) does not matter here. Spatial parsers search for visual object relations but do not “care” about content.

The following example is intended to make that a bit clearer:

Assume we have selected five random information units u from Ω. For the sake of simplicity we number them continuously from u_0 to u_4. The exact values of these variables shall not play a role here. Then we can, according to Eq. 2.15 and Eq. 2.16 define both components U and A as follows:

$$\begin{aligned} U &= \{u_0, u_1, u_2, u_3, u_4\} \\ A &= \left\{ \begin{matrix} \{u_0, u_1\}, \{u_0, u_2\}, \{u_0, u_3\}, \{u_0, u_4\}, \\ \{u_1, u_2\}, \{u_1, u_3\}, \{u_1, u_4\}, \\ \{u_2, u_3\}, \{u_2, u_4\}, \\ \{u_3, u_4\} \end{matrix} \right\} \end{aligned} \tag{2.20}$$

In addition, the following parameter sets shall be given:

$$\begin{aligned} P_{type} &= \{\text{type}_0, \text{type}_1, \text{type}_2\} \\ P_{weight} &= \{0.25, 0.50, 0.75, 1.00\} \\ P_\varepsilon &= \{\varepsilon\} \end{aligned}$$

Here, P_{type} is a set of discrete type indicators (ranging from type_0 to type_2) and P_{weight} is a discrete set of real-valued weights, which we define (for reasons of simplicity) in steps of $\frac{1}{4}$ from 0.25 to 1.00. The mandatory set of empty parameterizations P_ε shall comprise here only of a single element, which we denote as ε. With these parameter sets and the definitions given by Eq. 2.18, we can now define the parameterization set P as follows:

$$\begin{aligned} P &= (P_{type} \times P_{weight}) \cup P_\varepsilon \\ &= \left\{ \begin{matrix} (\text{type}_0, 0.25), (\text{type}_0, 0.50), (\text{type}_0, 0.75), (\text{type}_0, 1.00), \\ (\text{type}_1, 0.25), (\text{type}_1, 0.50), (\text{type}_1, 0.75), (\text{type}_1, 1.00), \\ (\text{type}_2, 0.25), (\text{type}_2, 0.50), (\text{type}_2, 0.75), (\text{type}_2, 1.00), \\ \varepsilon \end{matrix} \right\} \end{aligned} \tag{2.21}$$

Based on these definitions of U, A and P we will now describe three example interpretations: $X_I = \{I_0, I_1, I_2\}$.

According to Eq. 2.14, each of these interpretations I_i is a triple (U, A, w_i), comprising of the two shared components U and A, and the interpretation specific weighting function w_i. As per Eq. 2.18 w_i is a total function and thus a special binary relation. We therefore illustrate w_0, w_1 and w_2 in tabular form: see Fig. 2.2.

Each table in Fig. 2.2 includes $|A| = 10$ ordered pairs consisting of associations $\{u, u'\} \in A$ (Eq. 2.20) and assigned parameterizations $p \in P$ (Eq. 2.21).

$I_0 = (U, A, \boldsymbol{w}_0)$

$\boldsymbol{w}_0$	
$\{u_0, u_1\}$	ε
$\{u_0, u_2\}$	ε
$\{u_0, u_3\}$	ε
$\{u_0, u_4\}$	ε
$\{u_1, u_2\}$	$(type_1, 0.75)$
$\{u_1, u_3\}$	$(type_0, 0.50)$
$\{u_1, u_4\}$	$(type_1, 0.50)$
$\{u_2, u_3\}$	ε
$\{u_2, u_4\}$	$(type_2, 1.00)$
$\{u_3, u_4\}$	ε

$I_1 = (U, A, \boldsymbol{w}_1)$

$\boldsymbol{w}_1$	
$\{u_0, u_1\}$	ε
$\{u_0, u_2\}$	ε
$\{u_0, u_3\}$	$(type_0, 0.50)$
$\{u_0, u_4\}$	ε
$\{u_1, u_2\}$	$(type_0, 0.50)$
$\{u_1, u_3\}$	ε
$\{u_1, u_4\}$	$(type_1, 1.00)$
$\{u_2, u_3\}$	ε
$\{u_2, u_4\}$	$(type_1, 0.75)$
$\{u_3, u_4\}$	ε

$I_2 = (U, A, \boldsymbol{w}_2)$

$\boldsymbol{w}_2$	
$\{u_0, u_1\}$	ε
$\{u_0, u_2\}$	$(type_2, 0.50)$
$\{u_0, u_3\}$	ε
$\{u_0, u_4\}$	$(type_1, 1.00)$
$\{u_1, u_2\}$	ε
$\{u_1, u_3\}$	$(type_0, 0.50)$
$\{u_1, u_4\}$	ε
$\{u_2, u_3\}$	ε
$\{u_2, u_4\}$	$(type_1, 0.75)$
$\{u_3, u_4\}$	ε

Figure 2.2: Three example weighting functions w_0, w_1 and w_2 illustrated in tabular form.

Values of p were chosen arbitrarily. The definitions given by Eq. 2.19 allow for identifying a number of discrete information structures in the form of connected sub-graphs. Value pairs belonging to such structures are marked with different colors (red and blue). When we combine these colored tuples in the form described in Eq. 2.19 we get the following information structures:

$$I'_{00} = \left(\begin{array}{c} \{u_1, u_2, u_3, u_4\}, \\ \left\{ \{u_1, u_2\}, \{u_1, u_3\}, \{u_1, u_4\}, \{u_2, u_4\} \right\}, \\ \left\{ \begin{array}{c} (\{u_1, u_2\}, (\text{type}_1, 0.75)), \\ (\{u_1, u_3\}, (\text{type}_0, 0.50)), \\ (\{u_1, u_4\}, (\text{type}_1, 0.50)), \\ (\{u_2, u_4\}, (\text{type}_2, 1.00)) \end{array} \right\} \end{array} \right)$$

$$I'_{10} = \left(\begin{array}{c} \{u_0, u_3\}, \\ \left\{ \{u_0, u_3\} \right\}, \\ \left\{ (\{u_0, u_3\}, (\text{type}_0, 0.50)) \right\} \end{array} \right) \quad I'_{11} = \left(\begin{array}{c} \{u_1, u_2, u_4\}, \\ \left\{ \begin{array}{c} \{u_1, u_2\}, \{u_1, u_4\}, \\ \{u_2, u_4\} \end{array} \right\}, \\ \left\{ \begin{array}{c} (\{u_1, u_2\}, (\text{type}_0, 0.50)), \\ (\{u_1, u_4\}, (\text{type}_1, 1.00)), \\ (\{u_2, u_4\}, (\text{type}_1, 0.75)) \end{array} \right\} \end{array} \right)$$

$$I'_{20} = \left(\begin{array}{c} \{u_1, u_3\}, \\ \left\{ \{u_1, u_3\} \right\}, \\ \left\{ (\{u_1, u_3\}, (\text{type}_0, 0.50)) \right\} \end{array} \right) \quad I'_{21} = \left(\begin{array}{c} \{u_0, u_2, u_4\}, \\ \left\{ \begin{array}{c} \{u_0, u_2\}, \{u_0, u_4\}, \\ \{u_2, u_4\} \end{array} \right\}, \\ \left\{ \begin{array}{c} (\{u_0, u_2\}, (\text{type}_2, 0.50)), \\ (\{u_0, u_4\}, (\text{type}_1, 1.00)), \\ (\{u_2, u_4\}, (\text{type}_1, 0.75)) \end{array} \right\} \end{array} \right)$$

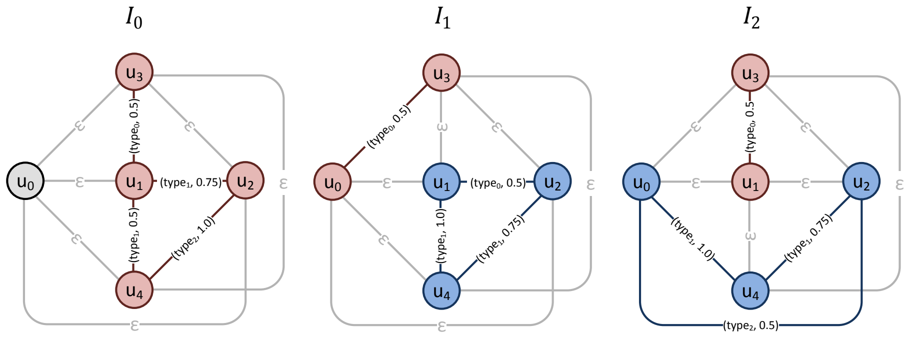


Figure 2.3: Three example interpretations I_0, I_1 and I_2 illustrated in the form of two-dimensional graph diagrams. Included structures are highlighted in red and blue.

Since these tuples are graphs, one can also illustrate them in form of diagrams. Fig. 2.3 illustrates I_0, I_1 and I_2 in two-dimensional graph representation (aligned from left to right). Included information structures are highlighted with respective colors, red and blue (which is consistent with Fig. 2.2).

Summarising the above it can be said, that interpretations of spatial hypertext artifacts are collections of clearly separable networks of meaningfully connected information units, which we denote as information structures. Together with our notion of spatial hypertext languages (from Sect. 2.1) this understanding of interpretations and information structures will play an important role in following chapters.

Chapter 3

Critical Review

The last two chapters provided different views on spatial hypermedia. It was shown how spatial hypertext and spatial parsing are informally defined in literature (Sect. 1.2 and Sect. 1.3) and which practical implementations of spatial parsers exist (Sect. 1.5). We also had a formal view on spatial hypertext. Chapter 2 provided a common mathematical, and hence terminological basis, which allows us now to talk about spatial hypertexts in a uniform way. According to Sect. 2.1, spatial hypertext languages are defined as sets of spatial hypertext artifacts, which are flat collections of spatial hypertext symbols. This forms our "syntactic view" on such languages. Semantics are defined by interpretations (Sect. 2.2). Interpretations are encodings of how spatial hypertext can be understood and thus form the semantic complement to spatial hypertext artifacts. Technically they are graphs of connected information objects.

Spatial parsers are the linking element between spatial hypertext artifacts and interpretations. They take artifacts as input, analyse them according to predefined heuristics and generate output in the form of interpretations. In a nutshell, spatial parsers *map* artifacts to interpretations (Fig. 3.1).

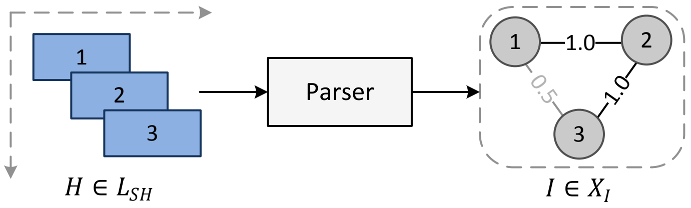


Figure 3.1: spatial parsers map spatial hypertext artifacts H to interpretations I

One of the major goals in spatial hypermedia research is to design and develop good spatial parsing services. That is, finding feasible ways for automatically retrieving hidden structure from human generated layouts. Such implicit structure detection belongs to the most essential features of a spatial hypermedia system. One could even go further arguing that only spatial parsers transform visual information spaces into real spatial hypermedia systems. Without structure recognition they would be nothing more than plain graphical editors with a database in the backend. Or to put it differently, only spatial parsing makes an image on the screen a spatial hypertext. But this is almost a philosophical issue which can be discussed controversially. We shall not go into any more detail on this.

Regardless of the role spatial parsers play in setting spatial hypermedia systems apart from normal diagramming applications, we would like to ask the following question:

What makes a *good* spatial parser?

The answer to this question is strongly linked to the meaning of “good” and hence on what a potential system user is likely to expect from a parsing service. “good” is a vague rating and can relate to many different assessment criteria. That is why in the following we focus on two major aspects only:

(a) efficiency and (b) effectiveness.

Usually spatial parsers are realized as software components. Therefore, aspects such as processing speed and resource consumption have, without any doubt, a great relevance to developing *good* parsers. In concrete terms, when designing and implementing a fully-functional spatial hypermedia system it must be ensured that resource allocation by parsers is kept to a minimum and user interaction with the visual medium is not interrupted or interfered by unnecessarily long parser runs. To put it briefly, a good spatial parser should operate *efficiently.* In this respect, spatial parsers do not differ from other software components. But, efficiency alone does not automatically make a good parser in terms of good structure detection performance.

More important than efficiency is the quality of parser output (i. e. the accuracy of interpretations). The closer a spatial parser's interpretation of a spatial hypertext comes to the real understanding of human users, the greater the parser's accuracy. The more accurate a parser's interpretations are the better the parsing algorithm and hence the stronger the parser's performance. Ideally, spatial parsers extract *only* structure that was really intended by authors. In other words, a *perfect* spatial parser detects *exactly* what a human user intended to express by a certain arrangement of information objects.

Therefore, *good* spatial parsers do not only perform as resource-saving and as fast as possible, but also generate interpretations of high accuracy and hence results of high quality.

Unfortunately both objectives, maximized efficiency and maximal effectiveness are inherently difficult to reconcile. Especially in computer science they are often seen as competing aims. This is why we focus on a single aspect only, namely parsing *accuracy*. This leads us to the following question:

Is there a need for increased parsing accuracy?

To avoid misunderstandings, it shall be noted that in this thesis we consider only *un*–customized parsers and hence systems that do *not* need to be tailored for user preferences. Thus, *Adaptive Spatial Parsing* (Sect. 1.5.2) is not our issue here.

Does the state-of-the-art in (non-adaptive) spatial parsing call for improved algorithms that provide more accurate and hence better results? In our opinion this questions can be answered with "Yes", for the following reason:

Traditional (non-adaptive) parsing strategies (as described in Sect. 1.5.1) build on the assumption that a single spatial hypertext artifact is a sufficient source of information for retrieving implicitly encoded structure that was intended by authors. Or in other words, it is assumed that *static* "snapshots" of visual information spaces include sufficient information for inferring correct structural meaning. But this need not always be the case.

Quite the contrary: single spatial hypertext artifacts rarely allow for clear inferences on what authors really intended to express. The reason for this is that drawing unambiguous conclusions only from spatial and visual properties would require that structural descriptions given in a spatial hypertext artifact are also clear and explicit. Although being possible, this still contradicts spatial hypertext's ambiguous and implicit nature (Sect. 1.2).

Typically, spatial hypertext rather leaves some room for interpretation. That is, spatial hypertext allows the viewer (either human or machine) leeway to interpret spatial and visual properties (Sect. 2.2). The only one who could resolve such ambiguities correctly is the creater of the respective visual language and hence the author of the hypertext (Sect. 2.1). In short, spatial hypertext is ambiguous and disambiguation requires knowledge which only authors can have. Conventional (non-adaptive) parsers, however, do not consider such knowledge in their analysis, which innately limits parsing accuracy. This conceptual restriction makes it difficult to design *good* parsing algorithms.

Thus, there is with no doubt a need for improvement!

3.1 Ambiguous Structures

As already highlighted in Sect. 1.2, lack of clarity is inherently part of spatial hypertext. As a consequence, disambiguation plays an essential role in spatial parsing. Nevertheless, apart from only few exceptions, such as FLAPS (Sect. 1.5.1), the focus in designing spatial parsers has been put on aspects such as collaboration (Sect. 1.5.3), three-dimensionality (Sect. 1.5.4), or linear document authoring (Sect. 1.5.5) rather than on resolving ambiguities. In other words, although being crucial for successful spatial parsing, the topic of disambiguation has been rather neglected in most current implementations. For this reason it is worth delving a bit deeper into that subject:

What does "ambiguous" mean in connection with spatial hypertext?

An "ambiguous" spatial hypertext has different possible meanings. The meaning of a spatial hypertext as a whole is defined by the meaning of individual information units (nodes), which depends on both node content and context [1]. Context again is defined by (implicit) associations of information units with adjacent nodes within the hypertext [1], which is determined by spatial and visual properties respectively. As discussed in Sect. 2.2, information units and associations together form information structures and structures build interpretations. So, one could argue that interpretations are manifestations of the meaning of spatial hypertext. Thus, by calling a spatial hypertext "ambiguous" we are emphasizing that one could infer more than only one interpretation.

Intended (or constructive [3,9]) ambiguity is given, when users *deliberately* express unclear or fuzzy information structures and herewith allow for alternative interpretation. This means, authors are aware that there are multiple ways how a given hypertext could be understood. This constructive use of (intended) ambiguities to express vague structures is a core feature of spatial hypertext [3,9]. We discussed that already in Sect. 1.2.

In addition to such intended ambiguities, however, evolution of spatial hypertexts might also result in ambiguities that are unintended by authors [35]. Spatial hypertext's implicit, informal and emergent nature (Sect. 1.2) may lead to inconsistencies between intention and actual expression. So, it is for example possible, that edit operations applied to spatial objects may *accidentally* modify the context of related expressions, which then may become undesirably ambiguous [13, 36]. Another example refers to the visual language used. It is not only the case that the spatial hypertext changes over time [9], the users' understanding of their task and herewith the spatial hypertext language may evolve as well [9, 20, 36, 37]. As a consequence, expressions that could be clearly understood before a change of language, may become ambiguous afterwards since the meaning of those visual features used has changed [37]. This

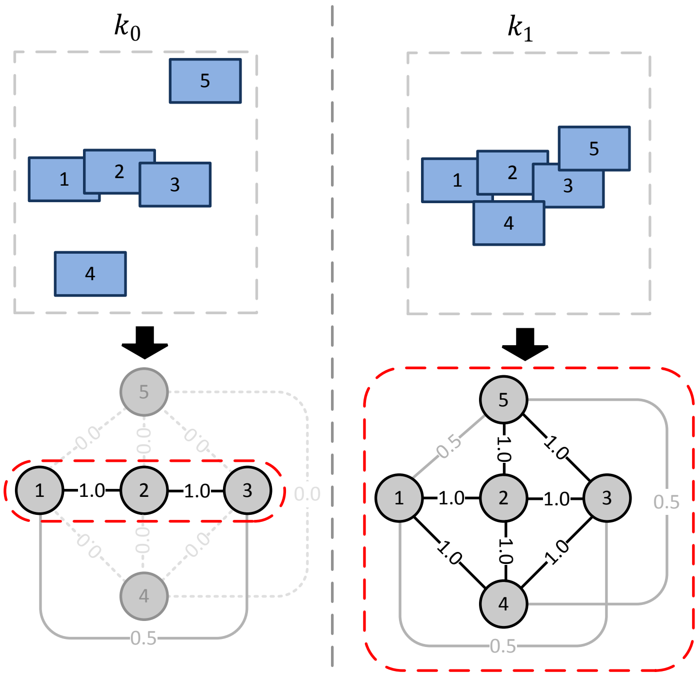


Figure 3.2: example: list-structures formed by ambiguously aligned rectangles

complicates interpretation of information or makes it practically impossible at all [36, 37].

This is best explained with an example. Fig. 3.2 illustrates an ambiguous sample hypertext and its interpretation by a conventional spatial parser at two consecutive points in time (k_0 and k_1).

Interpretations are represented here as graphs which are complete, undirected and weighted. Given weights assigned to edges (i. e., to associations) between information units range from 0.0 to 1.0 and indicate the strength of a certain relationship (in this case the strength of spatial dependencies). Thus, a weight of 0.0 would mean that two objects have no spatial relation, whereas a weight of 1.0 indicates an immediate dependency. For further details on interpretations see our previous definitions in Sect. 2.2.

In state k_0 (top-left of Fig. 3.2) objects 1, 2 and 3 in the middle of the information space apparently form a horizontal list; thus they can be interpreted as structure elements. Rectangles 4 and 5 in the bottom-left and top-right corner, however, are located too far away from potential neighbors to have a spatial relation; thus they do not contribute to any structure. A spatial parser can easily detect that, as can be seen in the bottom-left of Fig. 3.2.

When an author pushes now rectangle 4 and 5 closer together, in order to save space for example, then we get what is illustrated at k_1. Suddenly the layout becomes ambiguous. Only by analysing the spatial layout it is not possible anymore to decide which rectangles form a discrete list and which objects do not. Consequently, the spatial parser's interpretation becomes ambiguous either. This is why the interpretation graph in the bottom right of Fig. 3.2 comprizes a single, big information structure which includes *all five* information units. The list-structure formed by objects 1, 2 and 3, which could be recognized at state k_0, is still part of this big structure, but, since all the delimiting zero-weightings have gone, it cannot be unambiguously identified anymore.

The author of this sample hypertext, however, still might recognize that horizontal list as an independent structure. Perhaps the snapshot on the right was simply intended as a spatially compact (i. e., "compressed") version of the hypertext on the left and hence the author does not see any semantic difference between k_0 and k_1. Possibly in the user's mind the list formed by objects 1, 2 and 3 still exists as a discrete unit. In the spatial parser's interpretation, however, it does not.

From this we can conclude that implicit information structures in spatial hypertext are always a mixture of desired and undesirable structures. Thus, spatial hypertext includes both, intended and unintended ambiguities.

We denote the degree of such (intended and unintended) ambiguousness as *level-of-ambiguity*. It can be measured as the number of possible alternative interpretations that can be derived from a spatial hypertext. It herewith determines how hard it is to come to a correct interpretation. The smaller that level is the easier it gets to infer the correct meaning and hence the better a spatial parser will perform. Consequently, if one intends to improve parsing performance that level must be *decreased*. This brings us back to the subject of "disambiguation".

A possible way to achieve that is suggested in [36]. According to [36], (unwanted) ambiguities can be eliminated by considering a spatial hypertext's edit history (called "navigable history"). In concrete terms, if we do not only look at a static image of our information space, but rather take into account its evolution as an additional source of information, disambiguation of visual structures may become possible. This is why in [36] it was suggested to use such a history to enhance spatial parsing. A possible implementaton, however, was not described.

Thus, we can see an edit history as a means to reduce the level of (unintended) ambiguity in spatial hypertext and thus as a means to increase parsing accuracy.

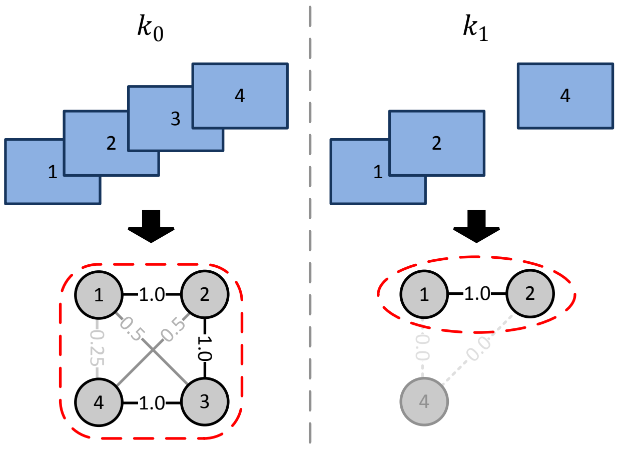


Figure 3.3: example: (partly) destroyed list-structure of diagonally aligned rectangles

3.2 Destroyed Structures

Besides *ambiguous structures* (Sect. 3.1) we can identify another structure category. So-called *destroyed structures* are information structures described spatially or visually, which used to exist in the past, but were *partly* or *completely* destroyed in the course of an editing process. Thus, all information structures which could be automatically detected at a given time in the past but are not (fully) visible anymore at present, due to changes made to spatial and visual attributes, fall in this category.

A very simple example which illustrates the nature of such *destroyed structures* is given in Fig. 3.3. Fig. 3.3 illustrates a diagonally aligned list of rectangular shaped information objects and its interpretation by a spatial parser at two different points in discrete time (k_0 and k_1). At k_0 the interpretation of the given spatial hypertext comprizes a single information structure including all four information units. At k_1 however unit number 3 is missing which leads to a drop of the spatial dependency of units 1,2 with rectangle 4 to zero. Deleted object number 3 is not included at all. The recognized information structure includes only units 1 and 2. However, what might be geometrically correct does not necessarily need to be valid from a human user's perspective. Probably object number 3 was *accidentally* destroyed or the user simply forgot to move 2 and 4 closer together. Maybe from a user's point of view units 1, 2 and 4 still form a list. A conventional spatial parser, however, recognizes only a small fraction of that intended structure. A similar example that illustrates this issue can be found in [36].

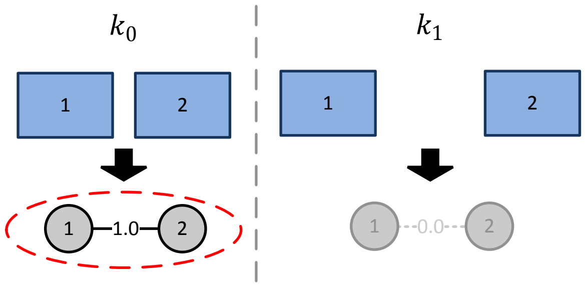


Figure 3.4: example: (completely) destroyed object relation

This becomes even more apparent in cases where information structures are *completely* destroyed. You only have to look at Fig. 3.4. As in the previous example, also Fig. 3.4 illustrates two consecutive snapshots of a sample hypertext together with its interpretation by a spatial parser. Due to their spatial proximity at time point k_0, both objects 1 and 2 are considered as being strongly related. That is, the parser "sees" a spatial relation with a strength of one-hundred percent. A single time step later, however, both objects 1 and 2 have moved too far away from each other to still have a spatial relation. Consequently, the parser does not just detect a weak connection, but he recognizes none at all. This does not necessarily need to be correct. Maybe the change from k_0 to k_1 happened accidentally (possibly as a side effect of another editing operation), or the parser's configuration simply does not fit the author's understanding of proximity. Note, that we have absolutely no guarantee that heuristics used for spatial parsing are universally correct. So there are enough reasons why a conventional spatial parser erroneously detects nothing in k_1.

3.3 Temporal Structures

Both *ambiguous* (Sect. 3.1) and *destroyed structures* (Sect. 3.2) can not be detected properly by conventional spatial parsers, even though they are expressed spatially and visually. It is not surprising that structures formed by totally different attributes than position, size, shape, color etc. can not be recognized either. This category includes (pure) *temporal structures*.

The very simple example illustrated in Fig. 3.5 is characteristic of this type of structures. Fig. 3.5 shows two rectangular information units which are located too far away from each other to be regarded as spatially associated (i. e., the strength of spatial dependency would be 0.0). Thus a spatial parser would not see any information structure. However there still might be an association between unit 1 and 2, as, when you look back in edit history it could be

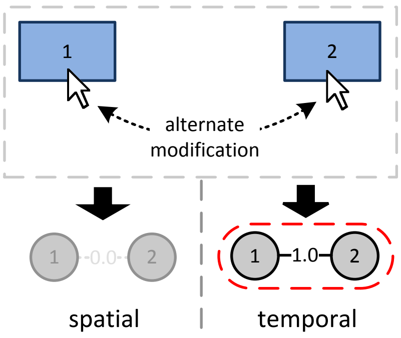


Figure 3.5: example: (pure) temporal structure formed by alternate modification of rectangular information objects

recognized that both objects were repeatedly modified in an alternate fashion. This operation pattern might indicate that there *is* an association between object 1 and 2; that is, both objects seem to be related *somehow*. Human users might have the same association in mind. This information, however, is not available to conventional spatial parsers, which limits their practical value.

3.4 Solution

These issues may only be overcome by giving up the idea that spatial and visual attributes are sufficient for detecting intended structure. In addition to properties like position, size, shape etc. further sources of information need to be considered in spatial parser designs.

We suggest that *temporal* aspects of spatial hypertext are the perfect choice for solving those problems discussed in the previous sections. A spatial parser which is “aware” of previous structures (i. e., information structures which used to exist but are not visible anymore) and “knows” about temporal dependencies between information units could (a) filter out discrete structures most likely seen by human users; (b) complete corrupted structures and (c) detect associations that are purly temporal. We expect this to lead to a significant increase in parsing accuracy, and hence higher parser performance.

Although the idea of enhancing spatial parsing by temporal information was proposed already in [12] and [36] it has never been realized. Neither there is an algorithmic design for such a temporal extension nor was it implemented in prototypical form. It is therefore still unknown whether such a *spatio-temporal parser* would perform better than a conventional spatial parser. This thesis is intended to change that.

Chapter 4

Prototype

Research results only applicable in a single application context are generally very limited, as conclusions cannot be applied to other contexts. This is why we tried to carry out our investigations as context-independent and therefore as implementation-independent as possible. To achieve that, we did not build our research algorithms on some *specific* spatial hypermedia application. This would have been too restrictive. Instead, our algorithm design was rather driven by a theoretical model of a *universal* spatial hypermedia system. In other words, in contrast to the usual practice in spatial hypermedia research we decided to come from theory to prototype, and not the other way round.

We define "typical" spatial hypermedia systems as compositions of two subsystems: (a) *Editing System* (Sect. 4.1) and (b) *Interpretation System* (Sect.4.2). In a nutshell, editing systems support creation of visual structure, whereas interpretation systems perform automatic structural analysis. Linked together they realise interactive structure creation loops. This represents the functional core of spatial hypermedia systems.

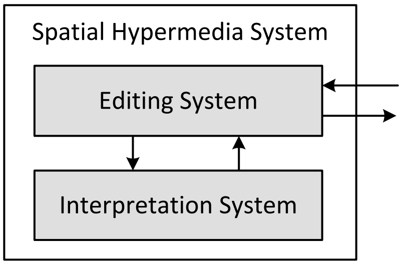


Figure 4.1: Spatial hypermedia system defined as composite of two interconnected subsystems: Editing System (Sect. 4.1) and Interpretation System (Sect.4.2)

4.1 Editing System

Spatial hypertext is not designed on a drawing board and thus is not constructed following a predefined blueprint. Instead, visual information structure emerges (Sect. 1.2). That is, users do not “engineer” formal information structure, but rather interact with objects in a visual information space to gradually develop meaningful visual expression. This is rather a creative than a constructive process. Therefore special development tools are required. So-called *editing systems* provide users with interactive access to visual information spaces and herewith support the step-by-step creation of visual structure. Thus, from an application-oriented perspective, editing systems are tools for spatial hypertext development. They therefore form the basis for any spatial hypermedia system.

In order to formally describe such systems it is crucial to understand how spatial hypertext evolves. If we can describe emerging spatial or visual expressions then we can also define how an editing system behaves.

4.1.1 Evolution of Spatial Hypertext

As far as possible, we tried to keep the following definitions implementation-independent and avoided the use of an own formal notation. The only exceptions to this are generic definitions, that is, the way we describe parameterised sets, functions or other mathematical objects.

In concrete terms, when the definition of a mathematical object X depends on one or several other objects (or parameters) $P_0, P_1, \ldots, P_n$, then we formally express this as follows:

$$X\langle P_0, P_1, \ldots, P_n\rangle$$

This notation was inspired by generic programming, hence the expression above looks quite similar to what you might know from source code templates.

Note, that we used this notation already in Sect. 2.1 when we introduced spatial hypertext languages L_{SH} (for details see page 28). So, a good example for this “template-like” notation would be ...

$$L_{SH}\langle \Sigma_{SH}, X_{SH}\rangle := \left(2^{\Sigma_{SH}} \setminus X_{SH}\right)$$

... which is nothing else than a parameterized set $\left(2^{\Sigma_{SH}} \setminus X_{SH}\right)$ or rather a set of sets determined by two factors: Σ_{SH} and X_{SH}. This could also be applied to other mathematical objects, such as tuples, relations or functions. In fact this is exactly what we do in our following definitions.

In order to keep mathematical expressions as compact as possible and therefore to facilitate readability of our formal model several basic definitions are required.

This includes the following definition of $R_s\langle\Sigma_{SH}\rangle$:

$$R_s\langle\Sigma_{SH}\rangle := \left\{R \,\middle|\, R \subseteq \big(\Sigma_{SH} \cup \{\varepsilon\}\big) \times \big(\Sigma_{SH} \cup \{\varepsilon\}\big)\right\} = 2^{\left(\Sigma_{SH}\cup\{\varepsilon\}\right)\times\left(\Sigma_{SH}\cup\{\varepsilon\}\right)} \tag{4.1}$$

According to Eq. 4.1 $R_s\langle\Sigma_{SH}\rangle$ is defined as the parameterized set of all binary relations $R \subseteq \big(\Sigma_{SH} \cup \{\varepsilon\}\big) \times \big(\Sigma_{SH} \cup \{\varepsilon\}\big)$ and therefore depends on two factors: Σ_{SH} und ε. Here, Σ_{SH} represents any set of spatial hypertext symbols and thus any spatial hypertext "alphabet" (see Eq. 2.4 in Sect. 2.1). The second parameter ε is an empty symbol (i. e., a symbolic placeholder for "nothing") and is supposed to be no element of Σ_{SH}.

As an example, for a given symbol set $\Sigma_{SH} = \{s_0, s_1, s_2, s_3\}$ a valid relation $R \in R_s\langle\Sigma_{SH}\rangle$ might look as follows:

$$\left\{\begin{matrix} (\,\varepsilon\,, s_0)\,, \\ (s_1, s_2)\,, \\ (s_3,\, \varepsilon\,) \end{matrix}\right\} \in R_s\langle\{s_0, s_1, s_2, s_3\}\rangle \tag{4.2}$$

Therefore, elements (or instances) of $R_s\langle\Sigma_{SH}\rangle$ are nothing more than sets of binary symbol tuples or rather mappings of symbols (i. e., elements on the left side are mapped to symbols on the right).

We use such relations to describe substitutions or replacement operations. In the example given above (in Eq. 4.2) we perform three substitutions at once: ε (quasi "nothing") becomes s_0, s_1 transforms into s_2 and s_3 changes to ε (i. e., s_3 is getting deleted). Thus, relation R describes a transformation of the symbol set $\mathit{Pre}_s(R) = \{s_1, s_3\}$, which is defined in Eq. 4.3, into the target set $\mathit{Post}_s(R) = \{s_0, s_2\}$, given by Eq. 4.4.

$$\mathit{Pre}_s : R_s\langle\Sigma_{SH}\rangle \to 2^{\Sigma_{SH}},\; R \mapsto \left(\bigcup_{(s,s')\in R} \{s\}\right) \setminus \varepsilon \tag{4.3}$$

$$\mathit{Post}_s : R_s\langle\Sigma_{SH}\rangle \to 2^{\Sigma_{SH}},\; R \mapsto \left(\bigcup_{(s,s')\in R} \{s'\}\right) \setminus \varepsilon \tag{4.4}$$

However, our goal here is not to describe *general* transformations on *any* symbol sets. Our aim is rather to define replacement operations on spatial hypertext artifacts. After all we want to describe formally how spatial hypertext evolves. In concrete terms, we are looking for a universal description of how artifacts or "words" of any spatial hypertext language are generated. This requires to refine or rather to constrain the previously defined set $R_s\langle\Sigma_{SH}\rangle$ as follows:

$$R_H\langle\Sigma_{SH},X_{SH}\rangle = \left\{ R \;\middle|\; \begin{array}{c} R\in R_s\langle\Sigma_{SH}\rangle,\\ R\neq\emptyset \\ \wedge \\ \left(\begin{array}{c} \nexists \left\{(a,a'),(b,b')\right\} \in \binom{R}{2} : \\ (a,b\neq\varepsilon\wedge a=b)\vee\left(a',b'\neq\varepsilon\wedge a'=b'\right) \end{array}\right) \\ \wedge \\ Pre_s(R), Post_s(R)\in 2^H,\ H\in L_{SH}\langle\Sigma_{SH},X_{SH}\rangle \end{array}\right\} \tag{4.5}$$

Unlike $R_s\langle\Sigma_{SH}\rangle$ this definition of $R_H\langle\Sigma_{SH},X_{SH}\rangle$ does not only depend on the spatial hypertext "alphabet" Σ_{SH} but also on X_{SH}. In Sect. 2.1 we defined X_{SH} as an exclusion set on $2^{\Sigma_{SH}}$. According to Eq. 2.8 X_{SH} forms together with Σ_{SH} a spatial hypertext language $L_{SH}\langle\Sigma_{SH},X_{SH}\rangle$. Thus, the definition of R_H does not just depend on a spatial hypertext "alphabet", but on a full spatial hypertext language. Our definition of $R_H\langle\Sigma_{SH},X_{SH}\rangle$ restricts $R_s\langle\Sigma_{SH}\rangle$ by three constraints:

Firstly, $R_H\langle\Sigma_{SH},X_{SH}\rangle$ may not include the empty set. Empty relations $R=\emptyset$ include no mappings of symbols and hence describe no changes. Even though such "neutral" operations (or rather transitions) have their right to exist, we will deal with them separately in later sections. For our current definitions we rather assume that $\underline{\text{H}}$ypertext-$\underline{\text{R}}$elations $\in R_H\langle\Sigma_{SH},X_{SH}\rangle$ always contain at least one symbol tuple.

Second and third constraint refer to tuple elements: There may be no unordered pair $\{(a,a'),(b,b')\}$ of binary tuples $\in R$ for which $(a,b\neq\varepsilon\wedge a=b)$ or $(a',b'\neq\varepsilon\wedge a'=b')$. This means in plain language that symbols may occur only once, on the left as well as on the right side of relations $R\in R_H\langle\Sigma_{SH},X_{SH}\rangle$. The exception to this are ε-entries.

The reason for the last restriction lies in constraint number three: $Pre_s(R)$ as well as $Post_s(R)$ must be (not necessarily proper) subsets of a valid spatial hypertext artifact. With "valid" we mean, that the respective artifact has to be element of the spatial hypertext language formed by Σ_{SH} and X_{SH}, which is $L_{SH}\langle\Sigma_{SH},X_{SH}\rangle$. Therefore $Pre_s\,(R), Post_s\,(R)\in 2^H$ with $H\in L_{SH}\langle\Sigma_{SH},X_{SH}\rangle$.

In conclusion, $R_H\langle\Sigma_{SH},X_{SH}\rangle$ represents the set of all (theoretically) possible replacement operations on spatial hypertext artifacts $H\in L_{SH}\langle\Sigma_{SH},X_{SH}\rangle$.

Regardless of how the underlying spatial hypertext language is defined, we can always distinguish between three basic operations on spatial hypertext artifacts: Artifacts, which are nothing more than flat collections of symbols, can be (1) extended by new symbols (i. e., they can “increase” in size); (2) symbols can be removed from an artifact (i. e., they can “decrease”) and (3) included symbols can be modified (by “changing” their attribute values).

These three classes of relations can be defined as subsets of $R_H\langle\Sigma_{SH}, X_{SH}\rangle$:

$$\begin{aligned}
R_{increase}\langle\Sigma_{SH}, X_{SH}\rangle &= \left\{ R \;\middle|\; \begin{array}{c} R \in R_H\langle\Sigma_{SH}, X_{SH}\rangle, \\ \forall\,(s, s') \in R:\; s = \varepsilon \wedge s' \neq \varepsilon \end{array} \right\} \\
R_{decrease}\langle\Sigma_{SH}, X_{SH}\rangle &= \left\{ R \;\middle|\; \begin{array}{c} R \in R_H\langle\Sigma_{SH}, X_{SH}\rangle, \\ \forall\,(s, s') \in R:\; s \neq \varepsilon \wedge s' = \varepsilon \end{array} \right\} \\
R_{change}\langle\Sigma_{SH}, X_{SH}\rangle &= \left\{ R \;\middle|\; \begin{array}{c} R \in R_H\langle\Sigma_{SH}, X_{SH}\rangle, \\ \forall\,(s, s') \in R:\; (s, s' \neq \varepsilon) \wedge (s \neq s') \end{array} \right\}
\end{aligned} \tag{4.6}$$

If one understands now single hypertexts $H \in L_{SH}$ as states and elements e of $R_{increase} \cup R_{decrease} \cup R_{change}$ (Eq. 4.6) as state transitions or rather as events, then we can effectively use sequences of spatial hypertext artifacts and symbol relations for describing a spatial hypertext's evolution.

For this we use the following automaton:

$$A_H\langle\Sigma_{SH}, X_{SH}, M_H\rangle = \begin{pmatrix} S_H\langle\Sigma_{SH}, X_{SH}\rangle, \\ E_H\langle\Sigma_{SH}, X_{SH}\rangle, \\ \delta_H\langle\Sigma_{SH}, X_{SH}\rangle, \\ H_{init}, \\ F_H\langle M_H\rangle \end{pmatrix}, \; M_H \subseteq L_{SH}\langle\Sigma_{SH}, X_{SH}\rangle \tag{4.7}$$

A_H is a deterministic and (potentially) infinite automaton, whose five components are defined using three parameters: Σ_{SH}, X_{SH} and M_H.

Just like Eq. 4.5 and Eq. 4.6, also the definition of this (H̲ypertext) A̲utomaton A_H depends on Σ_{SH}, X_{SH} and thus on the language $L_{SH}\langle\Sigma_{SH}, X_{SH}\rangle$.

In addition, however, A_H is also determined by a very special set of so-called “M̲ature” artifacts $M_H \subseteq L_{SH}\langle\Sigma_{SH}, X_{SH}\rangle$. These artifacts are meant to be hypertexts that include already meaningful structure and therefore should have significant information content. Thus, M_H represents a subset of $L_{SH}\langle\Sigma_{SH}, X_{SH}\rangle$ whose elements are more than just any steps in a development process. They rather should be understood as intermediate results with a significant degree of consistency.

The exact definition of M_H strongly depends on individual development context and development goals of hypertext authors. This is why, at this point, we cannot specify M_H any further.

States of A_H are nothing more than "words" of a spatial hypertext language. For this reason, we set S_H, which is the set of all possible states of A_H, equal to the language L_{SH}:

$$S_H\langle \Sigma_{SH}, X_{SH}\rangle = L_{SH}\langle \Sigma_{SH}, X_{SH}\rangle \tag{4.8}$$

Following this definition, the automaton $A_H\langle \Sigma_{SH}, X_{SH}, M_H\rangle$ could (theoretically) take up any $H \in L_{SH}\langle \Sigma_{SH}, X_{SH}\rangle$ as machine state.

Based on Eq. 4.6 we define the total set of input signals or rather events e as:

$$E_H\langle \Sigma_{SH}, X_{SH}\rangle = \left\{ e \,\middle|\, e \in \begin{pmatrix} R_{increase}\langle \Sigma_{SH}, X_{SH}\rangle \\ \cup\, R_{decrease}\langle \Sigma_{SH}, X_{SH}\rangle \\ \cup\, R_{change}\langle \Sigma_{SH}, X_{SH}\rangle \end{pmatrix} \right\} \tag{4.9}$$

Consequently, any addition, removal or modification of symbols could trigger a state transition.

How such events $e \in E_H\langle \Sigma_{SH}, X_{SH}\rangle$ make the automaton A_H "move" through $S_H\langle \Sigma_{SH}, X_{SH}\rangle$ and thus through the language $L_{SH}\langle \Sigma_{SH}, X_{SH}\rangle$, is defined by the following transition function δ_H:

$$\begin{aligned} \delta_H\langle \Sigma_{SH}, X_{SH}\rangle : \big(S_H\langle \Sigma_{SH}, X_{SH}\rangle \times E_H\langle \Sigma_{SH}, X_{SH}\rangle\big) &\rightharpoonup S_H\langle \Sigma_{SH}, X_{SH}\rangle, \\ (H, e) \mapsto \Big(Post_s(e) \cup \big(H \setminus Pre_s(e)\big)\Big) &\in S_H\langle \Sigma_{SH}, X_{SH}\rangle, \\ Pre_s(e) \subseteq H \wedge Post_s(e) \cap \big(H \setminus Pre_s(e)\big) &= \emptyset \end{aligned} \tag{4.10}$$

The transition function $\delta_H\langle \Sigma_{SH}, X_{SH}\rangle$ maps pairs of states and events (H, e) to successor states $\in S_H\langle \Sigma_{SH}, X_{SH}\rangle$ by substituting $Pre_s(e) \subseteq H$ for $Post_s(e)$. The resulting collections of symbols, or rather the resulting successor states, must be element of $S_H\langle \Sigma_{SH}, X_{SH}\rangle$ and thus have to be valid artifacts of the underlying spatial hypertext language. Furthermore, the symbols to be replaced are expected to be part of H (i. e., $Pre_s(e) \subseteq H$) and, as a last requirement, $Post_s(e)$ may not contain symbols that are already included in $(H \setminus Pre_s(e))$, which is the immutable part of H. Function δ_H is only defined if all these conditions are met.

The initial state of $A_H\langle \Sigma_{SH}, X_{SH}, M_H\rangle$ shall be equal to H_ε:

$$H_{init} = H_\varepsilon \in S_H\langle \Sigma_{SH}, X_{SH}\rangle \tag{4.11}$$

Spatial hypertexts do not fall simply from sky, but are always the result of a development process that has to start from somewhere (typically from an empty information space). Our formal equivalent to "empty space" is the empy spatial hypertext artifact H_ε (i. e., the empty set; see Sect. 2.1). Since H_ε is element of any spatial hypertext language, it is necessarily also $\in S_H\langle \Sigma_{SH}, X_{SH}\rangle$. Therefore the empty artifact may be used as initial state.

The set of final states or rather final hypertexts F_H shall be equal to the previously mentioned set of mature hypertext artifacts M_H:

$$F_H\langle M_H\rangle = M_H \tag{4.12}$$

Consequently, each spatial hypertext artifact $H \in M_H$ represents a potential final state of $A_H\langle \Sigma_{SH}, X_{SH}, M_H\rangle$.

Having defined all five components of A_H we can finally discuss its dynamics. Let $\mathcal{B}_H\langle \Sigma_{SH}, X_{SH}, M_H\rangle$ be the behaviour of our automaton A_H; that is, the set of all valid sequences of input signals (i. e., events) and their respective sequences of states (i. e., hypertexts) the automaton would pass through.

In addition, let $\mathcal{B}'_H\langle \Sigma_{SH}, X_{SH}, M_H\rangle$ be a subset of $\mathcal{B}_H\langle \Sigma_{SH}, X_{SH}, M_H\rangle$ refining the behaviour of A_H:

$$\mathcal{B}'_H\langle \Sigma_{SH}, X_{SH}, M_H\rangle \subseteq \mathcal{B}_H\langle \Sigma_{SH}, X_{SH}, M_H\rangle \tag{4.13}$$

Each element of this subset $\mathcal{B}'_H\langle \Sigma_{SH}, X_{SH}, M_H\rangle$ shall be a binary tuple of value sequences which we denote as (E, H).

E is a sequence of input signals or rather events $e_k \in E_H\langle \Sigma_{SH}, X_{SH}\rangle$. For a given time horizon $k_e \in \mathbb{N}^+$ we define E as follows:

$$\begin{aligned} E\,(0 \ldots k_e - 1) = \bigl(e_0, e_1, \ldots, e_{k_e-1}\bigr)\,, \\ e_k \in E_H\langle \Sigma_{SH}, X_{SH}\rangle,\ k = 0, 1, \ldots, k_e - 1 \end{aligned} \tag{4.14}$$

Driven by $E\,(0 \ldots k_e - 1)$ the automaton A_H passes through certain sequences of states. We denote these sequences as $H\,(0 \ldots k_e)$ and define them as:

$$\begin{aligned} H\,(0 \ldots k_e) = \bigl(H_0, H_1, \ldots, H_{k_e}\bigr)\,, & \\ H_0 = H_{init} & \\ H_{k+1} = \delta_H\langle \Sigma_{SH}, X_{SH}\rangle\,(H_k, e_k)\,, & \\ \delta_H\langle \Sigma_{SH}, X_{SH}\rangle\,(H_k, e_k)!,\ k = 0, 1, \ldots, k_e - 1 & \\ H_{k_e} \in F_H\langle M_H\rangle & \end{aligned} \tag{4.15}$$

Therefore $\mathcal{B}'_H\langle \Sigma_{SH}, X_{SH}, M_H\rangle$ describes how a spatial hypertext, encoded in $L_{SH}\langle \Sigma_{SH}, X_{SH}\rangle$, can evolve from H_ε to a (final) mature state $\in M_H$.

As pointed out already, the exact definition of M_H strongly depends on individual development context and development goals of hypertext authors. Consequently, if there is no such information available, which is the default case, then it is not possible to give a precise definition of M_H. So we must assume that any spatial hypertext could be in a form which is, from an observer's perspective, consistent and therefore potentially final. This is why in this general case A_H may be in a final state $\in M_H$ at any point in discrete time.

This allows for the following simplification:

$$A_H\langle\Sigma_{SH}, X_{SH}\rangle := A_H\langle\Sigma_{SH}, X_{SH}, L_{SH}\langle\Sigma_{SH}, X_{SH}\rangle\rangle \Leftrightarrow \begin{pmatrix} S_H\langle\Sigma_{SH}, X_{SH}\rangle, \\ E_H\langle\Sigma_{SH}, X_{SH}\rangle, \\ \delta_H\langle\Sigma_{SH}, X_{SH}\rangle, \\ H_\varepsilon \end{pmatrix} \tag{4.16}$$

With this partial default-parameterization, if you want to call it that, we effectively set F_H equal to S_H which is the underlying spatial hypertext language $L_{SH}\langle\Sigma_{SH}, X_{SH}\rangle$. This is almost like defining A_H without final states F_H.

Based on this simplification we can redefine the behaviour $\mathcal{B}_H$ of $A_H\langle\Sigma_{SH}, X_{SH}\rangle$ as follows:

$$\begin{aligned} \mathcal{B}_H\langle\Sigma_{SH}, X_{SH}\rangle &:= \mathcal{B}_H\langle\Sigma_{SH}, X_{SH}, L_{SH}\langle\Sigma_{SH}, X_{SH}\rangle\rangle \\ &= \mathcal{B}'_H\langle\Sigma_{SH}, X_{SH}, L_{SH}\langle\Sigma_{SH}, X_{SH}\rangle\rangle \end{aligned} \tag{4.17}$$

4.1.2 Editing Processes

So far we considered only dynamics of artifacts or rather "words" in spatial hypertext languages. Thus, with our considerations from previous Sect. 4.1.1 we focused solely on language attributes. In concrete terms, we described formally how spatial and visual representations of implicit structure can evolve, starting from an empty visual word.

However, in this section we are not simply interested in dynamics of language elements but rather in complete editing systems.

The behaviour of such interactive visual information spaces is not only determined by the underlying visual language. In addition to spatial and visual symbol properties there are further attributes which play an important role for editing spatial hypertext (e. g., unique object identifiers, content, usage statistics etc.). We summarize these attributes by the term "workspace meta data".

In this section we complement our previous definitions by such Meta data M.

To this end, we extend the basic component Σ_{SH}, as it was used already in our initial definition of $R_s\langle\Sigma_{SH}\rangle$ in Eq. 4.1, to a binary cartesian product: $\Sigma_{SH} \times M$. Elements of $\Sigma_{SH} \times M$ are binary tuples of spatial hypertext symbols $\in \Sigma_{SH}$ and additional meta data $\in M$. In the following we refer to such tuples as *information units* u:

$$\begin{aligned} &\forall u \in (\Sigma_{SH} \times M): \\ &\qquad u := \begin{pmatrix} u.symbol, \\ u.meta\_data \end{pmatrix}; \quad \begin{matrix} u.symbol \in \Sigma_{SH}, \\ u.meta\_data \in M \end{matrix} \end{aligned} \tag{4.18}$$

The term "information unit" was introduced already in Sect. 2.2, but in a different context. Information units $u \in (\Sigma_{SH} \times M)$, as described here, are primarily used for spatial hypertext *Representation*. Information units, as we defined them in Sect. 2.2 rather deal with its *Interpretation*. Therefore, the term "information unit" is used in connection with both, spatial parsing as well as editing systems.

Having extended Σ_{SH} to $(\Sigma_{SH} \times M)$ we can reformulate the parameterized set of symbol relations $R_s\langle\Sigma_{SH}\rangle$, as it was defined in Eq. 4.1, as follows:

$$\begin{aligned} R_u\langle\Sigma_{SH}, M\rangle &= \Big\{ R \,\Big|\, R \subseteq \big((\Sigma_{SH} \times M) \cup \{\varepsilon\}\big) \times \big((\Sigma_{SH} \times M) \cup \{\varepsilon\}\big) \Big\} \\ &= 2^{\left(\left((\Sigma_{SH} \times M) \cup \{\varepsilon\}\right) \times \left((\Sigma_{SH} \times M) \cup \{\varepsilon\}\right)\right)} \end{aligned} \tag{4.19}$$

The basic structure of this relation set is the same as in Eq. 4.1. The only difference is, that elements of binary tuples may now include both spatial and visual attributes as well as meta data.

Analogous to this, both auxiliary functions Pre_s and $Post_s$, as defined in Eq. 4.3 and Eq. 4.4, can be adapted for use with information units $u \in (\Sigma_{SH} \times M)$. Following the naming convention from Eq. 4.19 we denote these functions with Pre_u (Eq. 4.20) and $Post_u$ (Eq. 4.21) and define them as follows:

$$Pre_u : R_u\langle\Sigma_{SH}, M\rangle \to 2^{(\Sigma_{SH} \times M)},\ R \mapsto \left(\bigcup_{(u,u') \in R} \{u\} \right) \setminus \varepsilon \tag{4.20}$$

$$Post_u : R_u\langle\Sigma_{SH}, M\rangle \to 2^{(\Sigma_{SH} \times M)},\ R \mapsto \left(\bigcup_{(u,u') \in R} \{u'\} \right) \setminus \varepsilon \tag{4.21}$$

In contrast to what we did previously in Sect. 4.1.1 we are here not just dealing with collections of symbols $s \in \Sigma_{SH}$, but rather with sets of information units $u \in (\Sigma_{SH} \times M)$. Consequently, our replacement operations cannot operate directly on spatial hypertext artifacts $H \in L_{SH}\langle\Sigma_{SH}, X_{SH}\rangle$. What we need instead is an adequate abstraction of $L_{SH}\langle\Sigma_{SH}, X_{SH}\rangle$ that we will refer to as $U\langle\Sigma_{SH}, X_{SH}, M\rangle$:

$$U\langle\Sigma_{SH}, X_{SH}, M\rangle = \left\{ U \;\middle|\; \begin{array}{c} U \in 2^{(\Sigma_{SH}\times M)}, \\ \nexists \{u, u'\} \in \binom{U}{2} : u.\mathit{symbol} = u'.\mathit{symbol} \\ \wedge \\ \left(\bigcup_{u\in U} \{u.\mathit{symbol}\}\right) \in L_{SH}\langle\Sigma_{SH}, X_{SH}\rangle \end{array} \right\} \tag{4.22}$$

Actually, $U\langle\Sigma_{SH}, X_{SH}, M\rangle$ is nothing more than the power set of $(\Sigma_{SH} \times M)$ restricted by two constraints:

Firstly, $U \in U\langle\Sigma_{SH}, X_{SH}, M\rangle$ may not include any pair of information units $\{u, u'\}$ where u and u' share their symbol attribute (i. e., $u.\mathit{symbol} = u'.\mathit{symbol}$). This means, any combination of spatial and visual properties (i. e., each symbol) may occur only once in U. The reason for this can be found in the second constraint of Eq. 4.22. According to that, all symbols of U together must form a valid element of $L_{SH}\langle\Sigma_{SH}, X_{SH}\rangle$, that is a spatial hypertext artifact. Therefore, each $U \in U\langle\Sigma_{SH}, X_{SH}, M\rangle$ implicitly contains an artifact $H \in L_{SH}\langle\Sigma_{SH}, X_{SH}\rangle$.

In a certain sense, one could say that $U\langle\Sigma_{SH}, X_{SH}, M\rangle$ represents a spatial hypertext language $L_{SH}\langle\Sigma_{SH}, X_{SH}\rangle$ "enriched" by meta data.

This also includes an empty element; based on the model of $H_\varepsilon \in L_{SH}\langle\Sigma_{SH}, X_{SH}\rangle$. For this we use the empty set of information units U_ε:

$$U_\varepsilon := \emptyset \in U\langle\Sigma_{SH}, X_{SH}, M\rangle \tag{4.23}$$

If one substitutes in Eq. 4.5 all components that are defined on symbols, such as $R_s\langle\Sigma_{SH}\rangle$, Pre_s, Post_s and $L_{SH}\langle\Sigma_{SH}, X_{SH}\rangle$, for $R_u\langle\Sigma_{SH}, M\rangle$, Pre_u, Post_u, and $U\langle\Sigma_{SH}, X_{SH}, M\rangle$, then we get the following set of relations:

$$R_U\langle\Sigma_{SH}, X_{SH}, M\rangle = \left\{ R \;\middle|\; \begin{array}{c} R \in R_u\langle\Sigma_{SH}, M\rangle, \\ R \neq \emptyset \\ \wedge \\ \left(\begin{array}{c} \nexists \left\{ \left(a, a'\right), \left(b, b'\right) \right\} \in \binom{R}{2} : \\ \left(a, b \neq \varepsilon \wedge a = b\right) \vee \left(a', b' \neq \varepsilon \wedge a' = b'\right) \end{array} \right) \\ \wedge \\ \mathit{Pre}_u(R), \mathit{Post}_u(R) \in 2^U,\; U \in U\langle\Sigma_{SH}, X_{SH}, M\rangle \end{array} \right\} \tag{4.24}$$

Just like $R_H\langle\Sigma_{SH}, X_{SH}\rangle$ in Eq. 4.6, also $R_U\langle\Sigma_{SH}, X_{SH}, M\rangle$ can be further refined by adding constraints. This way, we define three sub-classes of operations.

Unlike specified in Eq. 4.6 these operations are now called *create*, *delete* and *modify* rather than *increase*, *decrease* or *change*:

$$
\begin{aligned}
R_{create}\langle\Sigma_{SH}, X_{SH}, M\rangle &= \left\{ R \;\middle|\; \begin{array}{c} R \in R_U\langle\Sigma_{SH}, X_{SH}, M\rangle, \\ \forall \big(u, u'\big) \in R:\; u = \varepsilon \wedge u' \neq \varepsilon \end{array} \right\} \\
R_{delete}\langle\Sigma_{SH}, X_{SH}, M\rangle &= \left\{ R \;\middle|\; \begin{array}{c} R \in R_U\langle\Sigma_{SH}, X_{SH}, M\rangle, \\ \forall \big(u, u'\big) \in R:\; u \neq \varepsilon \wedge u' = \varepsilon \end{array} \right\} \\
R_{modify}\langle\Sigma_{SH}, X_{SH}, M\rangle &= \left\{ R \;\middle|\; \begin{array}{c} R \in R_U\langle\Sigma_{SH}, X_{SH}, M\rangle, \\ \forall \big(u, u'\big) \in R:\; \big(u, u' \neq \varepsilon\big) \wedge \big(u \neq u'\big) \end{array} \right\}
\end{aligned}
\tag{4.25}
$$

Again, $U\langle\Sigma_{SH}, X_{SH}, M\rangle$, Pre_u, $Post_u$, as well as the relations from Eq. 4.25 can be used to define an automaton. Unlike A_H (Eq. 4.7), however, that new automaton does not build on spatial hypertext artifacts H, but on sets of information units U. Consequently, we denote such information <u>U</u>nit <u>A</u>utomata with A_U:

$$
A_U\langle\Sigma_{SH}, X_{SH}, M\rangle = \begin{pmatrix} S_U\langle\Sigma_{SH}, X_{SH}, M\rangle, \\ E_U\langle\Sigma_{SH}, X_{SH}, M\rangle, \\ \delta_U\langle\Sigma_{SH}, X_{SH}, M\rangle, \\ U_{init} \end{pmatrix} \tag{4.26}
$$

Here, the set of possible states of A_U shall be equal to $U\langle\Sigma_{SH}, X_{SH}, M\rangle$:

$$
S_U\langle\Sigma_{SH}, X_{SH}, M\rangle = U\langle\Sigma_{SH}, X_{SH}, M\rangle \tag{4.27}
$$

Similar to E_H in Eq. 4.9, the set of input signals or rather events E_U is defined as a union of relations: R_{create}, R_{delete} and R_{modify}

$$
E_U\langle\Sigma_{SH}, X_{SH}, M\rangle = \left\{ e \;\middle|\; e \in \begin{pmatrix} R_{create}\langle\Sigma_{SH}, X_{SH}, M\rangle \\ \cup\, R_{delete}\langle\Sigma_{SH}, X_{SH}, M\rangle \\ \cup\, R_{modify}\langle\Sigma_{SH}, X_{SH}, M\rangle \end{pmatrix} \right\} \tag{4.28}
$$

Apparently, there are clear parallels to the previous definition of A_H in Sect. 4.1.1. This also applies to the state transition function δ. In concrete terms, δ_H from Eq. 4.10 and the following definition of δ_U (Eq. 4.29) differ only in their underlying sets of states and events. This is why the following partial mapping looks very similar to what was previously defined in Eq. 4.10:

$$\begin{aligned}
&\delta_U\langle\Sigma_{SH}, X_{SH}, M\rangle : \\
&\big(S_U\langle\Sigma_{SH}, X_{SH}, M\rangle \times E_U\langle\Sigma_{SH}, X_{SH}, M\rangle\big) \rightharpoonup S_U\langle\Sigma_{SH}, X_{SH}, M\rangle, \\
&\qquad (U, e) \mapsto \Big(Post_u(e) \cup \big(U \setminus Pre_u(e)\big)\Big) \in S_U\langle\Sigma_{SH}, X_{SH}, M\rangle, \\
&\qquad\qquad Pre_u(e) \subseteq U \wedge Post_u(e) \cap \big(U \setminus Pre_u(e)\big) = \emptyset
\end{aligned} \tag{4.29}$$

The fourth and last component of automaton A_U is its initial state or rather its initial set of information units U_{init} (Eq. 4.30). That initial state shall be equal to U_ε from Eq. 4.23. Final states are not defined. With this we follow the simplification from Eq. 4.16.

$$U_{init} = U_\varepsilon \in S_U\langle\Sigma_{SH}, X_{SH}, M\rangle \tag{4.30}$$

Finally, let $\mathcal{B}_U\langle\Sigma_{SH}, X_{SH}, M\rangle$ be the behaviour of $A_U\langle\Sigma_{SH}, X_{SH}, M\rangle$; that is, the set of all ingoing sequences of events and their respective states (or rather, collections of information units). Elements of $\mathcal{B}_U\langle\Sigma_{SH}, X_{SH}, M\rangle$ are binary tuples of event and state sequences denoted as (E, U).

For a given time horizon $k_e \in \mathbb{N}^+$ such E is define as:

$$\begin{aligned}
&E\,(0 \ldots k_e - 1) = \big(e_0, e_1, \ldots, e_{k_e-1}\big)\,, \\
&\qquad e_k \in E_U\langle\Sigma_{SH}, X_{SH}, M\rangle,\ k = 0, 1, \ldots, k_e - 1
\end{aligned} \tag{4.31}$$

The sequence of states A_U passes through for such an input $E\,(0 \ldots k_e - 1)$ is denoted as $U\,(0 \ldots k_e)$ and can be formally described as follows:

$$\begin{aligned}
&U\,(0 \ldots k_e) = \big(U_0, U_1, \ldots, U_{k_e}\big)\,, \\
&\qquad U_0 = U_{init} \\
&\qquad U_{k+1} = \delta_U\langle\Sigma_{SH}, X_{SH}, M\rangle\,(U_k, e_k)\,, \\
&\qquad\qquad \delta_U\langle\Sigma_{SH}, X_{SH}, M\rangle\,(U_k, e_k)!\,, \\
&\qquad\qquad k = 0, 1, \ldots, k_e - 1
\end{aligned} \tag{4.32}$$

Each tuple $(E, U) \in \mathcal{B}_U\langle\Sigma_{SH}, X_{SH}, M\rangle$ that fulfills both conditions, Eq. 4.31 and Eq. 4.32, is called an *Editing Process*!

4.1.3 Workspace Model

Object attributes that can be found in every spatial hypermedia workspace and therefore also in any editing system are: (1) object keys (i. e., unique object identifiers) and (2) node content (i. e., "payload" of information units).

Certainly, there may be further attributes, such as author information, usage statistics etc. Those, however, do not necessarily need to be implemented in a system. As for keys and content, on the other hand, we can be certain that in some form they can be found in every visual information space. They may be realised in different ways (e. g., as integers, texts etc.) but they still share common semantics; that is, in principle they serve the same purpose:

Object keys are used for unique identification of information units and therefore make it possible to associate or rather to link objects with each other. Node content, again, can be understood as "payload" of information units, that is, as what makes a simple object a carrier of information. Since there is no hypermedia system without nodes and links, both node content as well as keys are inevitably part of any real spatial hypermedia system. For this reason we decided to integrate both attributes into our theoretical model.

To this end, we substitute the previously introduced meta data symbol M for the cartesian product of two new set-parameters: $(K \times C)$. Here, K stands for <u>K</u>eys and C means <u>C</u>ontent. Therefore, that unary attribute $u.meta\_data$, which we defined in Eq. 4.18, changes into a binary tuple of key und content:

$$\forall u \in \big(\Sigma_{SH} \times (K \times C)\big) : \quad u := \left(\begin{array}{c} u.symbol, \\ \left(\begin{array}{c} u.key, \\ u.content \end{array} \right) \end{array} \right) ; \quad \begin{array}{c} u.symbol \in \Sigma_{SH}, \\ u.key \in K, \\ u.content \in C \end{array} \tag{4.33}$$

An appropriate derivative of $U\langle\Sigma_{SH}, X_{SH}, M\rangle$, which accepts both set-parameters K and C instead of M, shall be defined as follows:

$$U\langle\Sigma_{SH}, X_{SH}, K, C\rangle = \left\{ U \;\middle|\; \begin{array}{c} U \in U\langle\Sigma_{SH}, X_{SH}, (K \times C)\rangle, \\ \nexists \left\{u, u'\right\} \in \binom{U}{2} : u.key = u'.key \end{array} \right\} \tag{4.34}$$

This partial refinement of the parameterized set $U\langle\Sigma_{SH}, X_{SH}, M\rangle$ (Eq. 4.22) by $M = (K \times C)$ is combined with the constraint, that keys $\in K$ have to be unique in each $U \in U\langle\Sigma_{SH}, X_{SH}, (K \times C)\rangle$. This means, each $u \in U$ must be clearly identifiable by $u.key$. Thus, there may be no unordered pair $\{u, u'\}$ in U for which $u.key = u'.key$.

R_U is defined similarly:

$$R_U\langle\Sigma_{SH}, X_{SH}, K, C\rangle = \left\{ R \;\middle|\; \begin{array}{c} R \in R_U\langle\Sigma_{SH}, X_{SH}, (K \times C)\rangle, \\ \mathit{Pre}_u(R), \mathit{Post}_u(R) \in 2^U, \\ U \in U\langle\Sigma_{SH}, X_{SH}, K, C\rangle \end{array} \right\} \tag{4.35}$$

Also here we substitute M for $(K \times C)$ and hence we refine the given superset of relations $R_U\langle\Sigma_{SH}, X_{SH}, M\rangle$ to a subset $R_U\langle\Sigma_{SH}, X_{SH}, (K \times C)\rangle$. Replacement operations R described this way may only be defined on elements of $U\langle\Sigma_{SH}, X_{SH}, K, C\rangle$. This is why our definition in Eq. 4.35 also includes the constraints $\mathit{Pre}_u(R), \mathit{Post}_u(R) \in 2^U$ and $U \in U\langle\Sigma_{SH}, X_{SH}, K, C\rangle$.

When we intersect now $R_U\langle\Sigma_{SH}, X_{SH}, K, C\rangle$ with relation sets R_{create}, R_{delete} and R_{modify} (which were originally defined in Eq. 4.25), and when we substitute M for $(K \times C)$ again, then we can apply the previously known replacement operations for creation, modification and deletion of information units also on elements of $U\langle\Sigma_{SH}, X_{SH}, K, C\rangle$. For R_{create} and R_{delete} this would look as follows:

$$\begin{aligned} R_{create}\langle\Sigma_{SH}, X_{SH}, K, C\rangle &= R_U\langle\Sigma_{SH}, X_{SH}, K, C\rangle \cap R_{create}\langle\Sigma_{SH}, X_{SH}, (K \times C)\rangle \\ R_{delete}\langle\Sigma_{SH}, X_{SH}, K, C\rangle &= R_U\langle\Sigma_{SH}, X_{SH}, K, C\rangle \cap R_{delete}\langle\Sigma_{SH}, X_{SH}, (K \times C)\rangle \end{aligned} \tag{4.36}$$

According to this, both R_{create} as well as R_{delete} are defined as intersections of $R_U\langle\Sigma_{SH}, X_{SH}, K, C\rangle$ with partially refined relation sets from Eq. 4.25. With this we effectively describe *Types* of relations that combine properties of both $R_U\langle\Sigma_{SH}, X_{SH}, K, C\rangle$ as well as of R_{create} or R_{delete}. This is a simple form of multiple inheritance.

In a similar way we can describe *modify*-relations. Just like R_{create} and R_{delete} also R_{modify} can be defined by intersecting two sets of relations: in this case $R_U\langle\Sigma_{SH}, X_{SH}, K, C\rangle$ and $R_{modify}\langle\Sigma_{SH}, X_{SH}, (K \times C)\rangle$. Also R_{modify} "inherits" properties from $R_U\langle\Sigma_{SH}, X_{SH}, K, C\rangle$ and from its counterpart in Eq. 4.25. Here we add the constraint, that information units may only be mapped to units with the same key. This means, for each binary tuple (u, u') which is an element of a modify relation $R \in R_{modify}\langle\Sigma_{SH}, X_{SH}, K, C\rangle$ we expect that $u.key = u'.key$.

The purpose of this constraint is to ensure that information units keep their identity. Otherwise R_{modify} would not describe modifications but rather implicit deletion and recreation. Semantically, this would not be what we intend to express with R_{modify}.

$$R_{modify}\langle \Sigma_{SH}, X_{SH}, K, C\rangle = \left\{ R \;\middle|\; \begin{array}{c} R \in \left(\begin{array}{c} R_U\langle \Sigma_{SH}, X_{SH}, K, C\rangle \\ \cap \\ R_{modify}\langle \Sigma_{SH}, X_{SH}, (K \times C)\rangle \end{array} \right), \\ \forall \left(u, u'\right) \in R :\; u.key = u'.key \end{array} \right\} \tag{4.37}$$

Relations included in $R_{modify}\langle \Sigma_{SH}, X_{SH}, K, C\rangle$ describe changes of both symbols as well as content. It may be useful now to view both categories of replacement operations separately. This can be achieved by mapping both attributes *u.content* and *u.symbol* to identical values (just as we did with *u.key*). More precisely, we extend our previous definition from *Eq.* 4.37 by constraints ...

$$\begin{array}{c} \forall (u, u') \in R : u.content = u'.content \\ \text{or} \\ \forall (u, u') \in R : u.symbol = u'.symbol \end{array}$$

This reduces R_{modify} to those relations, that either describe only modifications of symbols (Eq. 4.38) or change of content (Eq. 4.39).

$$R_{change\_symbol}\langle \Sigma_{SH}, X_{SH}, K, C\rangle = \left\{ R \;\middle|\; \begin{array}{c} R \in R_{modify}\langle \Sigma_{SH}, X_{SH}, K, C\rangle, \\ \forall \left(u, u'\right) \in R : \\ u.content = u'.content \end{array} \right\} \tag{4.38}$$

$$R_{change\_content}\langle \Sigma_{SH}, X_{SH}, K, C\rangle = \left\{ R \;\middle|\; \begin{array}{c} R \in R_{modify}\langle \Sigma_{SH}, X_{SH}, K, C\rangle, \\ \forall \left(u, u'\right) \in R : \\ u.symbol = u'.symbol \end{array} \right\} \tag{4.39}$$

No matter if element of $R_H\langle \Sigma_{SH}, X_{SH}\rangle$ (Eq. 4.5), $R_U\langle \Sigma_{SH}, X_{SH}, M\rangle$ (Eq. 4.24) or now of $R_U\langle \Sigma_{SH}, X_{SH}, K, C\rangle$, relations always had to contain at least one binary tuple (i. e., $R \neq \emptyset$). Replacement operations, that replace nothing were not allowed. We intend to change that with the following definition of R_ε:

$$R_\varepsilon = \emptyset \tag{4.40}$$

R_ε is an empty set of binary tuples and thus an empty binary relation. Empty relations describe no changes and therefore can be used as “neutral” transitions in a dynamic system, that is, as switching operations that do not change the current system state. This is exactly what R_ε is used for in the following automaton.

As we could see already in previous sections, we can use definitions of sets, such as Eq. 4.22, and relations, like in Eq. 4.24, to create system models in form of deterministic automata. Since Eq. 4.34 builds on Eq. 4.22 and Eq. 4.35 is based on Eq. 4.24 we can do that also for the $(K \times C)$-extension from this section. We denote such automata that build on workspace meta data K and C as W̲orkspace A̲utomata A_W:

$$A_W\langle \Sigma_{SH}, X_{SH}, K, C, E_{symbol}\rangle = \begin{pmatrix} S_W\langle \Sigma_{SH}, X_{SH}, K, C\rangle, \\ E_W\langle \Sigma_{SH}, X_{SH}, K, C, E_{symbol}\rangle, \\ O_W\langle \Sigma_{SH}, X_{SH}, K, C\rangle, \\ T_W\langle \Sigma_{SH}, X_{SH}, K, C, E_{symbol}\rangle, \\ G_W\langle \Sigma_{SH}, X_{SH}, K, C, E_{symbol}\rangle, \\ s_{init} \end{pmatrix} \tag{4.41}$$

Unlike A_H (Eq. 4.7) and A_U (Eq. 4.26) the workspace automaton A_W is not only determined by Σ_{SH}, X_{SH} or by an abstract set of meta data M. Instead the given list of parameters in Eq. 4.41 comprises, in addition to Σ_{SH} and X_{SH}, also K, C and E_{symbol}. Here, parameters K and C replace the abstract meta data symbol M (just like in Eq. 4.34 or in Eq. 4.35). E_{symbol}, on the other hand, is a symbolic placeholder for a set of symbol events and therefore has to be subset of $R_{change\_symbol}\langle \Sigma_{SH}, X_{SH}, K, C\rangle$ (Eq. 4.38):

$$E_{symbol} \subset R_{change\_symbol}\langle \Sigma_{SH}, X_{SH}, K, C\rangle \tag{4.42}$$

This extended parameter list allows us to determine which operations, in addition to construction and destruction, are allowed on symbols included in information units. This makes it easy to adapt the behaviour of A_W to concrete spatial hypertext languages. You only need to define appropriate derivatives of R_{change_symbol}. This means, you only have to extend the definition of R_{change_symbol} with constraints that describe exactly how symbol attributes may change in certain language(s). Examples for this will be presented later on in Sect. 4.1.4.

Workspace automata A_W may be in the following states S_W:

$$S_W\langle \Sigma_{SH}, X_{SH}, K, C\rangle = \left\{ s := (U, V) \;\middle|\; \begin{matrix} (U, V) \in \left(U\langle \Sigma_{SH}, X_{SH}, K, C\rangle \times 2^K\right), \\ \left(\bigcup_{u \in U} \{u.key\}\right) \subseteq V \end{matrix} \right\} \tag{4.43}$$

Unlike S_H in Eq. 4.8 and S_U in Eq. 4.27 states $s \in S_W$ are not simply defined as sets of spatial hypertext symbols $H \in L_{SH}\langle \Sigma_{SH}, X_{SH}\rangle$ or as sets of information units $U \in U\langle \Sigma_{SH}, X_{SH}, M\rangle$. We rather define them as binary tuples (U, V), with $U \in U\langle \Sigma_{SH}, X_{SH}, K, C\rangle$ and $V \in 2^K$. Here, U represents, as before, a set of information units $u \in (\Sigma_{SH} \times (K \times C))$. Keys $\in K$ uniquely identify these objects and are collected in V. More precisely, the key set V contains all identifiers, that were assigned to information units until state $s = (U, V)$ and therefore cannot be used for labelling of new objects. This also includes keys of information units that used to exist in the past but are no element of U anymore at present state $s = (U, V)$. This also explains the second constraint in Eq. 4.43: $(\bigcup_{u \in U}\{u.key\}) \subseteq V$. According to this, all $u.key$ of current $u \in U$ must be included in V (i. e., they must be marked as "in use"). However, the constraint also says that not necessarily all keys $\in V$ must also be assigned to objects in U; thus V may also include "deprecated" keys. This allows us to keep track of assigned and unassigned object identifiers and therefore helps to ensure that new objects always get a key that is not in use already. This is an aspect of editing spatial hypertext that was not considered in previous Sect. 4.1.1 and Sect. 4.1.2.

The initial state s_{init} of our workspace automaton shall be equal to $(U_\varepsilon, \emptyset)$:

$$s_{init} = \big(U_\varepsilon, \emptyset\big) \in S_W\langle \Sigma_{SH}, X_{SH}, K, C\rangle \tag{4.44}$$

Thus, the starting point of any sequence of state transitions in A_W is always a binary tuple of two components: an empty set of information units U_ε (Eq. 4.23) and consequently also an empty set of keys $\emptyset \subset K$. Without objects there are also no identifiers.

Similar to E_H in Eq. 4.9 and E_U in Eq. 4.28 the set of workspace events E_W is defined as a union of replacement operations:

$$E_W\langle \Sigma_{SH}, X_{SH}, K, C, E_{symbol}\rangle = \left\{ e \;\middle|\; e \in \left(\begin{array}{l} \quad R_{create}\langle \Sigma_{SH}, X_{SH}, K, C\rangle \\ \cup\, R_{delete}\langle \Sigma_{SH}, X_{SH}, K, C\rangle \\ \cup\, R_{change\_content}\langle \Sigma_{SH}, X_{SH}, K, C\rangle \\ \cup\, \{R_\varepsilon\} \\ \cup\, E_{symbol} \end{array} \right) \right\} \tag{4.45}$$

The essential difference to E_H and E_U is that E_W includes, in addition to the already known relations such as R_{create} or R_{delete}, the empty relation or rather the empty event R_ε from Eq. 4.40 as well as the previously mentioned parameter E_{symbol} (Eq. 4.42). Note, that E_{symbol} was defined as an exchangeable set of symbol events $\in R_{change\_symbol}\langle \Sigma_{SH}, X_{SH}, K, C\rangle$.

Like δ_H and δ_U also A_W's transition function maps binary tuples of states and events to successor states. Note, however, that unlike Eq. 4.10 and Eq. 4.29, this transition function is not denoted with δ. We use the name T_W instead:

$$T_W\langle \Sigma_{SH}, X_{SH}, K, C, E_{symbol}\rangle : \left(\begin{array}{l} \quad S_W\langle \Sigma_{SH}, X_{SH}, K, C\rangle \\ \times\, E_W\langle \Sigma_{SH}, X_{SH}, K, C, E_{symbol}\rangle \end{array} \right) \rightharpoonup S_W\langle \Sigma_{SH}, X_{SH}, K, C\rangle \tag{4.46}$$

This partial mapping is defined as follows:

$$\begin{aligned}
&\big((U, V)\,, e\big) \mapsto \big(U', V'\big)\,,\\
&\qquad U' = \Big(\mathit{Post}_u(e) \cup \big(U \setminus \mathit{Pre}_u(e)\big)\Big) \in U\langle\Sigma_{SH}, X_{SH}, K, C\rangle\\
&\qquad V' = \begin{cases} V \cup \left(\displaystyle\bigcup_{u \in \mathit{Post}_u(e)} \{u.\mathit{key}\}\right) & \text{, if } e \in R_{create}\langle\Sigma_{SH}, X_{SH}, K, C\rangle \\ V & \text{, else}\end{cases}\\
&\text{where:}\\
&\quad \big(\mathit{Pre}_u(e) \subseteq U\big) \wedge \Big(\mathit{Post}_u(e) \cap \big(U \setminus \mathit{Pre}_u(e)\big) = \emptyset\Big)\\
&\quad \text{and if } e \in R_{create}\langle\Sigma_{SH}, X_{SH}, K, C\rangle \text{ then } \nexists u \in \mathit{Post}_u(e) : u.\mathit{key} \in V
\end{aligned} \tag{4.47}$$

Driven by workspace events $e \in E_W$, T_W maps tuples $(U, V) \in S_W$ to ordered pairs $(U', V') \in S_W$. For mapping U to U', we replace in U subset $\mathit{Pre}_u(e)$ by $\mathit{Post}_u(e)$ which effectively creates a new element of $U\langle\Sigma_{SH}, X_{SH}, K, C\rangle$:

$$U' = \Big(\mathit{Post}_u(e) \cup \big(U \setminus \mathit{Pre}_u(e)\big)\Big) \in U\langle\Sigma_{SH}, X_{SH}, K, C\rangle \tag{4.48}$$

V' is defined by cases:

$$V' = \begin{cases} V \cup \left(\displaystyle\bigcup_{u \in \mathit{Post}_u(e)} \{u.\mathit{key}\}\right) & \text{, if } e \in R_{create}\langle\Sigma_{SH}, X_{SH}, K, C\rangle \\ V & \text{, else}\end{cases} \tag{4.49}$$

If e is a construction event (i. e., $e \in R_{create}\langle\Sigma_{SH}, X_{SH}, K, C\rangle$; see Eq. 4.36) then it shall be presumed that no $u \in \mathit{Post}_u(e)$ has already been recorded in V. Or, to express it differently: $\nexists u \in \mathit{Post}_u(e) : u.\mathit{key} \in V$. In this case V needs to be expanded. Otherwise, if $e \notin R_{create}\langle\Sigma_{SH}, X_{SH}, K, C\rangle$ then V remains unchanged. In this case we set $V' = V$.

What mainly differentiates A_W from A_H and A_U is its ability to generate output. This means, our workspace automaton A_W does not only react on ingoing editing events with (hidden) state transitions but it also generates signals that can be detected from the outside. In concrete terms, A_W notifies its observers of every single state change.

This is why that six-tuple given in Eq. 4.41 includes, in addition to already known components, such as sets of states and events etc., two other objects: G_W and O_W. G_W defines how A_W $\underline{\text{G}}$enerates output signals. O_W shall be the basic set of that $\underline{\text{O}}$utput.

The generating (or rather output) function G_W accepts tuples of states and events $(s, e) \in (S_W \times E_W)$ and provides the value calculated by $T_W(s, e)$ as output $\in O_W$:

$$\begin{aligned}
&G_W\langle \Sigma_{SH}, X_{SH}, K, C, E_{symbol}\rangle : \\
&\begin{pmatrix} S_W\langle \Sigma_{SH}, X_{SH}, K, C\rangle \\ \times\, E_W\langle \Sigma_{SH}, X_{SH}, K, C, E_{symbol}\rangle \end{pmatrix} \rightharpoonup O_W\langle \Sigma_{SH}, X_{SH}, K, C\rangle, \\
&(s, e) \mapsto T_W\langle \Sigma_{SH}, X_{SH}, K, C, E_{symbol}\rangle(s, e)
\end{aligned} \tag{4.50}$$

According to this, G_W calculates and returns the successor state to s without changing its value. It therefore makes sense to set output signals O_W equal to states S_W. Each machine state of A_W is also a potential output signal:

$$O_W\langle \Sigma_{SH}, X_{SH}, K, C\rangle = S_W\langle \Sigma_{SH}, X_{SH}, K, C\rangle \tag{4.51}$$

Similar to Eq. 4.14 and Eq. 4.31 also the input of A_W shall be defined as tuples of input signals (or rather events) $e \in E_W$:

$$\begin{aligned}
&E\,(0 \dots k_e) = \left(e_0, e_1, \dots, e_{k_e}\right), \\
&\qquad e_k \in E_W\langle \Sigma_{SH}, X_{SH}, K, C, E_{symbol}\rangle,\ k = 0, 1, \dots, k_e
\end{aligned} \tag{4.52}$$

As pointed out already, there are two ways how workspace automata A_W react on such sequences of events $E\,(0 \dots k_e)$: Firstly, they pass through a hidden sequence of states $S\,(0 \dots k_e + 1)$; see Eq. 4.53. In this respect the definition of A_W is not different from A_H (Eq. 4.15) or A_U (Eq. 4.32).

$$\begin{aligned}
S\,(0 \dots k_e + 1) &= \left(s_0, s_1, \dots, s_{k_e+1}\right), \\
s_0 &= s_{init} \\
s_{k+1} &= T_W\langle \Sigma_{SH}, X_{SH}, K, C, E_{symbol}\rangle\,(s_k, e_k), \\
&\quad T_W\langle \Sigma_{SH}, X_{SH}, K, C, E_{symbol}\rangle\,(s_k, e_k)!\,, \\
&\quad k = 0, 1, \dots, k_e
\end{aligned} \tag{4.53}$$

Secondly, workspace automata A_W also provide these hidden states as output and thus notify their system environment (i. e., their external observers) of internal state changes. This is why we denote such sequential W̲orkspace-output as $W\,(0 \dots k_e)$:

$$\begin{aligned}
W\,(0 \dots k_e) &= \left(W_0, W_1, \dots, W_{k_e}\right), \\
W_k &= G_W\langle \Sigma_{SH}, X_{SH}, K, C, E_{symbol}\rangle\,(s_k, e_k), \\
&\quad G_W\langle \Sigma_{SH}, X_{SH}, K, C, E_{symbol}\rangle\,(s_k, e_k)!\,, \\
&\quad k = 0, 1, \dots, k_e
\end{aligned} \tag{4.54}$$

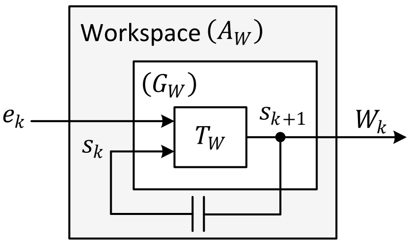
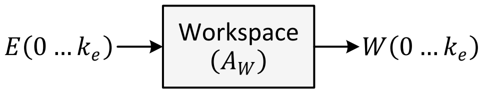


Figure 4.2: Abstract blockdiagram of a dynamic workspace system, defined as workspace automaton A_W. Functions T_W and G_W are illustrated as system blocks. The given pair of vertical lines at the bottom represents a memory component.

In a nutshell, workspace automata A_W transform ingoing sequences of edit events $E\,(0 \ldots k_e)$ into outgoing sequences of workspaces $W\,(0 \ldots k_e)$:



A more detailed illustration of such workspace systems, which includes both automata functions T_W (Eq. 4.46), G_W (Eq. 4.50) as well as a state memory can be found in Fig. 4.2. Such workspace models A_W form the basis for our definition of editing systems:

According to Fig. 4.3, editing systems extend workspaces by viewer and controller components and hence build on the MVC-pattern. Thus we can easily summarize an editing system's behaviour as follows: Viewing components receive sequences of user interface activities, such as pressing left mouse button, moving the mouse and releasing the button again, and pass them on to a controlling module. The controller accepts those sequences of activities and translates them into adequate commands (or editing events $e_k \in E_W$). These events are then forwarded as input signals to a workspace model A_W. That model switches to a new state, as described in Eq. 4.53 or Eq. 4.47 respectively, and finally publishes the updated status W_k to its observers; that is view and controller. Thus, one could understand editing systems as visual dynamic systems that are driven by user activities.

For easier legibility we define:

$$\forall W \in O_W\langle \Sigma_{SH}, X_{SH}, K, C\rangle :$$
$$W := \begin{pmatrix} W.info\_units, \\ W.keys\_in\_use \end{pmatrix}; \quad \begin{array}{c} W.info\_units \in U\langle \Sigma_{SH}, X_{SH}, K, C\rangle, \\ W.keys\_in\_use \in 2^K \end{array} \tag{4.55}$$

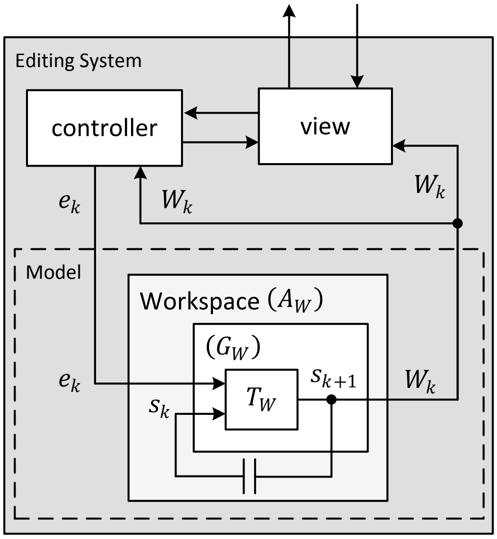


Figure 4.3: Editing system illustrated as a composite of workspace model A_W (Eq. 4.41), view and controller component

4.1.4 Specialization

In the last section we defined both workspaces and editing systems for *any* spatial hypertext language $L_{SH}\langle\Sigma_{SH}, X_{SH}\rangle$. We neither specified concrete spatial hypertext symbols Σ_{SH} nor was any specific exclusion set X_{SH} defined. Consequently, our system model above (Fig. 4.3) leaves it open which spatial and visual attributes to use for developing spatial hypertext. We also limited definitions for unique object identification and for content handling to what is strictly necessary (see Eq. 4.33, Eq. 4.39, or Eq. 4.47). This makes our system model highly adaptable to a variety of practical implementations, including our research prototype. For our test implementation we have chosen the following visual language $L^{\#}_{SH}$:

$$
\begin{aligned}
L^{\#}_{SH} &= \left(\Sigma^{\#*}_{SH} \setminus X^{\#}_{SH}\right)\\
&\qquad \Sigma^{\#*}_{SH} = \left\{H \middle| H \subseteq \Sigma^{\#}_{SH}\right\} = 2^{\Sigma^{\#}_{SH}}\\
&\qquad\qquad \Sigma^{\#}_{SH} = \left(A_{position} \times A_{size} \times A_{orientation} \times A_{shape} \times A_{color}\right)\\
&\qquad X^{\#}_{SH} = \emptyset\\
\Rightarrow L^{\#}_{SH} &= \left(2^{\Sigma^{\#}_{SH}} \setminus \emptyset\right) = 2^{\Sigma^{\#}_{SH}} = 2^{\left(A_{position} \times A_{size} \times A_{orientation} \times A_{shape} \times A_{color}\right)}
\end{aligned}
\tag{4.56}
$$

Here we follow our previous definitions from Eq. 2.7 in Sect. 2.1.

Eq. 4.56 includes five basic attribute types: position, size and orientation of objects as well as their shape and color. Further attributes, such as surface texture, line style etc., would have been too specific and would unnecessarily complicate the following definitions. For the same reason we defined no constraints on $L_{SH}^{\#}$, that is, exclusion set $X_{SH}^{\#}$ was set to $\emptyset$.

The above mentioned attribute types are defined as follows:

$$\begin{aligned} A_{position} &= \left\{(x,y,z)\middle| x,y \in \mathbb{Z},\ z \in \mathbb{N}_0\right\} \\ A_{size} &= \left\{(w,h)\middle| w,h \in \mathbb{N}^+\right\} \\ A_{orientation} &= \{AXIS\_ALIGNED\} \\ A_{shape} &= \{RECTANGLE, ELLIPSE\} \\ A_{color} &= \left\{(r,g,b)\middle| r,b,g \in \{0,1,\dots,255\}\right\} \end{aligned} \tag{4.57}$$

$A_{position}$ is defined as a set of 2- or rather 2.5-dimensional vectors. Here, x, y are position coordinates in a two-dimensional (discrete) grid $\mathbb{Z} \times \mathbb{Z}$. The depth coordinate $z \geq 0$ indicates an object layer. Thus, $z = 0$ identifies the bottom-level in an information space.

The given size-attribute is defined as proportions (w, h), where both width w as well as height h may only take values > 0. This is intended to guarantee that objects in an information space always have a (positive) spatial extent.

For the sake of simplicity we do not support object rotation, hence objects shall be axis-aligned. For this reason, $A_{orientation}$ comprises of only a single element, that is the symbolic placeholder *AXIS_ALIGNED*.

In A_{shape} we distinguish between rectangular and ellipsoidal shapes, represented by two discrete flags *RECTANGLE* and *ELLIPSE*. Although further shapes could be specified they are not necessarily required for our prototypical system.

Finally, elements of A_{color} are defined as rgb colors, that is, as triples (r, g, b) of integral values $\in \{0, 1, \dots, 255\}$. Further attributes are not used.

From these sets of attribute values we can form 5-tuples or rather symbols $s \in \Sigma_{SH}^{\#}$ (Eq. 4.56) whose elements shall be denoted as follows:

$$\forall s \in \Sigma_{SH}^{\#}:$$

$$s := \begin{pmatrix} s.position, \\ s.size, \\ s.orientation, \\ s.shape, \\ s.color \end{pmatrix};\ \begin{matrix} s.position \in A_{position}, \\ s.size \in A_{size}, \\ s.orientation \in A_{orientation}, \\ s.shape \in A_{shape}, \\ s.color \in A_{color} \end{matrix} \tag{4.58}$$

Now that both is known, symbol properties as well as their value ranges, we can specify operations on symbols. This means, now we can define R_{change_symbol}-relations, that describe how attributes from Eq. 4.58 may change.

This requires, in a first step, that we refine $R_{change\_symbol}\langle\Sigma_{SH}, X_{SH}, K, C\rangle$ from Eq. 4.38 to a fully specified $R_{change\_symbol}\langle\Sigma_{SH}^{\#}, \emptyset, \mathbb{N}_0, \mathit{String}\rangle$. $\Sigma_{SH} = \Sigma_{SH}^{\#}$ and $X_{SH} = \emptyset$ or rather $X_{SH} = X_{SH}^{\#}$ are integral components of $L_{SH}^{\#}$ and were already defined in Eq. 4.56.

With $K = \mathbb{N}_0$ we restrict the set of valid object keys $\in K$ to integers ≥ 0 and thus set the type of $u.key$ to "unsigned int", so to speak. Theoretically, one could use any set of uniquely identifiable objects as key type K.

String should be a set of character strings. The underlying character encoding (e. g., ASCII, UTF-8 etc.) is of no importance to us. Thus, with $C = \mathit{String}$ we limit our model to string-based content (i. e. text). Therefore, we deal with hyper-*text* in the classical sense of the term. However, our model is not limited to that. Just like K also C may be chosen arbitrarily.

The most fundamental editing operations on objects in a 2.5-dimensional space include translation (i. e., change of position), scaling (i. e., change of size or proportions) and shifting objects to different layers.

These three categories of edit operations (or rather events) can be defined as derivatives of $R_{change\_symbol}\langle\Sigma_{SH}^{\#}, \emptyset, \mathbb{N}_0, \mathit{String}\rangle$:

$$E_{translate} = \left\{ R \;\middle|\; \begin{array}{c} R \in R_{change\_symbol}\langle\Sigma_{SH}^{\#}, \emptyset, \mathbb{N}_0, \mathit{String}\rangle, \\ \forall \left(u, u'\right) \in R : \\ u'.\mathit{symbol} = \mathit{translate}\left(u.\mathit{symbol}, \vec{v_t}\right), \\ \vec{v_t} \in (\mathbb{Z} \times \mathbb{Z}) \end{array} \right\}$$

$$E_{scale} = \left\{ R \;\middle|\; \begin{array}{c} R \in R_{change\_symbol}\langle\Sigma_{SH}^{\#}, \emptyset, \mathbb{N}_0, \mathit{String}\rangle, \\ \forall \left(u, u'\right) \in R : \\ u'.\mathit{symbol} = \mathit{scale}\left(u.\mathit{symbol}, \vec{v_s}\right), \\ \vec{v_s} \in \left(\mathbb{R}^+ \times \mathbb{R}^+\right) \end{array} \right\}$$

$$E_{shift\_layer} = \left\{ R \;\middle|\; \begin{array}{c} R \in R_{change\_symbol}\langle\Sigma_{SH}^{\#}, \emptyset, \mathbb{N}_0, \mathit{String}\rangle, \\ \forall \left(u, u'\right) \in R : \\ u'.\mathit{symbol} = \mathit{shift\_layer}\left(u.\mathit{symbol}, \Delta z\right), \\ \Delta z \in \mathbb{Z} \end{array} \right\} \tag{4.59}$$

These relation sets include three core mappings that specify how symbols may change: *translate* (Eq. 4.60), *scale* (Eq. 4.61) and *shift_layer* (Eq. 4.62). For reasons of clarity, we decided to define them separately.

translate (Eq. 4.60) is a total mapping of binary tuples of symbols $s \in \Sigma_{SH}^{\#}$ and translation vectors $(a, b) \in (\mathbb{Z} \times \mathbb{Z})$ to symbols $s' \in \Sigma_{SH}^{\#}$ with altered position coordinates $s.position.x + a$ and $s.position.y + b$. Depth-coordinate z as well as remaining attributes do not change their values.

$$\begin{gathered}
translate : \left(\Sigma_{SH}^{\#} \times (\mathbb{Z} \times \mathbb{Z})\right) \to \Sigma_{SH}^{\#}\,, \\
\left(\begin{pmatrix} (x, y, z), \\ size, \\ orientation, \\ shape, \\ color \end{pmatrix}, (a, b)\right) \mapsto \begin{pmatrix} (x + a, y + b, z), \\ size, \\ orientation, \\ shape, \\ color \end{pmatrix}
\end{gathered} \tag{4.60}$$

Scaling of objects (Eq. 4.61) is defined similarly. Also here we map pairs of symbols and vectors to geometrically modified symbols. The only difference is that scaling vectors are used instead of translation vectors and object dimensions are modified instead of positions.

$$\begin{gathered}
scale : \left(\Sigma_{SH}^{\#} \times \left(\mathbb{R}^{+} \times \mathbb{R}^{+}\right)\right) \to \Sigma_{SH}^{\#}\,, \\
\left(\begin{pmatrix} position, \\ (w, h), \\ orientation, \\ shape, \\ color \end{pmatrix}, (a, b)\right) \mapsto \begin{pmatrix} position, \\ \left(round(w \times a), round(h \times b)\right), \\ orientation, \\ shape, \\ color \end{pmatrix}, \\
round : \mathbb{R}^{+} \to \mathbb{N}^{+},\ \forall x \in \mathbb{R}^{+} : \\
round(x) = \begin{cases} 1 \text{ , if } x < 0.5 \\ \lfloor x + 0.5 \rfloor \text{ , else} \end{cases}
\end{gathered} \tag{4.61}$$

Modification of depth coordinates (i. e., layers) is defined by *shift_layer*:

$$\begin{gathered}
shift\_layer : \left(\Sigma_{SH}^{\#} \times \mathbb{Z}\right) \to \Sigma_{SH}^{\#}\,, \\
\left(\begin{pmatrix} (x, y, z), \\ size, \\ orientation, \\ shape, \\ color \end{pmatrix}, \Delta z\right) \mapsto \begin{pmatrix} \left(x, y, shift\_depth\_val(z, \Delta z)\right), \\ size, \\ orientation, \\ shape, \\ color \end{pmatrix}, \\
shift\_depth\_val : (\mathbb{N}_0 \times \mathbb{Z}) \to \mathbb{N}_0,\ \forall\, (z, \Delta z) \in (\mathbb{N}_0 \times \mathbb{Z}) : \\
shift\_depth\_val(z, \Delta z) = \begin{cases} 0 \text{ , if } (z + \Delta z) < 0 \\ z + \Delta z \text{ , else} \end{cases}
\end{gathered} \tag{4.62}$$

$E_{translate}$, E_{scale} and E_{shift_layer} are not the only editing events that can be defined on symbol properties from Eq. 4.58. In addition we can also refine $R_{change\_symbol}\langle\Sigma_{SH}^{\#}, \emptyset, \mathbb{N}_0, \mathit{String}\rangle$ to modifications of $s.\mathit{shape}$ (Eq. 4.63) and $s.\mathit{color}$ (Eq. 4.64). Attribute $s.\mathit{orientation}$, however, cannot change its value, since $|A_{orientation}| = 1$ (see Eq. 4.57).

$$E_{shape\_changed} = \left\{ R \;\middle|\; \begin{array}{c} R \in R_{change\_symbol}\langle\Sigma_{SH}^{\#}, \emptyset, \mathbb{N}_0, \mathit{String}\rangle, \\ \forall \left(u, u'\right) \in R : \\ u.\mathit{symbol.shape} \neq u'.\mathit{symbol.shape} \end{array} \right\} \tag{4.63}$$

$$E_{color\_changed} = \left\{ R \;\middle|\; \begin{array}{c} R \in R_{change\_symbol}\langle\Sigma_{SH}^{\#}, \emptyset, \mathbb{N}_0, \mathit{String}\rangle, \\ \forall \left(u, u'\right) \in R : \\ u.\mathit{symbol.color} \neq u'.\mathit{symbol.color} \end{array} \right\} \tag{4.64}$$

All these edit events shall be combined under the common label $E_{symbol\_changed}$:

$$\begin{aligned} E_{symbol\_changed} = \quad & E_{translate} \cup E_{scale} \cup E_{shift\_layer} \\ & \cup E_{shape\_changed} \cup E_{color\_changed} \end{aligned} \tag{4.65}$$

We use $E_{symbol} = E_{symbol\_changed}$ together with the previously defined parameters $\Sigma_{SH} = \Sigma_{SH}^{\#}$, $X_{SH} = X_{SH}^{\#} = \emptyset$, $K = \mathbb{N}_0$ and $C = \mathit{String}$ as default-settings or rather as default-configuration of $A_W\langle\Sigma_{SH}, X_{SH}, K, C, E_{symbol}\rangle$ (Eq. 4.41), $E_W\langle\Sigma_{SH}, X_{SH}, K, C, E_{symbol}\rangle$ (Eq. 4.45) and $O_W\langle\Sigma_{SH}, X_{SH}, K, C\rangle$ (Eq. 4.51):

$$\begin{aligned} A_W &:= A_W\langle\Sigma_{SH}^{\#}, \emptyset, \mathbb{N}_0, \mathit{String}, E_{symbol\_changed}\rangle \\ E_W &:= E_W\langle\Sigma_{SH}^{\#}, \emptyset, \mathbb{N}_0, \mathit{String}, E_{symbol\_changed}\rangle \\ O_W &:= O_W\langle\Sigma_{SH}^{\#}, \emptyset, \mathbb{N}_0, \mathit{String}\rangle \end{aligned} \tag{4.66}$$

The last three types of symbol events we would like to introduce are, events for construction and destruction of information units (Eq. 4.67), events for modification of content (Eq. 4.68) and "neutral" or empty events (Eq. 4.69).

Events for creation and deletion of information units shall be denoted as E_{create} and E_{delete} and are given by:

$$\begin{aligned} E_{create} &:= R_{create}\langle\Sigma_{SH}^{\#}, \emptyset, \mathbb{N}_0, \mathit{String}\rangle \\ E_{delete} &:= R_{delete}\langle\Sigma_{SH}^{\#}, \emptyset, \mathbb{N}_0, \mathit{String}\rangle \end{aligned} \tag{4.67}$$

Change of content is defined similarly:

$$E_{content\_changed} := R_{change\_content}\langle\Sigma_{SH}^{\#}, \emptyset, \mathbb{N}_0, \mathit{String}\rangle \tag{4.68}$$

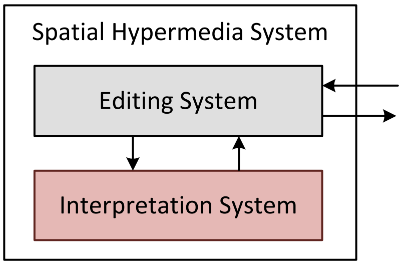


Figure 4.4: Only interpretation systems make an editing system a fully-fledged spatial hypermedia system

The last category of events to be defined in this section are empty events E_ε:

$$E_\varepsilon := \{R_\varepsilon\} \tag{4.69}$$

E_ε includes only a single element, which is the empty relation R_ε from Eq. 4.40.

Our definition of E_W from Eq. 4.66 implicitly includes these event categories:

$$E_W = E_{create} \cup E_{delete} \cup E_{symbol\_changed} \cup E_{content\_changed} \cup E_\varepsilon \tag{4.70}$$

This will play an important role in Sect. 4.3.

4.2 Interpretation System

The second integral component of spatial hypermedia systems, besides the previously described editing systems, are *interpretation systems* (see Fig. 4.4).

Editing systems serve the *creation* of spatial hypertext, whereas interpretation systems *analyze* spatial hypertext to infer visual structure (for this see our considerations on "spatial parsing" in Sect. 1.3). Without that capability of recognizing structural meaning such systems would be nothing more than just diagramming applications without significant intelligence. Only interpretation systems turn an editing system into what we regard as a fully-fledged spatial hypermedia system.

Interpretation systems transform ingoing sequences of $\underline{e}$vents e_k (Sect. 4.2.1) and info unit $\underline{D}$ata D_k (Sect. 4.2.2) into outgoing sequences of $\underline{I}$nterpretations I_k (Sect. 4.2.3). Fig. 4.5 illustrates that as a block diagram.

Note, however, that events e_k are not to be confused with our definitions from Eq. 4.66 or Eq. 4.45.

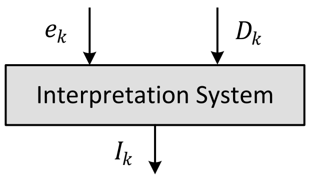


Figure 4.5: Interpretation systems transform ingoing sequences of events e_k and info unit data D_k into outgoing sequences of interpretations I_k

4.2.1 Events

Provided that Ω is the universal set of *all* theoretically possible information units (see Sect. 2.2), we define the basic set of ingoing Events $E\langle\Omega\rangle$ as:

$$E\langle\Omega\rangle = \left(\left(2^{\Omega} \setminus \emptyset\right) \times \left\{ \begin{matrix} CREATE, \\ MODIFY\_SPATIAL, \\ MODIFY\_VISUAL, \\ MODIFY\_CONTENT, \\ DELETE \end{matrix} \right\} \right) \cup \left\{ \left(\emptyset, \varepsilon\right) \right\} \tag{4.71}$$

Each of these events $e \in E\langle\Omega\rangle$ is a binary tuple $(e.objects, e.operation)$:

$$\begin{aligned} &\forall e \in E\langle\Omega\rangle : \\ &\qquad e := \begin{pmatrix} e.objects, \\ e.operation \end{pmatrix} \end{aligned} \tag{4.72}$$

Here, $e.objects$ represents a set of information units $\subseteq \Omega$ or rather their ids. This set identifies all objects, which are affected by the specified event e. The second tuple element $e.operation$ is a discrete operation flag which serves event classification. Thus, elements of $E\langle\Omega\rangle$ include both, a set of objects for which an event has occurred and an identifier of the event or operation type.

This is best explained with some examples: Suppose we define $\Omega = \{0, 1, 2\}$. Thus, the basic set of information units comprises only three possible objects. For the sake of simplicity we numbered them consecutively starting with zero. Now we shall be able, following our definition from Eq. 4.71, to express the joint creation of objects $0, 1, 2$ with $(\{0, 1, 2\}, CREATE)$, a possibly subsequent spatial modification of 1 and 2 (e. g., via translation, scaling etc.) could be described with $(\{1, 2\}, MODIFY\_SPATIAL)$ and a finishing deletion of object number 0 could be represented by $(\{0\}, DELETE)$. The binary tuple $(\emptyset, \varepsilon)$, which is also included in Eq. 4.71, is a special type of empty event.

Apparently, our definition of $E\langle\Omega\rangle$ is based on the assumption, that information units in a workspace can be (a) created; (b) modified (either spatially, visually or content-related) and (c) they can be deleted.

4.2.2 InfoUnitData

In addition to events $e_k \in E\langle\Omega\rangle$ (Sect. 4.2.1) interpretation systems accept a second input, which is info unit $\underline{\text{D}}$ata D_k. Such data packages D_k contain information, which is primarily used for structural analysis, that is, for (spatial) parsing. Here we assume, that this includes geometrical properties of information units (such as bounds, shape etc.) as well as information regarding color and content. Just like events $e_k \in E\langle\Omega\rangle$ from Sect. 4.2.1, also info unit data D_k in this section are not to be confused with our previous definitions from Eq. 4.66 or with $W.info\_units$ from Eq. 4.55.

For a precise mathematical definition of these data packages we need several auxiliary structures and operations. This includes, among other things, a definition of geometrical bounds:

Our definition of $Bounds\langle k\rangle$ builds on k-dops and thus generalizes the conventional bounding box:

$$Bounds\langle k\rangle = \left\{ \left(S_0, S_1, \ldots, S_{\left(\frac{k}{2}-1\right)} \right) \;\middle|\; \begin{matrix} S_i \in Slab, \\ 0 \le i \le \frac{k}{2} - 1, \\ k \in \{8, 16, 32, \ldots\} \end{matrix} \right\} \tag{4.73}$$

Each $b \in Bounds\langle k\rangle$ is the Boolean intersection of extents along $k/2$ directions and hence of $k/2$ bounding "Slabs" (Eq. 4.74). k is limited to $\{8, 16, 32, \ldots\}$.

$$Slab = \left\{ (\vec{n}, d_{min}, d_{max}) \;\middle|\; \begin{matrix} \vec{n} \in (\mathbb{R} \times \mathbb{R}) \wedge |\vec{n}| = 1, \\ d_{min}, d_{max} \in \mathbb{R} \wedge d_{max} > d_{min} \end{matrix} \right\} \tag{4.74}$$

Slabs are triples of normal vectors $\vec{n}$ and signed distances from the origin $d_{min}, d_{max} \in \mathbb{R}$ and describe the extent along a certain direction. This is also why $d_{max} > d_{min}$.

References to elements of such $k/2$-tuples of triples $(\vec{n}, d_{min}, d_{max})$ may become complex and therefore difficult to read. In order to avoid that, let us do the following notational simplification:

$$\forall b \in Bounds\langle k\rangle :$$

$$b := \begin{pmatrix} \big(b.normal\,[0], b.min\,[0], b.max\,[0]\big)\,, \\ \big(b.normal\,[1], b.min\,[1], b.max\,[1]\big)\,, \\ \ldots, \\ \left(b.normal\left[\frac{k}{2} - 1\right], b.min\left[\frac{k}{2} - 1\right], b.max\left[\frac{k}{2} - 1\right]\right) \end{pmatrix} \tag{4.75}$$

With this "array-like" notation we can reference single attributes of bounding volumes $b \in Bounds\langle k\rangle$ via $b.normal\,[i]$, $b.min\,[i]$ and $b.max\,[i]$ (for $0 \le i < k/2$).

For the sake of completeness, it should be pointed out that bounds $\in$ *Bounds*$\langle k\rangle$ share their normal vectors:

$$\begin{aligned} &\forall b, b' \in \mathit{Bounds}\langle k\rangle : \\ &\qquad b.\mathit{normal}[i] = b'.\mathit{normal}[i],\ i \in \left\{0, \ldots, \frac{k}{2} - 1\right\} \end{aligned} \tag{4.76}$$

It is quite obvious, that different bounds with different normal vectors would make no sense at all. Nevertheless, in order to keep our definitions formally correct we must exclude that explicitly.

This also applies to a series of auxiliary functions on which we will build on with our parsing algorithms. This includes functions for accessing horizontal, vertical and diagonal bounds of $b \in \mathit{Bounds}\langle k\rangle$. These access operations shall be defined for any $k \in \{8, 16, 32, \ldots\}$:

Assuming that the x-coordinate of $b.\mathit{normal}[0]$ is always 0 and that normal vectors $b.\mathit{normal}[i]$ are defined counter-clockwise for $0 \leq i < k/2$, we can define the following shortcuts:

$$\begin{aligned} \mathit{min}_h &: \mathit{Bounds}\langle k\rangle \to \mathbb{R},\ \forall b \in \mathit{Bounds}\langle k\rangle : \ \mathit{min}_h(b) = b.\mathit{min}\left[\frac{k}{4}\right] \\ \mathit{max}_h &: \mathit{Bounds}\langle k\rangle \to \mathbb{R},\ \forall b \in \mathit{Bounds}\langle k\rangle : \ \mathit{max}_h(b) = b.\mathit{max}\left[\frac{k}{4}\right] \end{aligned} \tag{4.77}$$

$$\begin{aligned} \mathit{min}_v &: \mathit{Bounds}\langle k\rangle \to \mathbb{R},\ \forall b \in \mathit{Bounds}\langle k\rangle : \ \mathit{min}_v(b) = b.\mathit{min}[0] \\ \mathit{max}_v &: \mathit{Bounds}\langle k\rangle \to \mathbb{R},\ \forall b \in \mathit{Bounds}\langle k\rangle : \ \mathit{max}_v(b) = b.\mathit{max}[0] \end{aligned} \tag{4.78}$$

$$\begin{aligned} \mathit{min}_{d0} &: \mathit{Bounds}\langle k\rangle \to \mathbb{R},\ \forall b \in \mathit{Bounds}\langle k\rangle : \ \mathit{min}_{d0}(b) = b.\mathit{min}\left[\frac{k}{8}\right] \\ \mathit{max}_{d0} &: \mathit{Bounds}\langle k\rangle \to \mathbb{R},\ \forall b \in \mathit{Bounds}\langle k\rangle : \ \mathit{max}_{d0}(b) = b.\mathit{max}\left[\frac{k}{8}\right] \end{aligned} \tag{4.79}$$

$$\begin{aligned} \mathit{min}_{d1} &: \mathit{Bounds}\langle k\rangle \to \mathbb{R},\ \forall b \in \mathit{Bounds}\langle k\rangle : \ \mathit{min}_{d1}(b) = b.\mathit{min}\left[\frac{3k}{8}\right] \\ \mathit{max}_{d1} &: \mathit{Bounds}\langle k\rangle \to \mathbb{R},\ \forall b \in \mathit{Bounds}\langle k\rangle : \ \mathit{max}_{d1}(b) = b.\mathit{max}\left[\frac{3k}{8}\right] \end{aligned} \tag{4.80}$$

Here, suffix h (Eq. 4.77) stands for “horizontal”, v (Eq. 4.78) for “vertical” and $d0$,$d1$ (Eq. 4.79, Eq. 4.80) for “diagonal”. We distinguish between diagonals from top-left to bottom-right ($d0$) and from top-right to bottom-left ($d1$). This also explains why in Eq. 4.73 k was limited to $\{8, 16, 32, \ldots\}$. Eq. 4.77–Eq. 4.80

require that all four bounding axes are defined, $h, v, d0$ as well as $d1$. This only holds true if $k \geq 8$.

We build on these shortcuts with several geometrical functions that will play an important role for our later definitions of parsing algorithms. Here, it is to be assumed that the basic semantics behind terms such as "union", "intersects", "contains" etc. are known. That is, it should be clear what it means to merge two bounding volumes, to check them for intersection, containment etc. Thus, we are not going to repeat the foundations of bounding volume geometry. Nevertheless, for consistency reasons, we must define at least those functions that will be used later on in our algorithms.

Width and height of bounding volumes are the easiest to define. For this we simply calculate the difference between horizontal and vertical bounds from Eq. 4.77 and Eq. 4.78:

$$\begin{aligned} \mathit{width}: \mathit{Bounds}\langle k\rangle \to \mathbb{R}^+,\ \forall b \in \mathit{Bounds}\langle k\rangle:\ \mathit{width}(b) = \mathit{max}_h(b) - \mathit{min}_h(b) \\ \mathit{height}: \mathit{Bounds}\langle k\rangle \to \mathbb{R}^+,\ \forall b \in \mathit{Bounds}\langle k\rangle:\ \mathit{height}(b) = \mathit{max}_v(b) - \mathit{min}_v(b) \end{aligned} \tag{4.81}$$

Scaling and union are slightly more complex:

$$\begin{gathered} \mathit{scale}: \big(\mathit{Bounds}\langle k\rangle \times \mathbb{R}\big) \rightharpoonup \mathit{Bounds}\langle k\rangle,\ (b, \mathit{offset}) \mapsto b', \\ \begin{pmatrix} \big(b'.\mathit{min}[i] = b.\mathit{min}[i] - \mathit{offset}\big) \\ \wedge \\ \big(b'.\mathit{max}[i] = b.\mathit{max}[i] + \mathit{offset}\big) \end{pmatrix},\ i \in \left\{0, \ldots, \frac{k}{2} - 1\right\} \end{gathered} \tag{4.82}$$

What is special about our definition of *scale* is, that no scaling factor is used. Instead dimensions of $b \in \mathit{Bounds}\langle k\rangle$ are increased or decreased by a fixed $\mathit{offset} \in \mathbb{R}$. Thus we use the term "scaling" differently than usual.

With our *union*-operation, however, we do not deviate from common semantics. Therefore it is defined as follows:

$$\begin{gathered} \mathit{union}: \big(\mathit{Bounds}\langle k\rangle \times \mathit{Bounds}\langle k\rangle\big) \to \mathit{Bounds}\langle k\rangle,\ (b_0, b_1) \mapsto b', \\ \begin{pmatrix} \Big(b'.\mathit{min}[i] = \mathit{min}\big(b_0.\mathit{min}[i], b_1.\mathit{min}[i]\big)\Big) \\ \wedge \\ \Big(b'.\mathit{max}[i] = \mathit{max}\big(b_0.\mathit{max}[i], b_1.\mathit{max}[i]\big)\Big) \end{pmatrix},\ i \in \left\{0, \ldots, \frac{k}{2} - 1\right\} \end{gathered} \tag{4.83}$$

The remaining five auxiliary functions $\mathit{delta}_{\mathit{max}}$, $\mathit{delta}_{\mathit{min}}$, *contains*, *intersects*, and *centroid* require descriptions in form of pseudo code.

Alg. 1 and Alg. 2 define maximal and minimal extent of bounding volumes $b \in \mathit{Bounds}\langle k\rangle$. delta_{max} maximizes the difference $(b.\mathit{max}[i] - b.\mathit{min}[i])$, for $0 \leq i < k/2$, whereas delta_{min} calculates the minimal $(b.\mathit{max}[i] - b.\mathit{min}[i])$.

Alg. 1 $\mathit{delta}_{max} : \mathit{Bounds}\langle k\rangle \to \mathbb{R}^+,\ b \mapsto r,$

```
1: $r \leftarrow \big(b.\mathit{max}[0] - b.\mathit{min}[0]\big)$
2: **for** $i = 1$ **to** $\left(\frac{k}{2} - 1\right)$ **do**
3:     **if** $\big(b.\mathit{max}[i] - b.\mathit{min}[i]\big) > r$ **then**
4:         $r \leftarrow \big(b.\mathit{max}[i] - b.\mathit{min}[i]\big)$
5:     **end if**
6: **end for**
7: **return** $r$
```

By substituting $((b.\mathit{max}[i] - b.\mathit{min}[i]) > r)$ for $((b.\mathit{max}[i] - b.\mathit{min}[i]) < r)$ this algorithm for determining the maximal extent of bounding volumes b changes into a function for calculating their minimal extent delta_{min}:

Alg. 2 $\mathit{delta}_{min} : \mathit{Bounds}\langle k\rangle \to \mathbb{R}^+,\ b \mapsto r,$

```
1: $r \leftarrow \big(b.\mathit{max}[0] - b.\mathit{min}[0]\big)$
2: **for** $i = 1$ **to** $\left(\frac{k}{2} - 1\right)$ **do**
3:     **if** $\big(b.\mathit{max}[i] - b.\mathit{min}[i]\big) < r$ **then**
4:         $r \leftarrow \big(b.\mathit{max}[i] - b.\mathit{min}[i]\big)$
5:     **end if**
6: **end for**
7: **return** $r$
```

Whether a given bounding volume $b_1 \in \mathit{Bounds}\langle k\rangle$ is contained by another bounding volume $b_0 \in \mathit{Bounds}\langle k\rangle$ is determined by Alg. 3. For this we define possible return values as elements of $\{\mathit{TRUE}, \mathit{FALSE}\}$.

Alg. 3 $\mathit{contains} :$ $\big(\mathit{Bounds}\langle k\rangle \times \mathit{Bounds}\langle k\rangle\big) \to \{\mathit{TRUE}, \mathit{FALSE}\},\ (b_0, b_1) \mapsto r,$

```
1: $r \leftarrow \mathit{TRUE}$
2: **for** $i = 0$ **to** $\left(\frac{k}{2} - 1\right)$ **do**
3:     **if** $\big(b_1.\mathit{min}[i] < b_0.\mathit{min}[i]\big) \vee \big(b_1.\mathit{max}[i] > b_0.\mathit{max}[i]\big)$ **then**
4:         **return** $\mathit{FALSE}$
5:     **end if**
6: **end for**
7: **return** $r$
```

When we substitute $(b_1.min[i] < b_0.min[i])$ for $(b_1.min[i] > b_0.max[i])$ and replace $(b_1.max[i] > b_0.max[i])$ by $(b_0.min[i] > b_1.max[i])$, then we can easily transform the *contains*-algorithm from Alg. 3 into an intersection test for bounding volumes:

Alg. 4 $intersects : \big(Bounds\langle k\rangle \times Bounds\langle k\rangle\big) \to \{TRUE, FALSE\},\ (b_0, b_1) \mapsto r,$

```
1: $r \leftarrow TRUE$
2: for $i = 0$ to $\left(\frac{k}{2} - 1\right)$ do
3:     if $\big(b_1.min[i] > b_0.max[i]\big) \vee \big(b_0.min[i] > b_1.max[i]\big)$ then
4:         return $FALSE$
5:     end if
6: end for
7: return $r$
```

The last auxiliary function we want to define, serves the purpose of calculating a bounding volume's centre point. We denote this function as *centroid* and define it with the following algorithm:

Alg. 5 $centroid : Bounds\langle k\rangle \to (\mathbb{R} \times \mathbb{R}),\ b \mapsto \begin{pmatrix} x \\ y \end{pmatrix},$

```
1: $x \leftarrow 0.0$
2: $y \leftarrow 0.0$
3: $P \leftarrow corner\_points(b)$
4: for all $\begin{pmatrix} a \\ b \end{pmatrix} \in P$ do
5:     $x \leftarrow x + a$
6:     $y \leftarrow y + b$
7: end for
8: return $\begin{pmatrix} x/|P| \\ y/|P| \end{pmatrix}$
```

Here we assume that *corner_points*(b), which is used in Alg. 5 in line 3, is defined for any $b \in Bounds\langle k\rangle$. The function value of *corner_points*(b) should be a set of real-valued vectors P, where each $\vec{p} \in P$ marks an intersection of two adjacent boundary lines of b (i. e., a corner point of b).

In addition to $Bounds\langle k\rangle$ from Eq. 4.73 and the previously defined auxiliary functions from Eq. 4.77–Eq. 4.83 and Alg. 1–Alg. 5, our definition of information unit data D requires *four* specific object types. These types are labeled as *Layer*, *Shape*, *Color*, *String* and are defined as can be seen in Eq. 4.84.

$$\begin{aligned}
&\mathit{Layer} = \mathbb{N}_0 \\
&\mathit{Shape} = \{\mathit{RECTANGLE}, \mathit{ELLIPSE}\} \\
&\mathit{Color} = \big\{(r,g,b)\big| r,b,g \in \{0,1,\ldots,255\}\big\} \\
&\mathit{String} : \text{set of character strings}
\end{aligned} \tag{4.84}$$

Here, the first three definitions of *Layer*, *Shape* and *Color* build on Eq. 4.57. *Layer* corresponds to depth coordinates $z \in \mathbb{N}_0$ from $A_{position}$, *Shape* comprises *RECTANGLE* and *ELLIPSE* from A_{shape}, and *Color* is, just like A_{color}, defined as a set of (r,g,b)-triples. *String*, on the other hand, represents a set of character strings. Just like in Sect. 4.1.4 also here the exact character encoding shall be of no importance. Although, in this special case, there is no substantial difference between these four sets of objects and our previous definitions from Sect. 4.1.4, they still should be understood as independent object types. At least we will treat them as such.

With all these definitions at hand we can finally determine info unit data D as:

$$D\langle\Omega,k\rangle = \left\{ D := (V,p) \;\middle|\; \begin{matrix} V \in 2^{\Omega}, \\ p : V \rightarrow \mathit{InfoUnitData}\langle k\rangle \end{matrix} \right\} \tag{4.85}$$

According to this definition, each data package $D \in D\langle\Omega,k\rangle$ comprises two components: (1) a set of information units V and (2) a function p which assigns to each unit $\in V$ an element out of $\mathit{InfoUnitData}\langle k\rangle$.

$\mathit{InfoUnitData}\langle k\rangle$ is a placeholder for the cartesian product of $\mathit{Bounds}\langle k\rangle$ (Eq. 4.73) and our previously defined object types *Layer*, *Shape*, *Color* and *String* (Eq. 4.84):

$$\mathit{InfoUnitData}\langle k\rangle := \big(\mathit{Bounds}\langle k\rangle \times \mathit{Layer} \times \mathit{Shape} \times \mathit{Color} \times \mathit{String}\big) \tag{4.86}$$

Thus each $(V,p) \in D\langle\Omega,k\rangle$ assigns to a selection of information units V five-tuples $\in \mathit{InfoUnitData}\langle k\rangle$. Attributes of such tuples are denoted as:

$$\forall d \in \mathit{InfoUnitData}\langle k\rangle :$$

$$d := \begin{pmatrix} d.\mathit{bounds}, \\ d.\mathit{layer}, \\ d.\mathit{shape}, \\ d.\mathit{color}, \\ d.\mathit{text} \end{pmatrix} \tag{4.87}$$

4.2.3 Interpretations

As we know already, interpretation systems transform ingoing sequences of events $e \in E\langle\Omega\rangle$ (Sect. 4.2.1) and info unit data $D \in D\langle\Omega, k\rangle$ (Sect. 4.2.2) into outgoing sequences of interpretations $I \in I\langle\Omega, P\rangle$.

Based on our considerations from Sect. 2.2 we define $I\langle\Omega, P\rangle$ as follows:

$$I\langle\Omega, P\rangle = \left\{ I := (U, A, w) \;\middle|\; \begin{array}{c} U \in 2^{\Omega}, \\ A = \binom{U}{2}, \\ w : A \to P \end{array} \right\} \tag{4.88}$$

The included "empty" interpretation I_ε can be described as:

$$I_\varepsilon := \big(\emptyset, \emptyset, w_\varepsilon\big) \in I\langle\Omega, P\rangle,\ \mathit{Def}(w_\varepsilon) = \emptyset \tag{4.89}$$

Thus, we use I_ε as a symbolic placeholer for a triple $(U, A, w) \in I\langle\Omega, P\rangle$, with $U = \emptyset$, the resulting empty set of associations $A = \emptyset$ and an empty mapping $w = w_\varepsilon = \emptyset$ (i. e., $\mathit{Def}(w_\varepsilon) = \emptyset$).

In the following we will work with a partial refinement of $I\langle\Omega, P\rangle$:

$$I\langle\Omega\rangle := I\langle\Omega, \{\varepsilon, 0.0, \ldots, 1.0\}\rangle \tag{4.90}$$

Here we set P equal to $\{\varepsilon, 0.0, \ldots, 1.0\}$.

According to Eq. 2.18 parameter set P shall meet the following criteria:

$$\begin{aligned} P = (P_0 \times P_1 \times \ldots \times P_{m-1}) \cup P_\varepsilon\ &,\ m \geq 1 \\ P_j \neq \emptyset\,(0 \leq j \leq m-1)& \\ P_\varepsilon \neq \emptyset,\ P_\varepsilon \subset P& \end{aligned}$$

For $m = 1$, $P_0 = \{\, p \in \mathbb{R} \mid 0.0 < p \leq 1.0 \,\}$, $P_\varepsilon = \{\varepsilon, 0.0\}$ and thus for $P = P_0 \cup P_\varepsilon = \{\varepsilon, 0.0, \ldots, 1.0\}$ this is satisfied.

Technically, such interpretations $(U, A, w) \in I\langle\Omega\rangle$ are complete, undirected and weighted graphs, which assign to each pair of information units $\{u, u'\} \in A$ a weight $\in \{\varepsilon, 0.0, \ldots, 1.0\}$. Such weights $w(\{u, u'\})$ shall describe the *strength* of a relation between u and u' (i. e., the higher the value the stronger the relationship). As an example, a weight $w(\{u, u'\}) = 0.0$ would mean that there is *no* immediate relationship between u and u'. A weight of ε, however, would indicate that it is unknown whether u and u' are associated or not; there might be an association, but the interpretation system cannot make a clear decision. We could see already some simple examples for such interpretations in Chapter. 3 in Fig. 3.2, Fig. 3.3, Fig. 3.4, or Fig. 3.5 respectively.

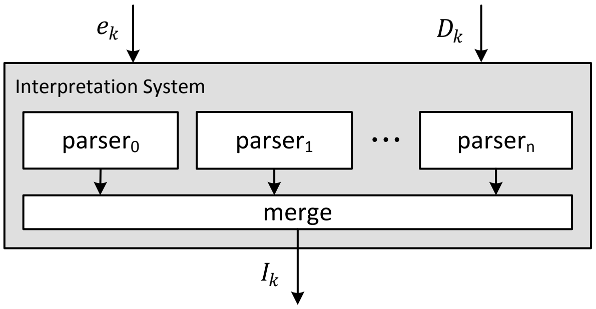


Figure 4.6: Interpretation systems forward incoming edit steps (e_k, D_k) to internal parsing components. Parse results are then merged together and are finally delivered as output I_k.

4.2.4 Merging

Interpretation systems, as we define them, transform events $e \in E\langle\Omega\rangle$ (Eq. 4.71) and info unit data $D \in D\langle\Omega, k\rangle$ (Eq. 4.85) into interpretations $I \in I\langle\Omega\rangle$ (Eq. 4.90). One could also say, that interpretation systems are dynamic systems which transform ingoing sequences of "edit steps" …

$$\left(\begin{pmatrix} e_0 \\ D_0 \end{pmatrix}, \begin{pmatrix} e_1 \\ D_1 \end{pmatrix}, \ldots, \begin{pmatrix} e_{k_e} \\ D_{k_e} \end{pmatrix}\right), \ \begin{pmatrix} e_i \\ D_i \end{pmatrix} \in \begin{pmatrix} E\langle\Omega\rangle \\ \times D\langle\Omega, k\rangle \end{pmatrix}, \ i \in \{0, 1, \ldots, k_e\}$$

… into outgoing sequences …

$$\left(I_0, I_1, \ldots, I_{k_e}\right), \ I_i \in I\langle\Omega\rangle, \ i \in \{0, 1, \ldots, k_e\}$$

What still remains to be clarified is how such transformations work; that is, how tuples (e, D) are finally mapped to I.

The main function of an interpretation system is to perform structural analyses on spatial hypertext, that is, to retrieve implicitly encoded structure. Theoretically, such an analysis may be conducted under various aspects, such as spatial, visual, temporal, content-related or others. Therefore, with our definition of interpretation systems we generalize the concept of conventional spatial parsing from Sect. 1.3. Fig. 4.6 illustrates that: Incoming edit steps (e_k, D_k) are forwarded to internal parsing components. These components (or sub-systems) are specialized in analysing certain aspects of spatial hypertext and hence provide different structural interpretations as parse result. Parse results are then merged together and are finally delivered as output I_k.

The rationale behind this is that when mixing spatial, visual etc. interpretations in the right proportions then we can filter out structures that were originally intended by hypertext authors. For this see also our considerations from Sect. 2.1 (page 31).

Before we can describe the aforementioned merge algorithm in detail, we need to define a specific auxiliary data structure, which is $\mathit{List}\langle Q\rangle$:

$$\mathit{List}\langle Q\rangle = \left\{ (q_0, q_1, \ldots, q_n) \;\middle|\; \begin{matrix} q_i \in Q, \\ i \in \{0, 1, \ldots, n\}, \\ n \in \mathbb{N}_0 \end{matrix} \right\} \cup \{\mathit{list}_\varepsilon\}, \;\; \mathit{list}_\varepsilon := () \tag{4.91}$$

Lists of "type" Q, if you want to call it that, are nothing more than ordered tuples $(q_0, q_1, \ldots, q_n)$ of list elements $q_i \in Q$ (for $i \in \{0, 1, \ldots, n\}$). For empty lists or rather 0-tuples we use the symbolic placeholder $\mathit{list}_\varepsilon$. This will serve as a kind of "default constructor" for empty lists.

Although, in theory, various operations could be defined on such lists, we limit ourselves only to those which are of importance to our algorithms. This includes, among other things, functions *add*, *size* and *get*:

Function $\mathit{add}\,(\mathit{list}, q')$ serves the purpose of extending a given $\mathit{list} \in \mathit{List}\langle Q\rangle$ by a new element $q' \in Q$. We formally describe that as follows:

$$\begin{aligned} &\mathit{add} : \big(\mathit{List}\langle Q\rangle \times Q\big) \to \mathit{List}\langle Q\rangle,\ \forall\,\big(\mathit{list}, q'\big) \in \big(\mathit{List}\langle Q\rangle \times Q\big) : \\ &\qquad \mathit{add}\,\big(\mathit{list}, q'\big) = \begin{cases} \big(q_0, q_1, \ldots, q_n, q'\big) \text{ , if } \mathit{list} = (q_0, q_1, \ldots, q_n) \text{ where } n \in \mathbb{N}_0 \\ \big(q'\big) \text{ , else} \end{cases} \end{aligned} \tag{4.92}$$

$\mathit{size}\,(\mathit{list})$ determines how many elements are included in a given $\mathit{list} \in \mathit{List}\langle Q\rangle$. For this it returns the length of the underlying tuple:

$$\begin{aligned} &\mathit{size} : \mathit{List}\langle Q\rangle \to \mathbb{N}_0,\ \forall \mathit{list} \in \mathit{List}\langle Q\rangle : \\ &\qquad \mathit{size}\,(\mathit{list}) = \begin{cases} (n+1) \text{ , if } \mathit{list} = (q_0, q_1, \ldots, q_n) \text{ where } n \in \mathbb{N}_0 \\ 0 \text{ , else} \end{cases} \end{aligned} \tag{4.93}$$

$\mathit{get}\,(\mathit{list}, i)$ serves index-based read access on single list elements. If the said index i lies out of range or if *list* is empty, then $\mathit{get}\,(\mathit{list}, i)$ returns as function value the empty symbol ε. This way, function *get* remains totally defined:

$$\begin{aligned} &\mathit{get} : \big(\mathit{List}\langle Q\rangle \times \mathbb{N}_0\big) \to \big(Q \cup \{\varepsilon\}\big),\ \forall \mathit{list} \in \mathit{List}\langle Q\rangle : \\ &\qquad \mathit{get}\,(\mathit{list}, i) = \begin{cases} q_i \text{ , if } \big(\mathit{list} = (q_0, q_1, \ldots, q_n)\big) \wedge (0 \leq i \leq n) \text{ where } n \in \mathbb{N}_0 \\ \varepsilon \text{ , else} \end{cases} \end{aligned} \tag{4.94}$$

Together with the definition of $List\langle Q\rangle$ in Eq. 4.91, functions *add* (Eq. 4.92), *size* (Eq. 4.93) and *get* (Eq. 4.94) allow for statements as illustrated in Alg. 6:

Alg. 6 sample algorithm based on definitions in Eq. 4.92, Eq. 4.93, Eq. 4.94

```
1: $l_0 \leftarrow list_\varepsilon$                          $\triangleright$ create an empty list $l_0$
2: **for** $i = 0$ **to** $999$ **do**
3:     $l_0 \leftarrow add(l_0, i)$                 $\triangleright$ fill $l_0$ with integers $\in \{0, 1, \ldots, 999\}$
4: **end for**
5: $l_1 \leftarrow list_\varepsilon$                          $\triangleright$ create another empty list $l_1$
6: **for** $i = 0$ **to** $\big(size(l_0) - 1\big)$ **do**
7:     $l_1 \leftarrow add\big(l_1, get(l_0, i)\big)$       $\triangleright$ copy all 1000 list entries from $l_0$ to $l_1$
8: **end for**
```

In addition to these three functions let us also describe the following auxiliary operations: *empty*, *first* und *last*. These build on Eq. 4.93 and Eq. 4.94 and are merely used for syntactic simplification:

empty(*list*) checks, whether the given *list* includes no elements and provides the result as an element of the discrete set $\{TRUE, FALSE\}$:

$$empty : List\langle Q\rangle \to \{TRUE, FALSE\}\,,\ \forall list \in List\langle Q\rangle :$$
$$empty(list) = \begin{cases} TRUE \text{ , if } size(list) = 0 \\ FALSE \text{ , else} \end{cases} \tag{4.95}$$

first(*list*) refines function $get(list, i)$ from Eq. 4.94 to $get(list, 0)$ and thus returns the first list element as function value:

$$first : List\langle Q\rangle \to \big(Q \cup \{\varepsilon\}\big)\,,\ list \mapsto get(list, 0) \tag{4.96}$$

Similarly, *last*(*list*) can be defined as:

$$last : List\langle Q\rangle \to \big(Q \cup \{\varepsilon\}\big)\,,\ list \mapsto get\big(list, size(list) - 1\big) \tag{4.97}$$

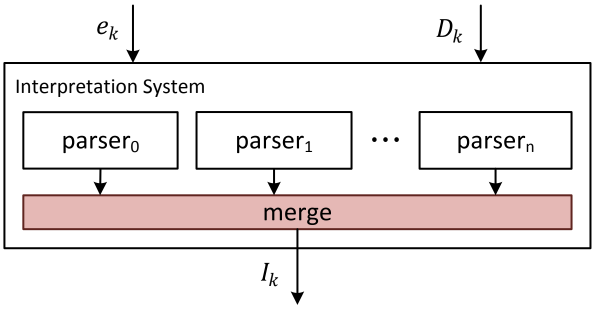


Figure 4.7: Interpretation systems delegate edit steps to internal parsers. Parse results are then mixed together using the merging algorithm described in Alg. 7.

With these definitions at hand we can now define *merge* (Fig. 4.7) as follows:

```
Alg. 7  $merge: \left(List\langle I\langle\Omega,(\mathbb{R}_0^+ \cup \{\varepsilon\})\rangle\rangle \times List\langle\mathbb{R}_0^+\rangle\right) \rightharpoonup I\langle\Omega,(\mathbb{R}_0^+ \cup \{\varepsilon\})\rangle,$
        $(interpretations, weightingFactors) \mapsto I_{merge}$

Require: $\left(\begin{array}{l} size(interpretations) = size(weightingFactors) \\ \wedge\ empty(interpretations) = FALSE \end{array}\right)$
1: $I_{merge} \leftarrow merge'\left(\begin{array}{l} first(interpretations), 0.0, \\ first(interpretations), first(weightingFactors) \end{array}\right)$
2: for $i = 1$ to $\big(size(interpretations) - 1\big)$ do
3:     $I_{merge} \leftarrow merge'\left(\begin{array}{c} I_{merge}, 1.0, \\ get(interpretations, i), get(weightingFactors, i) \end{array}\right)$
4: end for
5: return $I_{merge}$
```

Alg. 7 accepts a list of interpretations $\in I\langle\Omega,(\mathbb{R}_0^+ \cup \{\varepsilon\})\rangle$ and a list of weighting factors $\in \mathbb{R}_0^+$ as input and generates a single merged interpretation I_{merge} of type $I\langle\Omega,(\mathbb{R}_0^+ \cup \{\varepsilon\})\rangle$ as output.

Prerequisites for this algorithm are, firstly, that there must be a weighting factor for each ingoing interpretation and secondly, that the given list of interpretations may not be empty; that is, we expect at least one interpretation as input. Otherwise *merge* is undefined.

The algorithm essentially performs a cumulative, pairwise merge of interpretations from $i = 0$ to $(size(interpretations) - 1)$ using $get(weightingFactors, i)$.

The “heart” of this procedure is the pairwise merging function *merge′*, which is described in detail in the following Eq. 4.98:

$$
\begin{aligned}
&\mathit{merge}' : \begin{pmatrix} I\langle\Omega, (\mathbb{R}_0^+ \cup \{\varepsilon\})\rangle \times \mathbb{R}_0^+ \\ \times\, I\langle\Omega, (\mathbb{R}_0^+ \cup \{\varepsilon\})\rangle \times \mathbb{R}_0^+ \end{pmatrix} \rightharpoonup I\langle\Omega, (\mathbb{R}_0^+ \cup \{\varepsilon\})\rangle, \\
&\qquad \Big((U, A, w_0)\,, \alpha, (U, A, w_1)\,, \beta\Big) \mapsto (U, A, w')\,, \\
&\qquad w' : A \to \Big(\mathbb{R}_0^+ \cup \{\varepsilon\}\Big)\,,\ \forall a \in A : \\
&\qquad w'(a) = \begin{cases} \varepsilon\,, & \text{if } \Big(w_0(a) = \varepsilon\Big) \wedge \Big(w_1(a) = \varepsilon\Big) \\ \alpha \times w_0(a)\,, & \text{else if } \Big(w_0(a) \neq \varepsilon\Big) \wedge \Big(w_1(a) = \varepsilon\Big) \\ \beta \times w_1(a)\,, & \text{else if } \Big(w_0(a) = \varepsilon\Big) \wedge \Big(w_1(a) \neq \varepsilon\Big) \\ \alpha \times w_0(a) + \beta \times w_1(a)\,, & \text{else} \end{cases}
\end{aligned}
\tag{4.98}
$$

merge′$(I_0, \alpha, I_1, \beta)$ accepts two interpretations $I_0, I_1 \in I\langle\Omega, (\mathbb{R}_0^+ \cup \{\varepsilon\})\rangle$ as input, mixes them according to given weighting factors $\alpha, \beta \in \mathbb{R}_0^+$ and returns the merge result as $\in I\langle\Omega, (\mathbb{R}_0^+ \cup \{\varepsilon\})\rangle$.

This is best explained with an example. Let us assume that there are two interpretations given:

$$I_0 = (U, A, w_0)\ ;\ I_1 = (U, A, w_1)$$

Shared components U and A shall be defined as follows:

$$U = \{u_0, u_1, u_2\}\ \ ;\ \ A = \binom{U}{2} = \begin{Bmatrix} \{u_0, u_1\}\,, \\ \{u_0, u_2\}\,, \\ \{u_1, u_2\} \end{Bmatrix}$$

Weighting functions w_0, w_1 are given as:

$$w_0 = \begin{Bmatrix} (\{u_0, u_1\}\,, \varepsilon\ )\,, \\ (\{u_0, u_2\}\,, 1.0)\,, \\ (\{u_1, u_2\}\,, 1.0) \end{Bmatrix}\ \ ;\ \ w_1 = \begin{Bmatrix} (\{u_0, u_1\}\,, \varepsilon\ )\,, \\ (\{u_0, u_2\}\,, \varepsilon\ )\,, \\ (\{u_1, u_2\}\,, 0.5) \end{Bmatrix}$$

Weighting factors α, β shall be set to 0.5 each. That is, we intend to mix I_0, I_1 in a 50% : 50%-ratio.

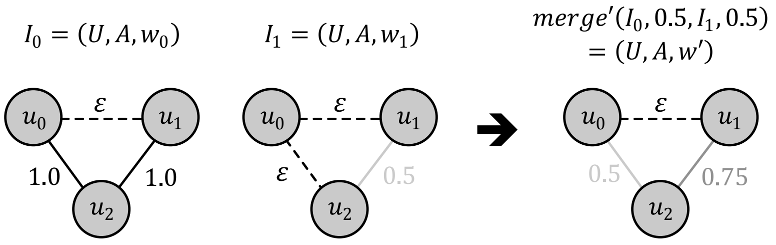


Figure 4.8: Two sample interpretations $I_0 = (U, A, w_0)$ and $I_1 = (U, A, w_1)$ mixed in a 50% : 50%-ratio using $merge'(I_0, 0.5, I_1, 0.5)$

When we apply now *merge'* from Eq. 4.98 on I_0, I_1 using $\alpha, \beta = 0.5$, then

$$merge'(I_0, \alpha, I_1, \beta) = merge'((U, A, w_0), 0.5, (U, A, w_1), 0.5)$$

results in a triple $(U, A, w') \in I\langle\Omega, (\mathbb{R}_0^+ \cup \{\varepsilon\})\rangle$, where w' is set up as follows:

$$w' = \left\{ \begin{array}{c} \left(\{u_0, u_1\}, \varepsilon\right), \\ \left(\{u_0, u_2\}, \left(\left(\alpha \times w_0\left(\{u_0, u_2\}\right)\right) = (0.5 \times 1.0) = 0.5\right)\right), \\ \left(\{u_1, u_2\}, \left(\left(\begin{array}{r} \alpha \times w_0\left(\{u_1, u_2\}\right) \\ +\beta \times w_1\left(\{u_1, u_2\}\right) \end{array}\right) = \left(\begin{array}{r} 0.5 \times 1.0 \\ +0.5 \times 0.5 \end{array}\right) = 0.75\right)\right) \end{array} \right\}$$

Thus it becomes:

$$w' = \left\{ \begin{array}{l} \left(\{u_0, u_1\}, \varepsilon \quad\right), \\ \left(\{u_0, u_2\}, 0.50\right), \\ \left(\{u_1, u_2\}, 0.75\right) \end{array} \right\}$$

This transformation of I_0 and I_1 into $merge'(I_0, 0.5, I_1, 0.5) = (U, A, w')$ is illustrated graphically in Fig. 4.8.

4.3 Linking Editing and Interpretation System

Spatial hypermedia systems are compound systems formed by editing systems (Sect. 4.1) and interpretation systems (Sect. 4.2).

Editing systems transform user interface activities (such as keyboard or mouse events etc.) into edit events $e_k \in E_W\langle\Sigma_{SH}, X_{SH}, K, C, E_{symbol}\rangle$ (Eq. 4.45) and workspaces $W_k \in O_W\langle\Sigma_{SH}, X_{SH}, K, C\rangle$ (Eq. 4.51, Eq. 4.55). Both, e_k and W_k

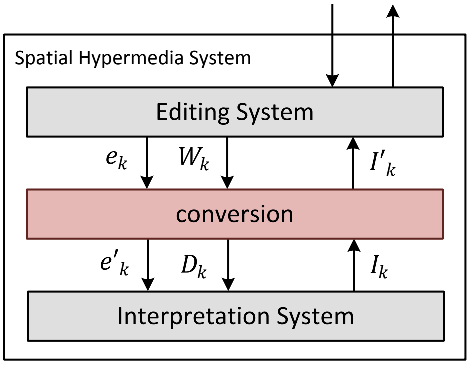


Figure 4.9: Editing system and interpretation system linked by signal conversion layer

strongly depend on the workspace model and hence on the concrete editing system. That is, parameters Σ_{SH}, X_{SH}, K, C and E_{symbol} have significant impact on the definition of an editing system's output. The best illustration of this was given in Sect. 4.1.4.

Interpretation systems, on the other hand, turn events $e_k \in E\langle\Omega\rangle$ (Eq. 4.71) and info unit data $D_k \in D\langle\Omega, k\rangle$ (Eq. 4.85) into interpretations $I_k \in I\langle\Omega\rangle$ (Eq. 4.90). There is no direct dependency on Σ_{SH} or X_{SH}, which desirably increases reusability and flexibility of our system model. In concrete terms, changes to workspaces do not automatically require adjustment of interpreters. This is especially helpful for application development.

However, since both system interfaces require different signals, it is not possible to link editing and interpretation system directly. Therefore, some kind of signal conversion layer is needed. According to Fig. 4.9 that conversion layer has to accomplish two functions:

Firstly, to transform editing system output …

$$\left(\begin{pmatrix} e_0 \\ W_0 \end{pmatrix}, \begin{pmatrix} e_1 \\ W_1 \end{pmatrix}, \ldots, \begin{pmatrix} e_{k_e} \\ W_{k_e} \end{pmatrix}\right), \quad \begin{pmatrix} e_k \\ W_k \end{pmatrix} \in \begin{pmatrix} E_W\langle\Sigma_{SH}, X_{SH}, K, C, E_{symbol}\rangle \\ \times O_W\langle\Sigma_{SH}, X_{SH}, K, C\rangle \end{pmatrix}$$

… into interpretation system input …

$$\left(\begin{pmatrix} e'_0 \\ D_0 \end{pmatrix}, \begin{pmatrix} e'_1 \\ D_1 \end{pmatrix}, \ldots, \begin{pmatrix} e'_{k_e} \\ D_{k_e} \end{pmatrix}\right), \quad \begin{pmatrix} e'_k \\ D_k \end{pmatrix} \in \begin{pmatrix} E\langle\Omega\rangle \\ \times D\langle\Omega, k_{dop}\rangle \end{pmatrix}, \; k \in \{0, 1, \ldots, k_e\}$$

… and, secondly, to transform resulting interpretations …

$$\left(I_0, I_1, \ldots, I_{k_e}\right), \;\; I_k \in I\langle\Omega\rangle, \; k \in \{0, 1, \ldots, k_e\}$$

… back into editing system input again:

$$\left(I'_0, I'_1, \ldots, I'_{k_e}\right), \; I'_k \in I\langle\Omega\rangle, \; k \in \{0, 1, \ldots, k_e\}$$

Such a two-way signal conversion strongly depends on the exact definition of input and output signals. Unlike previous definitions of editing processes, workspaces etc. there is no generic formalism that universally describes how (e_k, W_k) are mapped to (e'_k, D_k) or I_k is transformed into I'_k. Knowledge about the exact definition of Σ_{SH}, X_{SH}, K, Ω etc. is needed to accurately decribe such transformations. In other words, it becomes necessary to specify more precisely both, editing and interpretation system.

This is why our following definition of conversion functionality builds on E_W and O_W from Eq. 4.66. In concrete terms, we define ordered pairs (e_k, W_k) as $\in (E_W \times O_W)$ and hence build on our default workspace model A_W.

From Eq. 4.66 we know, that $K = \mathbb{N}_0$ (i. e., identifiers *u.key* are typed as “unsigned int”). This makes it possible to determine Ω for $E\langle\Omega\rangle$, $D\langle\Omega, k\rangle$ and $I\langle\Omega\rangle$. It should be recalled, that both, keys $\in K$ and objects $\in \Omega$ basically serve the same purpose: unique identification of information units. Thus, there is no general need for keys $\in K$ and objects $\in \Omega$ to be of different types. This is why we set $\Omega = K = \mathbb{N}_0$ and hence $E\langle\Omega\rangle$ to $E\langle\mathbb{N}_0\rangle$ and $I\langle\Omega\rangle$ to $I\langle\mathbb{N}_0\rangle$. The bounding volume parameter k, that is required by $D\langle\Omega, k\rangle$, is set to 16. $k = 16$ fulfills the condition $k \in \{8, 16, 32, \ldots\}$ from Eq. 4.73 and provides an approximation of the convex hull that is accurate enough for our purposes. Thus, tuples (e'_k, D_k) are defined as $\in (E\langle\mathbb{N}_0\rangle \times D\langle\mathbb{N}_0, 16\rangle)$. Interpretations I_k, I'_k shall be of type $I\langle\mathbb{N}_0\rangle$.

For easier handling we have separated the conversion layer into three components: (1) *convert_event*; (2) *convert_info_units* and (3) *convert_interpretation.* Here, *convert_event* serves the transformation of $e_k \in E_W$ into $e'_k \in E\langle\mathbb{N}_0\rangle$, *convert_info_units* converts $W_k \in O_W$ into $D_k \in D\langle\mathbb{N}_0, 16\rangle)$ and the third component *convert_interpretation* realizes the change of $I_k \in I\langle\mathbb{N}_0\rangle$ to $I'_k \in I\langle\mathbb{N}_0\rangle$. The blockdiagram in Fig. 4.10 illustrates that.

convert_event maps events $e \in E_W$ to pairs $(e'.objects, e'.operation) \in E\langle\mathbb{N}_0\rangle$:

$$convert\_event\colon E_W \to E\langle\mathbb{N}_0\rangle, \; e \mapsto \begin{pmatrix} e'.objects, \\ e'.operation \end{pmatrix} \tag{4.99}$$

Here, $e'.objects \subset \mathbb{N}_0$ includes identifiers for all information units that are affected by event e.

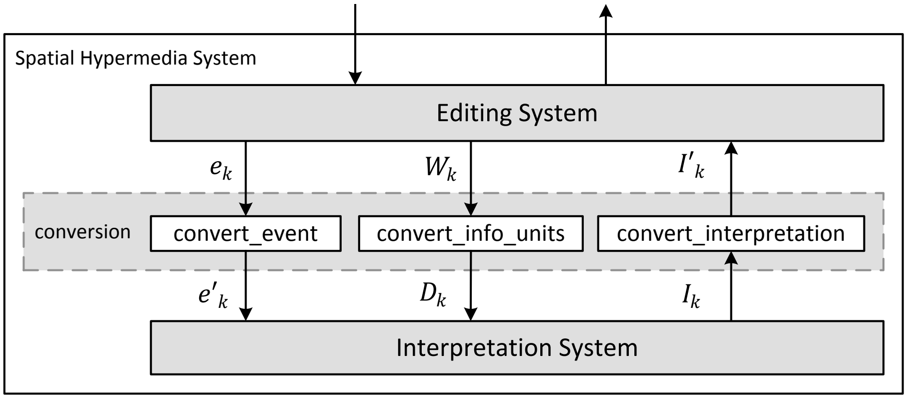


Figure 4.10: Signal conversion between editing and interpretation system is defined by three functions: *convert_event*, *convert_info_units*, and *convert_interpretation*.

It can be formed from e as follows:

$$e'.objects = \begin{cases} \emptyset & \text{, if } e \in E_\varepsilon \\ \left(\bigcup\limits_{u \in Pre_u(e)} \{u.key\} \right) & \text{, if } e \in E_{delete} \\ \left(\bigcup\limits_{u \in Post_u(e)} \{u.key\} \right) & \text{, else} \end{cases} \tag{4.100}$$

Since $K = \Omega = \mathbb{N}_0$ no special mapping is required for transforming $u.key \in \mathbb{N}_0$ into objects $\in \Omega = \mathbb{N}_0$. In other words, we can copy them directly. $e \in E_\varepsilon$ indicates an empty event (Eq. 4.69) and hence that there are no objects for which an event has occurred. In this case, $e'.objects$ can be set to $\emptyset$. If e is not an empty event but $\in E_{delete}$ (Eq. 4.67), then $e'.objects$ can be formed from keys included in $Pre_u(e)$ (Eq. 4.20). If e is neither $\in E_\varepsilon$ nor $\in E_{delete}$, then identifiers from $Post_u(e)$ are used instead (Eq. 4.21).

The second tuple element besides $e'.objects$ is the operation flag $e'.operation$. It can be formed from e as follows:

$$e'.operation = \begin{cases} CREATE & \text{, if } e \in E_{create} \\ MODIFY\_SPATIAL & \text{, if } e \in E_{symbol\_changed} \setminus E_{color\_changed} \\ MODIFY\_VISUAL & \text{, if } e \in E_{color\_changed} \\ MODIFY\_CONTENT & \text{, if } e \in E_{content\_changed} \\ DELETE & \text{, if } e \in E_{delete} \\ \varepsilon & \text{, else} \end{cases} \tag{4.101}$$

Eq. 4.101 is a conditional mapping of events $e \in E_W$ to discrete operation flags (as defined in Eq. 4.71). It therefore classifies events e according to whether they describe construction or destruction, change of spatial, visual or content-related attributes, or whether they represent any other event.

convert_info_units defines, how we transform workspaces $W \in O_W$ into adequate info unit data packages $(V, p) \in D\langle\mathbb{N}_0, 16\rangle$:

$$
\begin{aligned}
&\mathit{convert\_info\_units} : O_W \to D\langle\mathbb{N}_0, 16\rangle,\ W \mapsto (V, p)\,,\\
&\qquad V = \left(\bigcup_{u \in W.\mathit{info\_units}} \{u.\mathit{key}\} \right)\\
&\qquad p : V \to \mathit{InfoUnitData}\langle 16\rangle,\ \ p = \left(\bigcup_{u \in W.\mathit{info\_units}} \Big\{ \big(u.\mathit{key}, \mathit{value}(u)\big) \Big\} \right)
\end{aligned}
\tag{4.102}
$$

In V we collect keys for all information units included in workspace W. Function p assigns to each $u.\mathit{key} \in V$ some $\mathit{value}(u) \in \mathit{InfoUnitData}\langle 16\rangle$ (Eq. 4.86).

We define $\mathit{value}(u)$ as a mapping of tuples $u \in (\Sigma_{SH}^{\#} \times (\mathbb{N}_0 \times \mathit{String}))$, as described in Eq. 4.33, to $\mathit{InfoUnitData}\langle 16\rangle$:

$$
\begin{aligned}
&\mathit{value} : \Big(\Sigma_{SH}^{\#} \times (\mathbb{N}_0 \times \mathit{String})\Big) \to \mathit{InfoUnitData}\langle 16\rangle,\\
&\left(\begin{matrix} \mathit{symbol}, \\ \left(\begin{matrix} \mathit{key}, \\ \mathit{content} \end{matrix} \right) \end{matrix} \right) \mapsto \left(\begin{matrix} \mathit{create\_bounds} \left(\begin{matrix} \mathit{symbol.position}, \\ \mathit{symbol.size}, \\ \mathit{symbol.orientation}, \\ \mathit{symbol.shape} \end{matrix} \right), \\ \mathit{symbol.position.z}, \\ \mathit{symbol.shape}, \\ \mathit{symbol.color}, \\ \mathit{content} \end{matrix} \right)
\end{aligned}
\tag{4.103}
$$

Here, only the *bounds*-attribute (see Eq. 4.87) requires some additional calculation. This is represented by *create_bounds*:

$$
\begin{aligned}
&\mathit{create\_bounds} : \big(A_{position} \times A_{size} \times A_{orientation} \times A_{shape}\big) \to \mathit{Bounds}\langle 16\rangle,\\
&\qquad \left(\begin{matrix} \mathit{position}, \\ \mathit{size}, \\ \mathit{orientation}, \\ \mathit{shape} \end{matrix} \right) \mapsto \left(\begin{matrix} (\vec{n_0}, \mathit{min}_0, \mathit{max}_0)\,, \\ (\vec{n_1}, \mathit{min}_1, \mathit{max}_1)\,, \\ \ldots, \\ (\vec{n_7}, \mathit{min}_7, \mathit{max}_7) \end{matrix} \right)
\end{aligned}
\tag{4.104}
$$

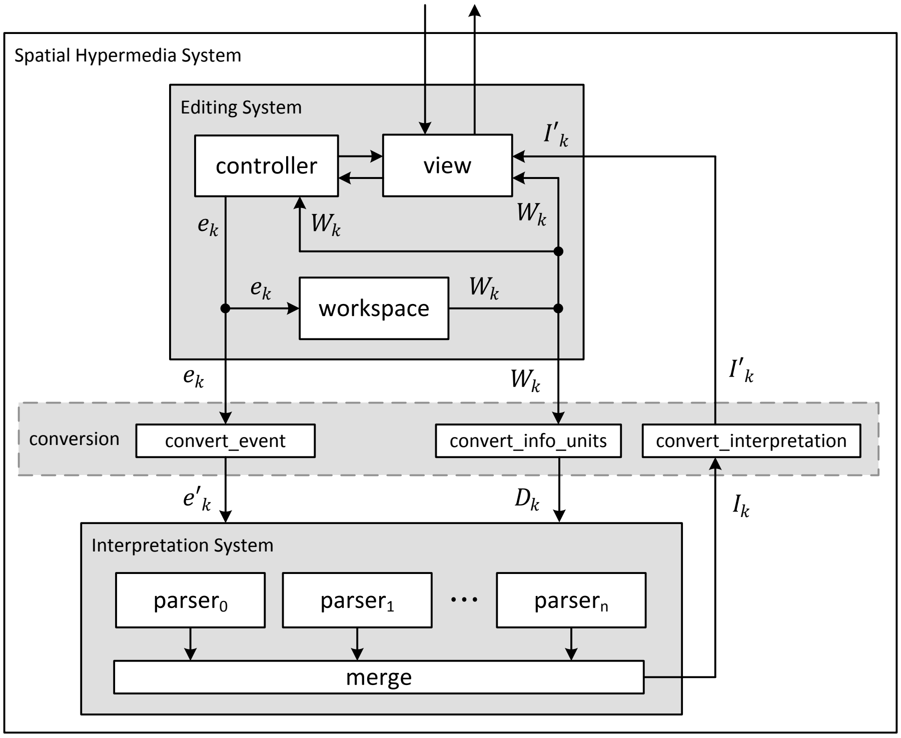


Figure 4.11: Editing system (Sect. 4.1) and interpretation system (Sect. 4.2) interconnected via signal conversion functions *convert_event* (Eq. 4.99), *convert_info_units* (Eq. 4.102), and *convert_interpretation* (Eq. 4.105).

Possible algorithms that realize this mapping of spatial symbol properties (Eq. 4.57) to 8-tuples of slabs (Eq. 4.74) shall not be defined here.

The third and last conversion function besides *convert_event* (Eq. 4.99) and *convert_info_units* (Eq. 4.102) is *convert_interpretation*:

$$\mathit{convert\_interpretation} : I\langle\mathbb{N}_0\rangle \to I\langle\mathbb{N}_0\rangle,\ I \mapsto I \qquad (4.105)$$

In our special case there is no need for modifying interpretations before they get transmitted to application level. For our prototypical spatial hypermedia system we assume, that feedback provided by interpretation systems is used for graphical display only (i. e., as input for viewing components). For this, original $I \in I\langle\mathbb{N}_0\rangle$ as generated by *merge* (see Alg. 7) is perfectly sufficient. That is why we define *convert_interpretation* as an identical mapping $I \mapsto I$.

When we merge Fig. 4.3 (editing system), Fig. 4.6 (interpretation system) and Fig. 4.10 (conversion) together, we get what is illustrated in Fig. 4.11. Following this model, one could understand spatial hypermedia applications as dynamic,

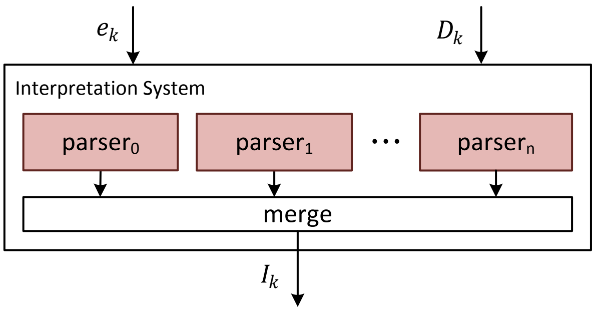


Figure 4.12: The core of every interpretation system are integrated parsing components implementing highly specialized parsing algorithms.

graphical and intelligent information systems that only "come to life" through user interaction. Activities in the visual information space are quasi the "pulse" of spatial hypermedia systems.

4.4 Parsers

Spatial hypermedia systems can be defined as composites of editing systems (Sect. 4.1) and interpretation systems (Sect. 4.2). Editing systems are mainly determined by workspace models as described in Sect. 4.1.3, whereas interpretation systems are primarily defined by parsing algorithms. In other words, the core of every editing system is an information workspace and the heart of an interpretation system are integrated parsing components. On several occasions, we have already pointed out the importance of parsers for spatial hypermedia systems. According to Chapter. 3 and Sect. 4.2, only structural analyses make a visual editor a fully-fledged spatial hypermedia system. Thus, spatial parsers are of particular relevance to our system model. However, except for our informal descriptions in Sect. 1.3 and Sect. 1.5 and apart from our general considerations from Chapter. 3 no further explanations on the functionality of spatial parsers were given so far. Although being crucial for our system design, our model (see Fig. 4.12) is still missing some explicit and formal parser definition. This section is intended to change that.

4.4.1 Generic Parser Model

As we know already from Sect. 4.2.4, interpretation systems generalize the traditional concept of spatial parsing as we discussed it in Sect. 1.3. Parsers included in interpretation systems do not label their output with pre-defined

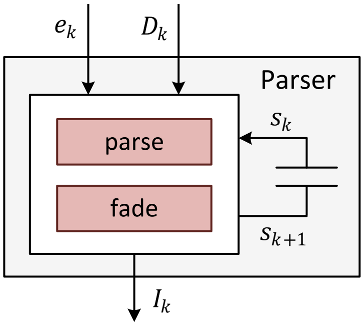


Figure 4.13: Parsers are dynamic systems determined by two components: *parse* and *fade*

semantic types (such as stack, heap, table etc.). In contrast to parsers in VIKI, VKB (Sect. 1.5.1) etc. they are not designed as configurable structure experts or independent structure recognizers. They rather analyse one and the same structure(s) from different perspectives (such as spatial, visual, temporal, or content-related). Figuratively speaking, different parsers view spatial hypertext through different "glasses" and thus realise a kind of multi-level filtering of structure. With this abstraction we expect to cover a much greater range of more general structures than possible with pattern-based approaches.

Although our parsers analyse different attributes with different heuristics, they still have some characteristics in common. This allows us to define a generic parser model.

Just like interpretation systems as a whole, also included parsers transform ingoing sequences of edit steps $(e, D) \in (E\langle\Omega\rangle \times D\langle\Omega, k\rangle)$ (see Eq. 4.71; Eq. 4.85) into outgoing sequences of interpretations $I \in I\langle\Omega\rangle$ (Eq. 4.90):

$$\begin{tikzcd}
\left(\binom{e_0}{D_0}, \binom{e_1}{D_1}, \dots, \binom{e_{k_e}}{D_{k_e}}\right) \arrow[r] & \boxed{\text{Parser}} \arrow[r] & (I_0, I_1, \dots, I_{k_e})
\end{tikzcd}$$

Therefore, both, interpretation systems and integrated parsers have more or less the same system interfaces. However, what differentiates them is their internal structure. Thus, we define parsers as dynamic systems determined by two components (Fig. 4.13): (1) *parse* and (2) *fade*.

With *parse* we specify for selected perspectives (such as spatial, visual etc.) how to infer the strength of pairwise object relations and therefore how to create structural interpretations. Thus, *parse* contains the real parsing algorithm.

The downstream function *fade* is to be understood as a sort of *post-processor*.

fade should post-process or rather *finish* interpretation results that were previously generated by *parse*. Basically, this subsequent processing of parse results is supposed to work as follows:

When *parse* cannot detect a relation between two objects, then the fading-module checks if there is still a connection known from a previous parser run. If the parser can "remember" such a relationship with a weight > 0.0, then that old weight gets multiplied by a given *fading factor* (which we will denote as α) and is treated as our current parse result. If, however, there is no such relationship, or if the product of previous weight and *fading factor* α reaches a pre-defined *fading limit* β, then we still accept the result of the current parser run. This means, then we accept that there is no perceivable association. In short, the "fading-feature" makes it possible, that associations which are not recognized anymore, do not simply get omitted. Instead, they *fade* away. Or in other words, the parser slowly "forgets" outdated associations. In this regard *fade* extends our parser model by some short-term memory. A similar aging-feature for viewport positions and motion paths in visual workspaces has been presented in [6]. This extension makes our parser model perfectly suited for the detection of destroyed structures, as we discussed them in Sect. 3.2.

We define our generic parser model using the following automaton:

$$A_P\langle\Omega, n, \alpha, \beta, \Phi, \varphi\rangle = \begin{pmatrix} S_P\langle\Omega\rangle, \\ E_P\langle\Omega, n\rangle, \\ O_P\langle\Omega\rangle, \\ T_P\langle\Omega, n, \alpha, \beta, \Phi, \varphi\rangle, \\ G_P\langle\Omega, n, \alpha, \beta, \Phi, \varphi\rangle, \\ s_{init} \end{pmatrix} \tag{4.106}$$

The generic <u>P</u>arser <u>A</u>utomaton A_P is a six-tuple, whose components are determined by six parameters: Ω, n, α, β, Φ, and φ.

Ω is the basic set of information units, as it was used already when we defined $E\langle\Omega\rangle$ (Eq. 4.71), $D\langle\Omega, k\rangle$ (Eq. 4.85) or $I\langle\Omega\rangle$ (Eq. 4.90). $n \in \{8, 16, 32, \ldots\}$ shall be the bounding volume parameter, which is, according to Eq. 4.85, expected by $D\langle\Omega, n\rangle$. Here we use n instead of k to avoid name collisions.

$\alpha, \beta \in \{0.0, \ldots, 1.0\}$ are the previously mentioned configuration parameters for the *fade*-post processor, that is, fading factor α and fading limit β. Φ is a set of event categories defining when to trigger a full reparse (for this see our definition of operation flags in Eq. 4.71).

The last parameter φ shall be a mapping $(I\langle\Omega\rangle \times E\langle\Omega\rangle \times D\langle\Omega, n\rangle) \rightarrow I\langle\Omega\rangle$ defining the core-parsing algorithm; that is, the algorithm behind the system component *parse*. This is intended to keep the parser's structure detection algorithm variable and therefore our theoretical model as flexible as possible.

These four constraints are summarized again in Eq. 4.107:

$$\begin{array}{c} n \in \{8, 16, 32, \ldots\} \\ \alpha, \beta \in \{0.0, \ldots, 1.0\} \\ \varphi : \big(I\langle\Omega\rangle \times E\langle\Omega\rangle \times D\langle\Omega, n\rangle\big) \to I\langle\Omega\rangle \end{array} \qquad \Phi \subseteq \left\{ \begin{array}{c} \mathit{CREATE}, \\ \mathit{MODIFY\_SPATIAL}, \\ \mathit{MODIFY\_VISUAL}, \\ \mathit{MODIFY\_CONTENT}, \\ \mathit{DELETE}, \\ \varepsilon \end{array} \right\} \tag{4.107}$$

Just like full interpretation systems, also included parsers A_P are to accept edit steps $\in (E\langle\Omega\rangle \times D\langle\Omega, n\rangle)$ as input and generate interpretations $\in I\langle\Omega\rangle$ as output. This is why we set $E_P\langle\Omega, n\rangle$ to $(E\langle\Omega\rangle \times D\langle\Omega, n\rangle)$ and $O_P\langle\Omega\rangle$ to $I\langle\Omega\rangle$. The set of possible states the parser automaton may pass through shall be equal to $(I\langle\Omega\rangle \times I\langle\Omega\rangle)$. In its initial state A_P includes only empty interpretations, hence s_{init} is set to $(I_\varepsilon, I_\varepsilon)$.

$$\begin{aligned} S_P\langle\Omega\rangle &= I\langle\Omega\rangle \times I\langle\Omega\rangle \\ E_P\langle\Omega, n\rangle &= E\langle\Omega\rangle \times D\langle\Omega, n\rangle \\ O_P\langle\Omega\rangle &= I\langle\Omega\rangle \\ s_{init} &= (I_\varepsilon, I_\varepsilon) \in S_P\langle\Omega\rangle \end{aligned} \tag{4.108}$$

Based on this, we can define the transition function T_P as follows:

$$\begin{aligned} T_P\langle\Omega, n, \alpha, \beta, \Phi, \varphi\rangle : {} & \big(S_P\langle\Omega\rangle \times E_P\langle\Omega, n\rangle\big) \to S_P\langle\Omega\rangle, \\ & \Big((I, J), (e, D)\Big) \mapsto \big(I', J'\big), \\ & I' = \mathit{parse}\,(I, e, D) \\ & J' = \mathit{fade}\,\big(J, I'\big) \end{aligned} \tag{4.109}$$

T_P maps pairs of states $(I, J) \in S_P\langle\Omega\rangle$ and input signals $(e, D) \in E_P\langle\Omega, n\rangle$ to subsequent states $(I', J') \in S_P\langle\Omega\rangle$. This mapping or rather this transition from (I, J) to (I', J') takes place in two steps: Firstly, *parse* transforms interpretation I into I', considering the ingoing edit step (e, D). One could also say, that *parse* switches the first half of the current state I to I'. In a second step, we map I' together with the remaining J to J'. This is accomplished by $J' = \mathit{fade}\big(J, I'\big)$. By this means T_P connects both functions *parse* and *fade* in series and thus realises a two-stage state transition from $(I, J) \in S_P\langle\Omega\rangle$ to $(I', J') \in S_P\langle\Omega\rangle$.

The included auxiliary function *parse* can be seen as a conditional call of the parsing function φ (Eq. 4.107). It thus controls when to perform a reparse and when to keep previous parse results instead:

$$\begin{aligned} \mathit{parse}:\ &\big(I\langle\Omega\rangle \times E\langle\Omega\rangle \times D\langle\Omega,n\rangle\big) \to I\langle\Omega\rangle, \\ &(I,e,D) \mapsto I' = \begin{cases} \varphi\,(I,e,D)\ , \text{ if } e.\mathit{operation} \in \Phi \\ \qquad I\ , \text{ else} \end{cases} \end{aligned} \tag{4.110}$$

A full reparse happens only if $e.\mathit{operation} \in \Phi$ (Eq. 4.107). Only when the assigned event category $e.\mathit{operation}$ identifies a given input event e as reparse event, then the ingoing triple (I,e,D) gets mapped to $\varphi\,(I,e,D)$. If, however, $e.\mathit{operation}$ is not an element of Φ, then previous interpretation I is used as function value instead; that is $\mathit{parse}\,(I,e,D) = I$.

The second function, besides *parse* (Eq. 4.110), that plays an important role for the definition of T_P (Eq. 4.109) is *fade*. *fade* maps pairs of interpretations $((U_0,A_0,w_0),(U_1,A_1,w_1))$ to triples (U_1,A_1,w'); expressed formally:

$$\mathit{fade}:\ \big(I\langle\Omega\rangle \times I\langle\Omega\rangle\big) \to I\langle\Omega\rangle,\ \Big((U_0,A_0,w_0)\,,(U_1,A_1,w_1)\Big) \mapsto \big(U_1,A_1,w'\big) \tag{4.111}$$

(U_0,A_0,w_0) shall be understood as the (temporal) predecessor of (U_1,A_1,w_1). This means, (U_0,A_0,w_0) is part of the (still) up-to-date parser state, whereas (U_1,A_1,w_1) already represents a new (internal) parse result; for details on this see $\mathit{fade}\,(J,I')$ in Eq. 4.109. U_1 and A_1 can be copied directly into the result triple. Only weighting function w' needs recalculation. It is set up as follows:

$$\begin{aligned} &w' : A_1 \to \{\varepsilon, 0.0, \ldots, 1.0\}\,,\ \forall a \in A_1 : \\ &w'(a) = \begin{cases} \mathit{fade}'\Big(w_0(a), w_1(a)\Big)\ , \text{ if } a \in A_0 \\ \qquad\qquad w_1(a)\ , \text{ else} \end{cases} \end{aligned} \tag{4.112}$$

For better legibility, we decided to split the definition of *fade* up into three parts: *fade*, which is given above, *fade′* (Eq. 4.113) and *fade″* (Eq. 4.114).

Eq. 4.112 is to be understood as follows: When a given $a \in A_1$ is also element of A_0, then both $w_0(a)$ and $w_1(a)$ and thus also $\mathit{fade}'(w_0(a), w_1(a))$ are defined. Then we can set $w'(a) = \mathit{fade}'(w_0(a), w_1(a))$. Or in other words, when associations a are included in both, current interpretation graph (U_1,A_1,w_1) as well as in its predecessor (U_0,A_0,w_0), then $w'(a)$ can be set equal to $\mathit{fade}'(w_0(a), w_1(a))$. If, however, a is element of A_1 but not of A_0, then it can be assumed that a is *new*; that is, association a must have been introduced with the last time step (e. g., by adding new information units to the workspace). In this case the new weight $w_1(a)$ is taken over into the result graph without modification, that is $w'(a) = w_1(a)$. Deleted associations are excluded from the outset by copying U_1, A_1 directly into the result triple (U_1,A_1,w'). With (U_1,A_1,w') we simply drop associations a which are $\in A_0$ but not $\in A_1$ anymore.

fade$'$ accepts pairs of weights (x_0, x_1), checks whether they allow for reasonable fading, and provides either *fade*$''(x_0, x_1)$ or x_1 as result. Here x_0 represents the (temporal) predecessor of x_1 (for this see Eq. 4.112).

$$\begin{aligned}
&\mathit{fade}' : \Big(\{\varepsilon, 0.0, \dots, 1.0\} \times \{\varepsilon, 0.0, \dots, 1.0\}\Big) \to \{\varepsilon, 0.0, \dots, 1.0\}\,, \\
&\qquad \forall x_0, x_1 \in \{\varepsilon, 0.0, \dots, 1.0\} : \\
&\qquad \mathit{fade}'(x_0, x_1) = \begin{cases} \mathit{fade}''(x_0, x_1) \text{ , if } (x_0 \neq \varepsilon) \wedge \big(x_1 \in \{\varepsilon, 0\}\big) \\ x_1 \text{ , else} \end{cases}
\end{aligned} \tag{4.113}$$

Only if fading of weights is necessary ($x_1 \in \{\varepsilon, 0\}$) and also possible ($x_0 \neq \varepsilon$), then we continue our calculation with *fade*$''(x_0, x_1)$. Otherwise, current weight x_1 is used as result; that is *fade*$'(x_0, x_1) = x_1$.

The *Fading Core-Algorithm* is defined by *fade*$''$:

fade$''$ accepts weights $x_0 \in \{0.0, \dots, 1.0\}$ and $x_1 \in \{\varepsilon, 0\}$ and maps them to values $\in \{\varepsilon, 0.0, \dots, 1.0\}$:

$$\begin{aligned}
&\mathit{fade}'' : \Big(\{0.0, \dots, 1.0\} \times \{\varepsilon, 0\}\Big) \to \{\varepsilon, 0.0, \dots, 1.0\}\,, \\
&\qquad \forall (x_0, x_1) \in \Big(\{0.0, \dots, 1.0\} \times \{\varepsilon, 0\}\Big) : \\
&\qquad \mathit{fade}''(x_0, x_1) = \begin{cases} x_0 \times \alpha \text{ , if } (x_0 \times \alpha) \geq \beta \\ x_1 \text{ , else} \end{cases}
\end{aligned} \tag{4.114}$$

When the product of old weight x_0 and *fading factor* α becomes greater or equal to *fading limit* β (Eq. 4.107), then we accept $x_0 \times \alpha$ as “faded” weighting. Otherwise x_1 remains unchanged; that is *fade*$''(x_0, x_1) = x_1$.

This is best explained with an example. Let us assume that there are two interpretations given:

$$I_0 = (U_0, A_0, w_0) \ ; \ I_1 = (U_1, A_1, w_1)$$

U_0, U_1 and A_0, A_1 shall be defined as follows:

$$U_0 = U_1 = \{u_0, u_1, u_2\} \ ; \ A_0 = A_1 = \left\{ \begin{array}{l} \{u_0, u_1\}\,, \\ \{u_0, u_2\}\,, \\ \{u_1, u_2\} \end{array} \right\}$$

Weighting functions w_0, w_1 are given as:

$$w_0 = \left\{ \begin{array}{l} \big(\{u_0, u_1\}, 1.0\big)\,, \\ \big(\{u_0, u_2\}, 1.0\big)\,, \\ \big(\{u_1, u_2\}, 0.0\big) \end{array} \right\} \ ; \ w_1 = \left\{ \begin{array}{l} \big(\{u_0, u_1\}, 1.0\big)\,, \\ \big(\{u_0, u_2\}, 0.0\big)\,, \\ \big(\{u_1, u_2\}, \varepsilon\ \big) \end{array} \right\}$$

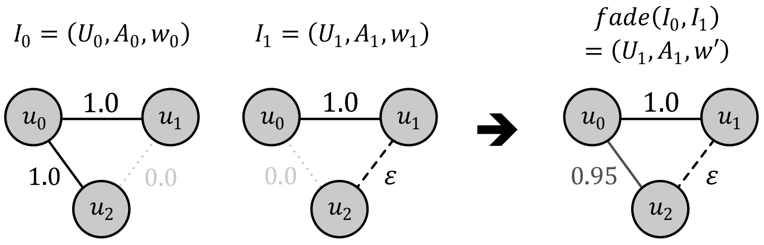


Figure 4.14: Two sample interpretations $I_0 = (U_0, A_0, w_0)$ and $I_1 = (U_1, A_1, w_1)$ processed by $\mathit{fade}(I_0, I_1)$ (Eq. 4.111). Fading factor α is set to 0.95 and fading limit β is 0.05.

Having set fading factor $\alpha = 0.95$ and fading limit $\beta = 0.05$, $\mathit{fade}(I_0, I_1)$ provides a result triple (U_1, A_1, w'), where w' is formed as follows:

$$w' = \left\{ \begin{array}{l} \left(\{u_0, u_1\}, \mathit{fade}' \begin{pmatrix} w_0(\{u_0, u_1\}), \\ w_1(\{u_0, u_1\}) \end{pmatrix} = \mathit{fade}' \begin{pmatrix} 1.0, \\ 1.0 \end{pmatrix} = 1.0 \right), \\ \left(\{u_0, u_2\}, \mathit{fade}'' \begin{pmatrix} w_0(\{u_0, u_2\}), \\ w_1(\{u_0, u_2\}) \end{pmatrix} = \mathit{fade}'' \begin{pmatrix} 1.0, \\ 0.0 \end{pmatrix} = \begin{array}{r} 1.00 \\ \times 0.95 \end{array} = 0.95 \right), \\ \left(\{u_1, u_2\}, \mathit{fade}'' \begin{pmatrix} w_0(\{u_1, u_2\}), \\ w_1(\{u_1, u_2\}) \end{pmatrix} = \mathit{fade}'' \begin{pmatrix} 0.0, \\ \varepsilon \end{pmatrix} = \varepsilon \right) \end{array} \right\}$$

Thus, w' becomes:

$$w' = \left\{ \begin{array}{l} (\{u_0, u_1\}, 1.00), \\ (\{u_0, u_2\}, 0.95), \\ (\{u_1, u_2\}, \varepsilon \quad) \end{array} \right\}$$

This is also illustrated graphically in Fig. 4.14.

Both auxiliary algorithms *parse* (Eq. 4.110) and *fade* (Eq. 4.111) are integral components of the transition function T_P (Eq. 4.109). T_P, in turn, can be found again in A_P's output function G_P:

$$\begin{aligned} G_P\langle \Omega, n, \alpha, \beta, \Phi, \varphi \rangle : \big(S_P\langle \Omega \rangle \times E_P\langle \Omega, n \rangle\big) &\to O_P\langle \Omega \rangle, \\ \Big(s, (e, D)\Big) &\mapsto J, \\ (I, J) &= T_P\langle \Omega, n, \alpha, \beta, \Phi, \varphi \rangle \Big(s, (e, D)\Big) \end{aligned} \tag{4.115}$$

Here, T_P determines for a given state s and input (e, D) the successor state to s, which is denoted as (I, J). The second tuple element of this successor state J is used as return value. From Eq. 4.109 we know, that J needs to be a function value of *fade* (Eq. 4.111). Thus G_P provides "faded" interpretations as output. Whether or how this influences parse results can be controlled via two

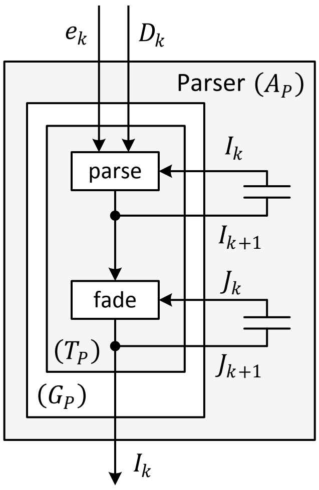


Figure 4.15: Blockdiagram of generic parser model A_P as defined in Eq. 4.106

tuning parameters α and β. As an example, $\alpha = 0.0, \beta = 1.0$ would deactivate the fading mechanism, since condition $(x_0 \times \alpha) \geq \beta$ from Eq. 4.114 cannot be satisfied for any x_0. In this case *fade* would map $((U_0, A_0, w_0), (U_1, A_1, w_1))$ to (U_1, A_1, w_1); hence G_P delivers the latest (unfaded) result provided by *parse*.

How G_P (Eq. 4.115), embedded transition function T_P (Eq. 4.109), as well as their integral components *parse* (Eq. 4.110) and *fade* (Eq. 4.111) are combined is best illustrated with a blockdiagram. Such a diagram can be found in Fig. 4.15.

We can conclusively summarize the behaviour of such dynamic parsing systems as follows:

Parsers defined by $A_P\langle\Omega, n, \alpha, \beta, \Phi, \varphi\rangle$ (Eq. 4.106) accept ingoing sequences of edit steps $V\,(0 \ldots k_e)$, where ...

$$V\,(0 \ldots k_e) = \left(v_0, v_1, \ldots, v_{k_e}\right) = \left(\begin{pmatrix} e_0 \\ D_0 \end{pmatrix}, \begin{pmatrix} e_1 \\ D_1 \end{pmatrix}, \ldots, \begin{pmatrix} e_{k_e} \\ D_{k_e} \end{pmatrix}\right),$$
$$v_k \in E_P\langle\Omega, n\rangle,\ k = 0, 1, \ldots, k_e \tag{4.116}$$

Driven by $V\,(0 \ldots k_e)$ they pass through states $S\,(0 \ldots k_e + 1)$,

$$S\,(0 \ldots k_e + 1) = \left(s_0, s_1, \ldots, s_{k_e+1}\right) = \left(\begin{pmatrix} I_0 \\ J_0 \end{pmatrix}, \begin{pmatrix} I_1 \\ J_1 \end{pmatrix}, \ldots, \begin{pmatrix} I_{k_e+1} \\ J_{k_e+1} \end{pmatrix}\right),$$
$$s_0 = s_{init},$$
$$s_{k+1} = T_P\langle\Omega, n, \alpha, \beta, \Phi, \varphi\rangle\,(s_k, v_k)\,,\ k = 0, 1, \ldots, k_e \tag{4.117}$$

... and finally they generate sequences of interpretations $I\,(0\ldots k_e)$ as output:

$$I\,(0\ldots k_e) = \left(I_0, I_1, \ldots, I_{k_e}\right),$$
$$I_k = G_P\langle\Omega, n, \alpha, \beta, \Phi, \varphi\rangle\,(s_k, v_k)\,,\; k = 0, 1, \ldots, k_e \tag{4.118}$$

4.4.2 Spatial Parser

Parsers defined by $A_P\langle\Omega, n, \alpha, \beta, \Phi, \varphi\rangle$ (Eq. 4.106) do not classify structures according to predefined patterns; that is, they do not label their output as stacks, heaps, tables or other semantic types. They rather analyze the strength of pairwise object relations and hence deliver weighted networks of objects $\in I\langle\Omega\rangle$, as defined in Eq. 4.90. Such analyses can be performed with regard to different structural aspects. This includes *spatial* relationships. We mentionated that already in Sect. 1.4 when we discussed common structure types. Parsers that are specialized in analysing spatial properties of spatial hypertext and furthermore build on our theoretical parser model from Eq. 4.106 are hereafter referred to as "spatial parsers".

Let Φ_S be a set of event categories that indicate when to perform a full spatial parse. For this see our definitions of *parse* in Eq. 4.110 and Φ in Eq. 4.107.

$$\Phi_S = \left\{\begin{matrix} \mathit{CREATE}, \\ \mathit{MODIFY\_SPATIAL}, \\ \mathit{DELETE} \end{matrix}\right\} \tag{4.119}$$

According to this definition of Φ_S, a full reparse only happens in three cases: (1) when new information units were added to a workspace and thus spatial structure might have changed; then the indicator *CREATE* is used; (2) when spatial symbol properties got modified, due to translation, scaling etc.; this is signaled by *MODIFY_SPATIAL* and (3) information units were removed, which might have destroyed spatial structure; this is indicated by *DELETE*.

Assuming that our spatial parsing algorithm is defined by some function $\mathit{parse}_S : (I\langle\Omega\rangle \times E\langle\Omega\rangle \times D\langle\Omega, k\rangle) \rightarrow I\langle\Omega\rangle$, we can use Φ_S from Eq. 4.119 to partially refine $A_P\langle\Omega, n, \alpha, \beta, \Phi, \varphi\rangle$ (Eq. 4.106), $T_P\langle\Omega, n, \alpha, \beta, \Phi, \varphi\rangle$ (Eq. 4.109), and $G_P\langle\Omega, n, \alpha, \beta, \Phi, \varphi\rangle$ from Eq. 4.115 as follows:

$$\begin{aligned} A_S\langle\Omega, n, \alpha, \beta\rangle &:= A_P\langle\Omega, n, \alpha, \beta, \Phi_S, \mathit{parse}_S\rangle \\ T_S\langle\Omega, n, \alpha, \beta\rangle &:= T_P\langle\Omega, n, \alpha, \beta, \Phi_S, \mathit{parse}_S\rangle \\ G_S\langle\Omega, n, \alpha, \beta\rangle &:= G_P\langle\Omega, n, \alpha, \beta, \Phi_S, \mathit{parse}_S\rangle \end{aligned} \tag{4.120}$$

This way our generic parser model $A_P\langle\Omega, n, \alpha, \beta, \Phi, \varphi\rangle$ turns into a parameterised model for spatial parsers $A_S\langle\Omega, n, \alpha, \beta\rangle$.

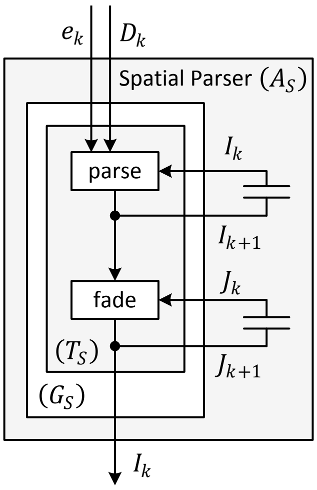


Figure 4.16: Blockdiagram of spatial parser model A_S as defined in Eq. 4.121

For our research prototype we specified in Sect. 4.1.4 several refinements of our workspace model A_W (Eq. 4.66). In Sect. 4.3 this has led to . . .

$$\Omega = \mathbb{N}_0 \text{ and } k = 16.$$

Both parameters can be used now for refining or rather configuring $A_S\langle\Omega, n, \alpha, \beta\rangle$, $T_S\langle\Omega, n, \alpha, \beta\rangle$ and $G_S\langle\Omega, n, \alpha, \beta\rangle$. As an example we set α, β to . . .

$$\alpha = 0.95 \text{ and } \beta = 0.05.$$

This results in the following default configuration for spatial parsers A_S:

$$\begin{aligned} A_S &:= A_S\langle\mathbb{N}_0, 16, 0.95, 0.05\rangle \\ T_S &:= T_S\langle\mathbb{N}_0, 16, 0.95, 0.05\rangle \\ G_S &:= G_S\langle\mathbb{N}_0, 16, 0.95, 0.05\rangle \end{aligned} \tag{4.121}$$

If one substitutes now in Fig. 4.15 components A_P, T_P and G_P for A_S, T_S and G_S from Eq. 4.121, then one gets the blockdiagram in Fig. 4.16.

Three-Stage Spatial Parsing Algorithm

According to our considerations from Sect. 2.1 (page 30), spatial parsers should rather *imitate* humans in the way they perceive structure than checking a canvas against pre-defined and supposedly universal patterns (such as heaps, piles, stacks etc.). In human perception, atomic objects are recognized first and more complex structures emerge from simpler ones, not vice-versa. This bottom-up principle gets also reflected in our spatial parsing algorithm:

$$\begin{aligned} \mathit{parse}_S : \big(I\langle\Omega\rangle \times E\langle\Omega\rangle \times D\langle\Omega,k\rangle\big) &\to I\langle\Omega\rangle, \\ (I,e,D) &\mapsto \mathit{normalize}\Big(\mathit{parse\_list\_structures}\big(\mathit{create\_terminals}(D)\big)\Big) \end{aligned} \tag{4.122}$$

parse_S follows our requirement from Eq. 4.107 and maps triples $(I,e,D) \in (I\langle\Omega\rangle \times E\langle\Omega\rangle \times D\langle\Omega,n\rangle)$ to interpretations $\in I\langle\Omega\rangle$. This mapping proceeds in three stages: Firstly, $\mathit{create\_terminals}(D)$ converts info unit data $D \in D\langle\Omega,k\rangle$ into a collection of terminal symbols. In a second step, the bottom-up algorithm *parse_list_structures* transforms these terminals into a tree of alignment-oriented structures (i. e., "parse tree"). Finally, function *normalize* converts this hierarchical structure into a weighted, "flat" graph $\in I\langle\Omega\rangle$. Note, that arguments I and e are not used in the current version of this algorithm.

Before we go into any details on *create_terminals*, *parse_list_structures* and *normalize*, some basic definitions are needed:

The previously mentioned bottom-up parse requires internal type identification of terminals and non-terminals and therefore needs definition of distinct symbols. For this purpose we introduce the discrete set *StructureType*:

$$\mathit{StructureType} = \left\{ \begin{array}{c} \mathit{ATOM}, \\ \mathit{UNALIGNED}, \\ \mathit{HORIZONTAL\_LIST}, \\ \mathit{VERTICAL\_LIST}, \\ \mathit{DIAGONAL\_LIST0}, \\ \mathit{DIAGONAL\_LIST1} \end{array} \right\} \tag{4.123}$$

Internally our spatial parser distinguishes between atoms, unaligned objects and collections of objects with horizontal, vertical or diagonal alignment. Analogous to our definitions from Sect. 4.2.2 we make a distinction between diagonal alignment from top-left to bottom-right (*DIAGONAL_LIST0*) and from top-right to bottom-left (*DIAGONAL_LIST1*). More complex structure types (such as tables, stacks, piles etc.) have deliberately *not* been provided. We rather build on two of the most fundamental attributes of spatial structure and hence of spatial perception: *spatial proximity* and *alignment*. With this we intend to avoid (potentially wrong) over-interpretation of structure.

As pointed out already, internally our parsing algorithm works with symbols. We denote the basic set of such parser symbols as $S\langle\Omega, k\rangle$ and define it as a cartesian product:

$$S\langle\Omega, k\rangle = \Big(\mathit{ID} \times \big(\mathit{StructureType} \cup \{\varepsilon\}\big) \times \mathit{Bounds}\langle k\rangle \times \big(\Omega \cup \{\varepsilon\}\big) \times \mathit{List}\langle \mathit{ID}\rangle\Big) \tag{4.124}$$

Here, *ID* shall be regarded as a set of unique identifiers and ε represents some empty (or NULL) element. The definition of $\mathit{Bounds}\langle k\rangle$ can be found in Eq. 4.73 and *List* was introduced already with Eq. 4.91.

Elements of such five-tuples $\in S\langle\Omega, k\rangle$ are denoted as follows:

$$\forall s \in S\langle\Omega, k\rangle : \quad s := \begin{pmatrix} s.\mathit{id}, \\ s.\mathit{type}, \\ s.\mathit{bounds}, \\ s.\mathit{info\_unit}, \\ s.\mathit{child\_ids} \end{pmatrix} \tag{4.125}$$

Thus, parser symbols $s \in S\langle\Omega, k\rangle$ have five attributes: (1) a unique identifier $s.\mathit{id} \in \mathit{ID}$ (we will see afterwards what this is used for); (2) an assigned structure type $s.\mathit{type} \in (\mathit{StructureType} \cup \{\varepsilon\})$; (3) geometrical properties encoded as $s.\mathit{bounds} \in \mathit{Bounds}\langle k\rangle$; (4) the information unit they are (possibly) linked with $s.\mathit{info\_unit} \in (\Omega \cup \{\varepsilon\})$ and (5) a list of child symbol identifiers $s.\mathit{child\_ids} \in \mathit{List}\langle \mathit{ID}\rangle$ (note: our algorithm operates on symbol *hierarchies*).

In the following we will frequently work with collections of symbols $\subset S\langle\Omega, k\rangle$ and thus with the power set of $S\langle\Omega, k\rangle$. However, we will not build directly on $2^{S\langle\Omega,k\rangle}$, but instead we use a constrained subset denoted as $P_S\langle\Omega, k\rangle$:

$$P_S\langle\Omega, k\rangle = \left\{ S \;\middle|\; \begin{array}{c} S \in 2^{S\langle\Omega,k\rangle}, \\ \nexists\, \{s, s'\} \in \binom{S}{2} : s.\mathit{id} = s'.\mathit{id} \end{array} \right\} \tag{4.126}$$

$P_S\langle\Omega, k\rangle$ was designed to ensure, that each s contained in an $S \subseteq S\langle\Omega, k\rangle$ can be clearly distinguished from other $s' \in S$, even though s and s' might have identical type, bounds etc. This makes elements of such S *referenceable*. From a software developer's perspective, one could imagine $S \in P_S\langle\Omega, k\rangle$ as occupied object memory and identifiers assigned to $s \in S$ as memory addresses or rather as pointers. At least you could implement them like this.

Following this idea, our definition of $P_S\langle\Omega, k\rangle$ requires two basic operations: (1) *create_symbol* (Eq. 4.127) and (2) *get_symbol* (Eq. 4.128).

$$
\begin{aligned}
\mathit{create\_symbol}\colon &\begin{pmatrix} \quad P_S\langle\Omega,k\rangle \\ \times\ \big(\mathit{StructureType}\cup\{\varepsilon\}\big) \\ \times\ \mathit{Bounds}\langle k\rangle \\ \times\ \big(\Omega\cup\{\varepsilon\}\big) \\ \times\ \mathit{List}\langle \mathit{ID}\rangle \end{pmatrix} \to S\langle\Omega,k\rangle, \\
&(S,a_1,a_2,a_3,a_4)\mapsto(\mathit{id},a_1,a_2,a_3,a_4)\,, \\
&(\mathit{id}\in \mathit{ID})\wedge(\nexists s\in S : s.\mathit{id}=\mathit{id})
\end{aligned}
\tag{4.127}
$$

create_symbol accepts a set of parser symbols $S \in P_S\langle\Omega,k\rangle$ and four arguments $a_1,\dots,a_4$, that are (according to Eq. 4.124) needed for setting up a new symbol tuple $\in S\langle\Omega,k\rangle$. These arguments $a_1,\dots,a_4$ are then combined with some $\mathit{id}\in\mathit{ID}$, so that there is no $s\in S$ for which $s.\mathit{id}=\mathit{id}$. This way *create_symbol* ensures, that new symbols always get an identifier that is not in use already in a given $S\in P_S\langle\Omega,k\rangle$. The only thing that remains to be done, is adding new symbols $s=\mathit{create\_symbol}(S,a_1,\dots,a_4)$ to S. For this, however, a statement such as $S\leftarrow(S\cup\{s\})$ is completely sufficient. In fact, this is exactly what we use in our algorithms.

Once created and added to $S\in P_S\langle\Omega,k\rangle$, symbols $s\in S$ can be retrieved again using the function *get_symbol*:

$$
\begin{aligned}
\mathit{get\_symbol}\colon &\Big(P_S\langle\Omega,k\rangle\times\big(\mathit{ID}\cup\{\varepsilon\}\big)\Big)\to\big(S\langle\Omega,k\rangle\cup\{\varepsilon\}\big)\,, \\
&\forall\,(S,\mathit{id})\in\Big(P_S\langle\Omega,k\rangle\times\big(\mathit{ID}\cup\{\varepsilon\}\big)\Big) : \\
&\mathit{get\_symbol}(S,\mathit{id}) = \begin{cases} s\ , \text{ if } (\mathit{id}\neq\varepsilon)\wedge(\exists! s\in S : s.\mathit{id}=\mathit{id}) \\ \varepsilon\ , \text{ else} \end{cases}
\end{aligned}
\tag{4.128}
$$

$\mathit{get\_symbol}(S,\mathit{id})$ tries to use a set of declared symbols $S\in P_S\langle\Omega,k\rangle$ in order to resolve a given $\mathit{id}\in(\mathit{ID}\cup\{\varepsilon\})$. When such an *id* is $\neq\varepsilon$ (i. e., it is not a "NULL pointer") and there is exactly one $s\in S$ for which $s.\mathit{id}=\mathit{id}$, then *get_symbol* delivers s as result. Otherwise *id* cannot be resolved in S, and hence ε ("nothing") is returned. This way we keep *get_symbol* totally defined.

In later algorithms we will operate on tree structures formed by parser symbols $s\in S\langle\Omega,k\rangle$. Hence, we will frequently work with child symbol lists $s.\mathit{child\_ids}$. Therefore, in order to decrease effort of expression, it makes sense to wrap the most frequently used list-features in helper routines. We define two such auxiliary functions that are supposed to simplify child symbol access. These access operations are: (1) *child_count* (Eq. 4.129) and (2) *child* (Eq. 4.130).

child_count(s) is nothing more than a shortcut to the number of child symbols beneath s. That is, it simply takes s' list of child ids and detects *size*$(s.\mathit{child\_ids})$, as defined in Eq. 4.93:

$$\mathit{child\_count}\colon S\langle\Omega,k\rangle \to \mathbb{N}_0,\ s \mapsto \mathit{size}\,(s.\mathit{child\_ids}) \tag{4.129}$$

Index-based access on single child symbols can be achieved via *child*(S,s,i). Here, s represents a parent symbol, i is a list index in $s.\mathit{child\_ids}$ and S is the basic set of symbols where the child with id *get*$(s.\mathit{child\_ids},i)$ is supposed to be found. Formally we define *child* as follows:

$$\begin{aligned}\mathit{child}\colon &\big(P_S\langle\Omega,k\rangle \times S\langle\Omega,k\rangle \times \mathbb{N}_0\big) \to \big(S\langle\Omega,k\rangle \cup \{\varepsilon\}\big)\,,\\ &(S,s,i) \mapsto \mathit{get\_symbol}\big(S, \mathit{get}\,(s.\mathit{child\_ids},i)\big)\end{aligned} \tag{4.130}$$

Both, *child_count* (Eq. 4.129) and *child* (Eq. 4.130) are merely used for notational simplification. This means, we could also do without them and use more verbose expressions instead. Our following definitions, however, are integral parts of the parsing algorithm and cannot be substituted:

This includes two basic types of symbols: (1) $S_{atom}\langle\Omega,k\rangle$ (Eq. 4.131) and (2) $S_{struct}\langle\Omega,k\rangle$ (Eq. 4.132). Both sets are derivatives of $S\langle\Omega,k\rangle$ (Eq. 4.124).

Symbols that can be used as structure elements, but do *not* describe structure themselves, are denoted as "atoms". Formally we define them as:

$$S_{atom}\langle\Omega,k\rangle = \left\{ s \;\middle|\; \begin{matrix} s \in S\langle\Omega,k\rangle, \\ (s.\mathit{type} = \mathit{ATOM}) \\ \wedge \\ \left(\begin{matrix} (s.\mathit{info\_unit} \neq \varepsilon) \wedge \big(\mathit{child\_count}\,(s) = 0\big) \\ \vee \\ (s.\mathit{info\_unit} = \varepsilon) \wedge \big(\mathit{child\_count}\,(s) = 1\big) \end{matrix}\right) \end{matrix} \right\} \subset S\langle\Omega,k\rangle \tag{4.131}$$

These are symbols $s \in S\langle\Omega,k\rangle$ that are explicitely marked with structure type $s.\mathit{type} = \mathit{ATOM}$ and for which either $(s.\mathit{info\_unit} \neq \varepsilon) \wedge (\mathit{child\_count}(s) = 0)$ or $(s.\mathit{info\_unit} = \varepsilon) \wedge (\mathit{child\_count}(s) = 1)$ holds true. This means, either symbols $s \in S_{atom}\langle\Omega,k\rangle$ are atomic leaf nodes with $s.\mathit{info\_unit} \neq \varepsilon$ or they are internal nodes with exactly one child and without a reference to any information unit.

By contrast, symbols that *do* represent structure are defined by $S_{struct}\langle\Omega,k\rangle$:

$$S_{struct}\langle\Omega,k\rangle = \left\{ s \;\middle|\; \begin{matrix} s \in S\langle\Omega,k\rangle, \\ (s.\mathit{type} \neq \mathit{ATOM}) \\ \wedge \\ (s.\mathit{info\_unit} = \varepsilon) \wedge \big(\mathit{child\_count}\,(s) > 0\big) \end{matrix} \right\} \subset S\langle\Omega,k\rangle \tag{4.132}$$

Structure symbols $s \in S_{struct}\langle\Omega, k\rangle$ are tagged with a structure type $\neq$ *ATOM*; thus the assigned value could also be ε. Additionally they must be linked with *at least one* child symbol as structure element. Consequently, symbols $\in S_{struct}\langle\Omega, k\rangle$ never can be found on leaf node level. As a last requirement, $s.info\_unit$ should always be ε. Thus, structures do not point directly to carriers of information in a workspace; only their elements on leaf node level do.

Creation of symbols $\in S_{struct}\langle\Omega, k\rangle$ is defined by the following algorithm:

$create\_structure$:

8 $$\begin{pmatrix} P_S\langle\Omega, k\rangle \\ \times \big(StructureType \setminus \{ATOM\} \cup \{\varepsilon\}\big) \\ \times \big(List\langle ID\rangle \setminus \{list_\varepsilon\}\big) \end{pmatrix} \to \begin{pmatrix} S_{struct}\langle\Omega, k\rangle \\ \cup \\ \{\varepsilon\} \end{pmatrix}, \begin{pmatrix} S, \\ type, \\ ids \end{pmatrix} \mapsto s,$$

```
1: $s \leftarrow \varepsilon$
2: $s_0 \leftarrow get\_symbol\big(S, get(ids, 0)\big)$    ▷ get first tentitive child symbol $s_0$
3: if $s_0 \neq \varepsilon$ then
4:     $b \leftarrow s_0.bounds$
5:     for $i = 1$ to $\big(size(ids) - 1\big)$ do ▷ calculate union of all child bounds $b$
6:         $s_i \leftarrow get\_symbol\big(S, get(ids, i)\big)$
7:         if $s_i = \varepsilon$ then
8:             return $s$
9:         end if
10:        $b \leftarrow union(b, s_i.bounds)$    ▷ see: Eq. 4.83 for definition of $union$
11:    end for
12:    $s \leftarrow create\_symbol(S, type, b, \varepsilon, ids)$    ▷ new $s \in S_{struct}\langle\Omega, k\rangle$
13: end if
14: return $s$
```

create_structure maps triples $(S, type, ids)$ to structure symbols $s \in S_{struct}\langle\Omega, k\rangle$ (in the fault case ε is used as return value instead). S is a set of predefined symbols needed for resolving structure element *ids*, *type* is either a structure type $\neq$ *ATOM* or ε and the last argument *ids* represents a non-empty list of structure element identifiers.

As an example, let us create a horizontal list formed from two atomic list elements; we set …

$$type = HORIZONTAL\_LIST \text{ and } ids = (s_0, s_1)$$

Both atoms s_0, s_1 shall be given by …

$$S = \left\{ \begin{array}{l} (s_0, ATOM, b_0, u_0, list_\varepsilon), \\ (s_1, ATOM, b_1, u_1, list_\varepsilon) \end{array} \right\} \subset S_{atom}\langle\Omega, k\rangle$$

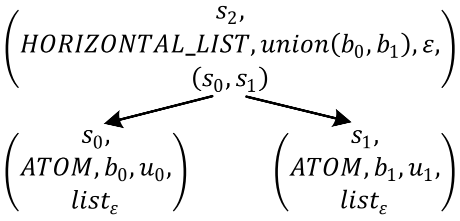

s_2, HORIZONTAL_LIST, union(b_0, b_1), ε, (s_0, s_1)
s_0, ATOM, b_0, u_0, $list_\varepsilon$
s_1, ATOM, b_1, u_1, $list_\varepsilon$

Figure 4.17: hierarchical relationship between horizontal list s_2 and atoms s_0, s_1

Here we assume, that $ID = \{s_0, s_1, \ldots\}$ and that $\Omega = \{u_0, u_1, \ldots\}$. This is why both attributes *id* and *info_unit* are set to s_0, s_1 and u_0, u_1 respectively. b_0, b_1 shall be arbitrary bounding volumes $\in Bounds\langle k\rangle$. Their exact position and dimensions do not matter in this example.

Given these definitions and assuming that *create_symbol* from Eq. 4.127 assigns incremented symbol ids $s_{i+1} \in ID$ to new symbols, *create_structure* $(S, type, ids)$ will deliver the following:

$$create\_structure\left(\begin{matrix}\left\{\begin{matrix}(s_0, ATOM, b_0, u_0, list_\varepsilon),\\ (s_1, ATOM, b_1, u_1, list_\varepsilon)\end{matrix}\right\},\\ HORIZONTAL\_LIST,\\ (s_0, s_1)\end{matrix}\right) = \left(\begin{matrix}s_2,\\ HORIZONTAL\_LIST,\\ union\,(b_0, b_1),\\ \varepsilon,\\ (s_0, s_1)\end{matrix}\right)$$

The apparent hierarchical relationship between horizontal list s_2 and atoms s_0, s_1 is illustrated in Fig. 4.17.

As already pointed out at the beginning, our spatial parsing algorithm realises a bottom-up parse. In this context, it works with two specific categories of symbols: (1) *terminal* and (2) *non-terminal* symbols.

Terminal symbols are defined as atoms $\in S_{atom}\langle\Omega, k\rangle$ (Eq. 4.131) that do *not* have child symbols. Hence, they are atomic leaf nodes:

$$S_{terminal}\langle\Omega, k\rangle = \left\{ s \;\middle|\; \begin{matrix} s \in S_{atom}\langle\Omega, k\rangle, \\ child\_count\,(s) = 0 \end{matrix} \right\} \subset S_{atom}\langle\Omega, k\rangle \tag{4.133}$$

The power set of $S_{terminal}\langle\Omega, k\rangle$ can be defined as a subset of $P_S\langle\Omega, k\rangle$ (Eq. 4.126):

$$P_{terminal}\langle\Omega, k\rangle = \left\{ S \;\middle|\; \begin{matrix} S \in P_S\langle\Omega, k\rangle, \\ S \subset S_{terminal}\langle\Omega, k\rangle \end{matrix} \right\} \subset P_S\langle\Omega, k\rangle \tag{4.134}$$

The following operation specifies how terminal symbols are instantiated:

$$\begin{aligned} \mathit{create\_terminal}\colon \big(P_S\langle\Omega,k\rangle \times \mathit{Bounds}\langle k\rangle \times \Omega\big) &\to S_{terminal}\langle\Omega,k\rangle, \\ (S,b,u) &\mapsto \mathit{create\_symbol}(S, \mathit{ATOM}, b, u, \mathit{list}_\varepsilon) \end{aligned} \tag{4.135}$$

The non-terminal counterpart to $S_{terminal}\langle\Omega,k\rangle$ is $S_{nonterminal}\langle\Omega,k\rangle$. Non-terminals either can be structure symbols $S_{struct}\langle\Omega,k\rangle$ or non-terminal atoms $(S_{atom}\langle\Omega,k\rangle \setminus S_{terminal}\langle\Omega,k\rangle)$:

$$S_{nonterminal}\langle\Omega,k\rangle = S_{struct}\langle\Omega,k\rangle \cup \big(S_{atom}\langle\Omega,k\rangle \setminus S_{terminal}\langle\Omega,k\rangle\big) \tag{4.136}$$

Similar to Eq. 4.134, also the power set of $S_{nonterminal}\langle\Omega,k\rangle$ can be expressed as a subset of $P_S\langle\Omega,k\rangle$ (Eq. 4.126):

$$P_{nonterminal}\langle\Omega,k\rangle = \left\{ S \;\middle|\; \begin{array}{c} S \in P_S\langle\Omega,k\rangle, \\ S \subset S_{nonterminal}\langle\Omega,k\rangle \end{array} \right\} \subset P_S\langle\Omega,k\rangle \tag{4.137}$$

Spatial Parsing Algorithm – Stage 1 – Creation of Terminal Symbols

Based on Eq. 4.134 and Eq. 4.135 we describe the first step of our three-stage spatial parsing algorithm parse_S (Eq. 4.122) as follows:

```
9 $\mathit{create\_terminals}\colon D\langle\Omega,k\rangle \to P_{terminal}\langle\Omega,k\rangle,\ (V,p) \mapsto S,$
1: $S \leftarrow \emptyset$
2: for all $u \in V$ do
3:     $S \leftarrow S \cup \Big\{\mathit{create\_terminal}\big(S, p(u).\mathit{bounds}, u\big)\Big\}$        ▷ see: Eq. 4.135
4: end for
5: return $S$
```

create_terminals takes a "map" $(V,p) \in D\langle\Omega,k\rangle$ (Eq. 4.85) and creates for each information unit $u \in V$ a new terminal symbol $\in S_{terminal}\langle\Omega,k\rangle$ (Eq. 4.133). Following Eq. 4.135 only information unit u and the assigned bounding volume $p(u).\mathit{bounds}$ are needed for this. Terminals created this way are collected in some $S \in P_{terminal}\langle\Omega,k\rangle$ (Eq. 4.134).

As an example, let us assume that (V,p) defines two red rectangles with ids u_0 and u_1 (provided that $\Omega = \{u_0, u_1, \ldots\}$):

$$(V,p) = \left(\{u_0,u_1\}, \left\{ \begin{array}{l} \big(u_0, (b_0, 0, \mathit{RECTANGLE}, (255,0,0), \text{“}\mathit{content0}\text{”})\big), \\ \big(u_1, (b_1, 0, \mathit{RECTANGLE}, (255,0,0), \text{“}\mathit{content1}\text{”})\big) \end{array} \right\} \right)$$

Both objects are located on the same layer 0 and their content is set to string literals "*content0*" and "*content1*" respectively. b_0, b_1 are arbitrary bounding volumes $\in \mathit{Bounds}\langle k\rangle$.

Assuming again, that $ID = \{s_0, s_1, \ldots\}$, *create_terminals* transforms this definition of (V, p) into the following set of terminal symbols S:

$$S = \left\{ \begin{array}{l} (s_0, \mathit{ATOM}, b_0, u_0, \mathit{list}_\varepsilon)\,, \\ (s_1, \mathit{ATOM}, b_1, u_1, \mathit{list}_\varepsilon) \end{array} \right\} \in P_{\mathit{terminal}}\langle \Omega, k\rangle$$

Apparently, we adopted here only spatial properties in form of bounding volumes b_0, b_1 into the newly generated atoms. Other attributes (such as layer, shape, color, content etc.) are of no relevance to our spatial analysis and were therefore discarded.

In the second stage of our spatial parsing algorithm, terminals generated according to Alg. 9 undergo a bottom-up analysis (see parse_S in Eq. 4.122).

Before we describe how such *parse_list_structures* (*create_terminals* $((V, p))$) or rather *parse_list_structures* (S) proceeds, let us first define a series of required auxiliary structures:

With our previous definitions of parser symbols $S\langle \Omega, k\rangle$ (Eq. 4.124) it is already possible to describe symbol hierarchies (i. e., trees of symbols). For this, see in particular *create_structure* from Alg. 8. Symbol trees are inextricably linked with bottom-up parsing and thus play an important role for the second phase of our three-stage spatial parsing algorithm. Therefore, in *parse_list_structures* we will make intensive use of our previous definitions on hierarchical linking of symbols. In addition to this, however, we need another alternative formalism for defining trees. This is described below.

In the following we will repeatedly work with (rooted) out-trees; that is, with trees having the following properties:

$$\begin{aligned} \mathit{OutTree}\langle Q\rangle &: \text{set of all (rooted) out-trees } T := (V, E, r), \\ V(T) &: \text{non-empty set of vertices ; } \big(V(T) \neq \emptyset\big) \wedge \big(V(T) \subseteq Q\big) \\ E(T) &: \text{set of directed edges } \big(v_i, v_j\big)\,,\ v_i, v_j \in V(T) \\ r(T) &: \text{root vertext of } T \text{ (i. e., } r \in V(T),\ d_T^-(r) = 0 \wedge d_T^+(r) \geq 0) \end{aligned} \tag{4.138}$$

In total there are *four* operations on trees $T \in \mathit{OutTree}\langle Q\rangle$ that will be of relevance to our parsing algorithm: (1) *leafs* (Eq. 4.139); (2) *children* (Eq. 4.140); (3) *level* (Eq. 4.141) and (4) *add_child* (Eq. 4.142).

The first operation *leafs* detects leaf nodes in out trees. This means, it takes a given $T \in \mathit{OutTree}\langle Q\rangle$ and delivers the set of all vertices $v \in V(T)$ which do not have any children; that is, for which $d^+_T(v) = 0$. For example, $\mathit{leafs}\left((\{r\}, \emptyset, r)\right)$ would return $\{r\}$.

$$\mathit{leafs} : \mathit{OutTree}\langle Q\rangle \to 2^Q,\ T \mapsto \left\{v \middle| v \in V(T) \wedge d^+_T(v) = 0\right\} \tag{4.139}$$

Access on a node's child vertices is defined by function *children*. It accepts a $T \in \mathit{OutTree}\langle Q\rangle$ and a vertex $v \in Q$ and returns the set of all vertices that are marked in $E(T)$ as children of v. If v is $\notin V(T)$ or if $v \in \mathit{leafs}(T)$ then $\emptyset$ is returned. A good example for this would be $\mathit{children}\left((\{r\}, \emptyset, r), r\right) = \emptyset$.

$$\mathit{children} : \big(\mathit{OutTree}\langle Q\rangle \times Q\big) \to 2^Q,\ (T, v) \mapsto \left\{v' \middle| \exists!\left(v, v'\right) \in E\left(T\right)\right\} \tag{4.140}$$

Levels of vertices v in trees $T \in \mathit{OutTree}\langle Q\rangle$ are determined by $\mathit{level}(T, v)$. For this we assume, that there is some function $\mathit{DISTANCE}(v_0, v_1, G(V, E))$ defined, which can tell us the length of the shortest path from vertex v_0 to v_1 in a given graph $G(V, E)$. Here we also assume, that such a function is only defined for $v_0, v_1 \in V$. As an example, $\mathit{level}(T, r(T))$ would be 0, since $\mathit{DISTANCE}(r(T), r(T), G(V(T), E(T))) = 0$ either.

$$\begin{aligned}&\mathit{level} : \big(\mathit{OutTree}\langle Q\rangle \times Q\big) \rightharpoonup \mathbb{N}_0,\\ &\qquad (T, v) \mapsto \mathit{DISTANCE}\Big(r\left(T\right), v, G\left(V\left(T\right), E\left(T\right)\right)\Big),\ v \in V\left(T\right)\end{aligned} \tag{4.141}$$

The last $\mathit{OutTree}\langle Q\rangle$-"method" is *add_child*. Applied on a given out tree T, $\mathit{add\_child}(T, v, v')$ extends T by a new leaf node v'. The second argument v indicates the vertex, beneath which v' is to be inserted. $\mathit{add\_child}\left((\{r\}, \emptyset, r), r, v\right)$, for instance, would result in $(\{r, v\}, \{(r, v)\}, r)$.

$$\begin{aligned}&\mathit{add\_child} : \big(\mathit{OutTree}\langle Q\rangle \times Q \times Q\big) \rightharpoonup \mathit{OutTree}\langle Q\rangle,\\ &\qquad \left(T, v, v'\right) \mapsto \left(\left(V\left(T\right) \cup \left\{v'\right\}\right), \left(E\left(T\right) \cup \left\{\left(v, v'\right)\right\}\right), r\left(T\right)\right),\\ &\qquad\qquad v \in V\left(T\right) \wedge v' \notin V\left(T\right)\end{aligned} \tag{4.142}$$

Note, that in the following we will not work directly with $\mathit{OutTree}\langle Q\rangle$ but with a refined subset denoted as $T_I\langle \Omega, k\rangle$:

$$T_I\langle \Omega, k\rangle \subset \mathit{OutTree}\langle P_S\langle \Omega, k\rangle\rangle \tag{4.143}$$

Apparently, subset $T_I\langle \Omega, k\rangle$ identifies specific out trees whose vertices are full *sets* of parser symbols rather than simple objects. That is, the tree node-"type" is $P_S\langle \Omega, k\rangle$. For details see our definition of $P_S\langle \Omega, k\rangle$ in Eq. 4.126.

Each so-called "Interpretation Tree" $T \in T_I\langle\Omega, k\rangle$ is subject to a number of conditions:

Firstly, root-vertices r of trees T shall comprise only terminal symbols, that is $r(T)$ may include only symbols $\in S_{terminal}\langle\Omega, k\rangle$ (Eq. 4.133). Thus we define:

$$r(T) \in P_{terminal}\langle\Omega, k\rangle \tag{4.144}$$

In contrast to root $r(T)$, remaining vertices $V(T) \setminus \{r(T)\}$ should contain only non-terminals $\in S_{nonterminal}\langle\Omega, k\rangle$ (Eq. 4.136):

$$\Big(V\left(T\right) \setminus \big\{r\left(T\right)\big\}\Big) \subset P_{nonterminal}\langle\Omega, k\rangle \tag{4.145}$$

Each parser symbol, no matter if terminal or non-terminal, may occur only once in T. Thus, for all $S \in V(T)$:

$$\left(\bigcap_{S \in V(T)} S\right) = \emptyset \tag{4.146}$$

When we denote the total set of parser symbols included in T as $S_\Sigma(T)$ and define it as follows …

$$S_\Sigma(T) := \left(\bigcup_{S \in V(T)} S\right) \in P_S\langle\Omega, k\rangle \tag{4.147}$$

… then $\forall s \in S_\Sigma(T)$:

$$S_\Psi\left(T, s\right) := \left(\bigcup_{i=0}^{child\_count(s)-1} \Big\{child\left(S_\Sigma(T), s, i\right)\Big\}\right) \subset S_\Sigma(T) \tag{4.148}$$

This means, that every single non-terminal in T has to be linked with child symbols that are also included in T. Here, $S_\Psi(T, s)$ represents the set of all child symbols of s in T. For a definition of *child* see Eq. 4.130.

Finally, $\forall(S_i, S_j) \in E(T)$:

$$\Big(0 < \left|S_j\right| < |S_i|\Big) \wedge \left(\left(\bigcap_{s \in S_j} S_\Psi\left(T, s\right)\right) = \emptyset\right) \wedge \left(\left(\bigcup_{s \in S_j} S_\Psi\left(T, s\right)\right) = S_i\right) \tag{4.149}$$

In plain language, this means that each (non-root) vertex $S \in (V(T) \setminus \{r(T)\})$ comprises *at least one* symbol (i. e., $|S| > 0$) and child vertices S_j always contain *less* symbols than their parents S_i (i. e., $\left|S_j\right| < |S_i|$). Furthermore, each $s \in S_i$ must be assigned *exactly one* $s' \in S_j$ and each $s' \in S_j$ has to be assigned *at least one* $s \in S_i$.

This is best explained with an example. Suppose, we have an interpretation tree $T = (V, E, r) \in T_I\langle\Omega, k\rangle$ with $V(T)$, $E(T)$ and $r(T)$ being defined as:

$$V(T) = \{S_0, S_1, S_2, S_3\} \;\; ; \;\; E(T) = \left\{ \begin{array}{l} (S_0, S_1)\,, \\ (S_1, S_2)\,, \\ (S_0, S_3) \end{array} \right\} \;\; ; \;\; r(T) = S_0$$

To simplify things, we set $S\langle\Omega, k\rangle$ equal to a set of symbolic placeholders s_i (for index $i \geq 0$):

$$S\langle\Omega, k\rangle = \{s_0, s_1, \ldots\}$$

$S_0 \in V(T)$ shall comprise three terminal symbols $s_0, s_1, s_2 \in S_{terminal}\langle\Omega, k\rangle$:

$$S_0 = \{s_0, s_1, s_2\} \in P_{terminal}\langle\Omega, k\rangle$$

For remaining elements of $V(T)$ we define:

$$S_1, S_2, S_3 \in P_{nonterminal}\langle\Omega, k\rangle$$

In detail these sets of symbols are:

$$S_1 = \{s_3, s_4\} \;;\; S_2 = \{s_5\} \;;\; S_3 = \{s_6\}$$

Based on these specifications, the total set of symbols $S_\Sigma(T)$ from Eq. 4.147 can be determined as:

$$S_\Sigma(T) = S_0 \cup S_1 \cup S_2 \cup S_3 = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6\} \in P_S\langle\Omega, k\rangle$$

Individual parent-child relations between these symbols are described below:

s_0, s_1, s_2 are terminals $\in S_{terminal}\langle\Omega, k\rangle$ and therefore have *no* child symbols:

$$s_0.\mathit{child\_ids} = s_1.\mathit{child\_ids} = s_2.\mathit{child\_ids} = \mathit{list}_\varepsilon$$

Thus, the following should apply:

$$S_\Psi\left(T, s_0\right) = S_\Psi\left(T, s_1\right) = S_\Psi\left(T, s_2\right) = \emptyset$$

For remaining s_3, s_4, s_5, s_6, however, $S_\Psi(T, s_i)$ should be $\neq \emptyset$ (for $3 \leq i \leq 6$):

$$\begin{array}{ll} s_3.\mathit{child\_ids} = (s_0.\mathit{id}, s_1.\mathit{id}) & \Rightarrow S_\Psi\left(T, s_3\right) = \{s_0, s_1\} \\ s_4.\mathit{child\_ids} = (s_2.\mathit{id}) & \Rightarrow S_\Psi\left(T, s_4\right) = \{s_2\} \\ s_5.\mathit{child\_ids} = (s_3.\mathit{id}, s_4.\mathit{id}) & \Rightarrow S_\Psi\left(T, s_5\right) = \{s_3, s_4\} \\ s_6.\mathit{child\_ids} = (s_0.\mathit{id}, s_1.\mathit{id}, s_2.\mathit{id}) & \Rightarrow S_\Psi\left(T, s_6\right) = \{s_0, s_1, s_2\} \end{array}$$

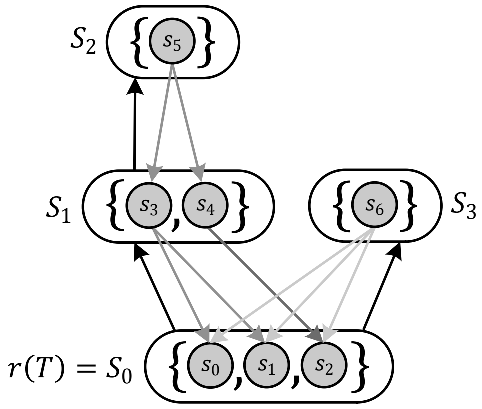


Figure 4.18: sample interpretation tree $T \in T_I\langle\Omega, k\rangle$

The interpretation tree formed from these definitions is illustrated graphically in Fig. 4.18. From this illustration it becomes particularly clear that interpretation trees $T_I\langle\Omega, k\rangle$ are composites or rather mixtures of *primary* and *secondary* structures. This means, in every $T \in T_I\langle\Omega, k\rangle$ we can differentiate between two structures, where one is embedded in the other. The primary structure in our example would be the out-tree of symbol sets defined by $V(T)$, $E(T)$ and $r(T)$. In Fig. 4.18 this is illustrated with black borders around braces that are connected by black upward pointing arrows. Embedded in this "host"-structure there is a (directed) graph, consisting of grey circles as nodes (each labeled as a symbol $s_i \in S_\Sigma(T)$) and grey arrows as edges pointing downwards. This symbol graph represents our secondary structure and is determined, as we know already from Eq. 4.125, by symbol attribute *s.child_ids*.

Another good example for an interpretation tree would be:

$$T = \Big(\{\emptyset\}, \emptyset, \emptyset\Big) \in T_I\langle\Omega, k\rangle$$

This T comprises of a single vertex only, $r(T)$, which does not contain any symbols (i. e., $r(T) = \emptyset$). For such a T the following applies:

$$r(T) = \emptyset \in P_{terminal}\langle\Omega, k\rangle \text{ (Eq. 4.144) ;}$$
$$\Big(V(T) \setminus \{r(T)\}\Big) = \emptyset \subset P_{nonterminal}\langle\Omega, k\rangle \text{ (Eq. 4.145)}$$
$$\text{and}$$
$$\Big(\bigcap\nolimits_{S \in \{\emptyset\}} S\Big) = \emptyset \text{ (Eq. 4.146) ; } S_\Sigma(T) = \emptyset \text{ (Eq. 4.147).}$$

This is why both is satisfied Eq. 4.148 and, since $E(T) = \emptyset$, also Eq. 4.149. Thus $(\{\emptyset\}, \emptyset, \emptyset)$ is a valid element of $T_I\langle\Omega, k\rangle$.

For this special case of an *empty interpretation tree* we use a separate symbol:

$$T_{I_\varepsilon} := \Big(\{\emptyset\}, \emptyset, \emptyset\Big) \in T_I\langle\Omega, k\rangle \tag{4.150}$$

Our previous definition of $T_I\langle\Omega, k\rangle$ allows trees that may contain any number of symbols on leaf-node level. This means, $T_I\langle\Omega, k\rangle$ also includes trees T for which there is at least one $S \in \mathit{leafs}(T)$ where $|S| > 1$. But this is not always desirable, as we will see later on. At the latest when we must process the results of our bottom-up parse we will expect that each leaf node of an interpretation tree includes one and only one symbol (i. e., quasi as "start symbol").

This is why we supplement our previous definitions of trees with the following specialization:

$$\begin{aligned} &T'_I\langle\Omega, k\rangle \subset T_I\langle\Omega, k\rangle : \\ &T'_I\langle\Omega, k\rangle = \left\{ T \;\middle|\; \begin{array}{c} T \in T_I\langle\Omega, k\rangle, \\ \forall S \in \mathit{leafs}(T) : |S| = 1 \end{array} \right\} \cup \{T_{I_\varepsilon}\} \end{aligned} \tag{4.151}$$

A good example for such $T \in T'_I\langle\Omega, k\rangle$ is the tree illustrated in Fig. 4.18.

Since $T'_I\langle\Omega, k\rangle \subset T_I\langle\Omega, k\rangle \subset \mathit{OutTree}\langle P_S\langle\Omega, k\rangle\rangle \subset \mathit{OutTree}\langle Q\rangle$, all operations that can be performed on $\mathit{OutTree}\langle Q\rangle$ can also be applied to $T_I\langle\Omega, k\rangle$. This includes, among others, the previously defined functions (1) *leafs* (Eq. 4.139); (2) *children* (Eq. 4.140); (3) *level* (Eq. 4.141) and (4) *add_child* (Eq. 4.142).

In addition let us introduce two operations which are *exclusively* defined on interpretation trees $T_I\langle\Omega, k\rangle$: (1) *child* (Eq. 4.152) and (2) *leafs* (Alg. 10).

The first of these functions "overloads" *child* from Eq. 4.130:

$$\begin{aligned} \mathit{child} \colon \big(T_I\langle\Omega, k\rangle \times S\langle\Omega, k\rangle \times \mathbb{N}_0\big) &\to \big(S\langle\Omega, k\rangle \cup \{\varepsilon\}\big), \\ (T, s, i) &\mapsto \mathit{child}\big(S_\Sigma(T), s, i\big) \end{aligned} \tag{4.152}$$

Here we set the basic set of symbols, which is required in Eq. 4.128 for resolving symbol ids, to $S_\Sigma(T)$ from Eq. 4.147. This way we specialize $\mathit{child}(S, s, i)$ for usage on interpretation trees and hence the argument list changes from (S, s, i) to (T, s, i). To give some examples, when we take our sample tree T from Fig. 4.18 again, then $\mathit{child}(T, s_3, 1)$ would identify s_1 as the second child symbol of s_3. Examples where we get ε as return value instead, include $\mathit{child}(T, s_5, 99)$ or $\mathit{child}(T, s_2, 0)$.

The second operation on $T_I\langle\Omega, k\rangle$ that we want to introduce is called *leafs* and builds on our previous definition of *child* from Eq. 4.152.

10 *leafs* : $\big(T_I\langle\Omega,k\rangle \times S\langle\Omega,k\rangle\big) \rightharpoonup P_{terminal}\langle\Omega,k\rangle,\ (T,s) \mapsto S,$

```
Require: $s \in S_\Sigma(T)$
1: if $child\_count(s) = 0$ then                          ▷ if $s$ is a leaf node, then ...
2:     return $\{s\}$                                     ▷ ... $s$ should be part of the result set
3: end if
4: $S \leftarrow \emptyset$
5: for $i = 0$ to $\big(child\_count(s) - 1\big)$ do      ▷ apply leafs on all children and ...
6:     $S \leftarrow S \cup leafs\big(T, child(T,s,i)\big)$  ▷ ... collect result in $S \in P_{terminal}\langle\Omega,k\rangle$
7: end for
8: return $S$
```

The recursive algorithm in Alg. 10 accepts an interpretation tree $T \in T_I\langle\Omega,k\rangle$ and a symbol $s \in S_\Sigma(T)$ and detects the set of all terminal symbols that can be found beneath s. In other words, *leafs*(T,s) returns all leaf nodes of the symbol-subtree that starts at root-node s. If s is a leaf node itself, then *leafs*(T,s) returns $\{s\} \in P_{terminal}\langle\Omega,k\rangle$. The leaf nodes beneath s_5 in our previous sample tree T (Fig. 4.18), for example, could be identified by *leafs*$(T,s_5) = \{s_0,s_1,s_2\}$.

With all these definitions at hand we can finally continue with step number two of our three-stage parsing algorithm (Eq. 4.122), that is *parse_list_structures*.

Spatial Parsing Algorithm – Stage 2 – Parsing List Structures

To give you an idea of how the list detection mechanism works, we will go through an example. For the following demonstration we set ...

$$\Omega = ID = \mathbb{N}_0 \text{ and } k = 16$$

This corresponds to the default setting of A_S from Eq. 4.121.

In addition let us assume, that there is some $D \in D\langle\mathbb{N}_0,16\rangle$ given, which describes four rectangular information units with grey fill color and their ids as content. The spatial arrangement of these objects shall approximately correspond to what is illustrated in Fig. 4.19.

Let us also assume, that applying *create_terminals*(D) from Alg. 9 on our given D has generated the following set of terminal symbols:

$$S_0 = \left\{\begin{matrix} s_0, \\ s_1, \\ s_2, \\ s_3 \end{matrix}\right\} = \left\{\begin{matrix} (0, ATOM, b_0, 0, list_\varepsilon), \\ (1, ATOM, b_1, 1, list_\varepsilon), \\ (2, ATOM, b_2, 2, list_\varepsilon), \\ (3, ATOM, b_3, 3, list_\varepsilon) \end{matrix}\right\} \in P_{terminal}\langle\mathbb{N}_0,16\rangle$$

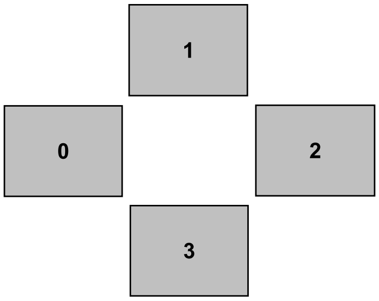


Figure 4.19: Four rectangular information units with light grey fill color and numerical ids as content

For the sake of simplicity, symbolic placeholders s_i as well as assigned attributes $s_i.id$, $s_i.bounds = b_i$ and $s_i.info\_unit$ were numbered sequentially from $i = 0$ to $i = 3$. Bounding volumes b_0 to b_3 approximate the convex hulls of the four rectangles from Fig. 4.19.

Informally expressed, our list detection algorithm *parse_list_structures* will process S_0 as follows:

In a first, preparatory step, we determine for all symbols $s \in S_0$ possible alignments. For this, our algorithm locates for each $s \in S_0$ potential structure-neighbors $N \subset S_0$, that is, objects s', that are close enough to have a spatial relation with s. If there are no objects within the reach of s (i. e., $|N| = 0$), we regard s as "unaligned". But if $|N| \neq 0$, then we determine for all neighbors $s' \in N$ the dominating alignment between s and s'. As defined already in Eq. 4.123, we differentiate between horizontal, vertical and diagonal alignment. Regarding the latter one, we explicitly separate between diagonals from top-left to bottom-right (denoted as "diagonal0") and diagonals from top-right to bottom-left (identified by "diagonal1"). In our example, neighbors of s_0 would be $N = \{s_1, s_3\}$. Object s_2 is located too far away from s_0 and thus may not have a direct spatial relation with s_0. Fig. 4.19 clearly shows that both s_0, s_1 as well as s_0, s_3 are diagonally aligned; once tilted to the right and once to the left. Therefore, s_0 is a potential element of structures with diagonal alignment. The same applies to the remaining symbols in S_0. Due to their relative positioning, also s_1, s_2 and s_3 are membership-candidates for diagonal lists.

In order to keep an overview of which symbols are candidates for what kind of alignment, we collect the results of this pre-analysis in a special data structure, a so-called "AlignmentAccumulator". We are using this term because such structures are effectively used for "accumulating" alignment information. A detailed formal definition can be found on pages 124 to 127.

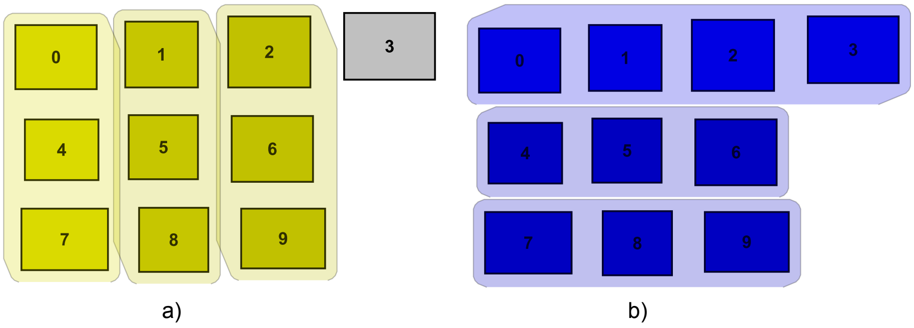


Figure 4.20: Depending on whether we search for vertical lists first and horizontal lists afterwards (snapshot *a*)) or vice versa (snapshot *b*)) you get different parse results

To simplify things we illustrate this alignment structure as a table:

symbol \ alignment	*UA*	*H*	*V*	*D0*	*D1*
s_0	*NC*	*NC*	*NC*	*C*	*C*
s_1	*NC*	*NC*	*NC*	*C*	*C*
s_2	*NC*	*NC*	*NC*	*C*	*C*
s_3	*NC*	*NC*	*NC*	*C*	*C*

This table assigns to each cell $(s, a) \in (S_0 \times \{UA, H, V, D0, D1\})$ an entry out of $\{NC, C, S\}$. The given column headers stand for "UnAligned", "Horizontal", "Vertical", "Diagonal0", and "Diagonal1". Possible cell values indicate whether the symbol associated with the table row is "No Candidate", "Candidate" or a "Structure element". According to our sample table above, all four symbols $\in S_0$ are membership-candidates for structures with *D0*- and *D1*-alignment. Based on this preliminary evaluation, our algorithm tries to detect structures.

At this point it is important to understand, that the order in which one searches for unaligned, horizontal, vertical etc. structures may have significant impact on the parse result. An example for this can be found above in Fig. 4.20: In snapshot *a*) nine out of ten objects are assigned to yellow vertical lists. Thus, there is only one object left which could be used for a subsequent horizontal analysis (here: object number 3). Logically, that object alone cannot form a list and hence remains unassigned. But, if we search for horizontal lists first and for vertical ones afterwards, then we get a very different picture, as illustrated in snapshots *b*). Suddenly all ten objects can be assigned to structures (here: blue horizontal lists).

Apparently, the order in which we perform different structural analyses may have significant influence on the structures that can be detected. Unfortunately, we do not know in general which structure types to prefer over others. This means, there is no universally valid order in which you should detect horizontal, vertical, diagonal etc. lists. This is why our general-purpose parsing algorithm has to consider *all* theoretical possibilities of interpretation.

Applied to our example, these are all permutations $(a_0, \ldots, a_4)$ of table colums $\{\mathit{UA}, H, V, \mathit{D0}, \mathit{D1}\}$, which is a total of $5! = 120$ combinations. Thus, in theory, we have to perform 120 alternative structure detection runs. For performance reasons, however, we decided to set a_0 to a fixed value, that is $a_0 = UA$. Varying four instead of five elements requires to handle only $4! = 24$ permutations, which reduces the workload to twenty percent. Moreover, it makes perfect sense to search for unaligned objects first and for lists afterwards. For these reasons we decided to omit all combinations with $a_i = UA$ for $1 \leq i \leq 4$. Possibly there are even further permutations that can be excluded right from the start, for instance because they would lead to redundant structures. In this version of our algorithm, however, we do not make such assumptions. Rather we have chosen the strategy to consider all theoretical possibilities of interpretation first and to eliminate redundant results afterwards.

Let us start out with $(\mathit{UA}, H, V, \mathit{D0}, \mathit{D1})$, which is the given column order in our sample table. The first three columns *UA*, *H*, and *V* do not include *C*-entries, thus we can skip them. Column *D0*, however, assigns to all four terminal symbols $s_0, \ldots, s_3$ the *C*-flag and hence marks them as potential elements of *D0*-aligned lists. Our algorithm sorts these candidate symbols $\{s_0, s_1, s_2, s_3\}$ in ascending order according to their top-left bounds (i. e., $min_{d0}(s_i.bounds)$; see Eq. 4.79). This results in a sorted list (s_0, s_1, s_3, s_2). The algorithm then sequentially runs through this list from left to right, that is from s_0 to s_2, and tries to combine symbols of the same type (here: *ATOM*) to diagonal lists with a well formed shape. With "well formed" we mean, that we try to form lists with symmetrically arranged elements. This way we intend to resolve ambiguities in list membership. In our example, s_3 is perfectly aligned with s_0. Hence, s_0 and s_3 unambiguously form a diagonal list. The same applies to s_2 and s_1. Thus, all four terminal symbols $s_0, \ldots, s_3$ can be regarded as structure elements.

Having updated our alignment table accordingly, we get:

alignment / symbol	*UA*	*H*	*V*	*D0*	*D1*
s_0	*NC*	*NC*	*NC*	*S*	*C*
s_1	*NC*	*NC*	*NC*	*S*	*C*
s_2	*NC*	*NC*	*NC*	*S*	*C*
s_3	*NC*	*NC*	*NC*	*S*	*C*

Apparently, all terminal symbols $s_0, \ldots, s_3$ are marked with the *S*-flag, which identifies them as structure elements. Hence, there are no symbols left which could be part of a *D1*-aligned list. For this reason, we can skip column *D1* and

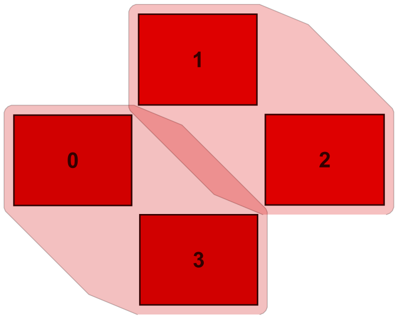


Figure 4.21: sample reduction set S_1

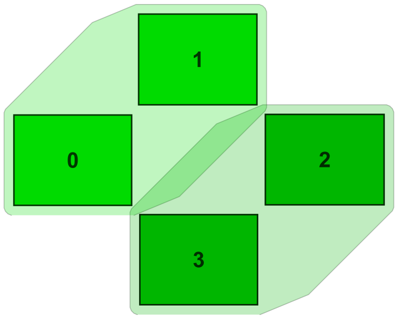


Figure 4.22: sample reduction set S_2

assemble our final set of non-terminals as illustrated in Fig. 4.21. Formally this can be expressed as:

$$S_1 = \left\{ \begin{matrix} \left(4, \mathit{DIAGONAL\_LIST0}, \mathit{union}\left(s_0.\mathit{bounds}, s_3.\mathit{bounds}\right), \varepsilon, (0,3)\right), \\ \left(5, \mathit{DIAGONAL\_LIST0}, \mathit{union}\left(s_1.\mathit{bounds}, s_2.\mathit{bounds}\right), \varepsilon, (1,2)\right) \end{matrix} \right\}$$

For this see our definitions of $S_{nonterminal}\langle\Omega, k\rangle$ from Eq. 4.136 and $S_{struct}\langle\Omega, k\rangle$ in Eq. 4.132.

When we repeat that with a slightly modified permutation $(\mathit{UA}, H, V, \mathit{D1}, \mathit{D0})$, which prefers *D1*-aligned lists over *D0*-aligned lists, we get what is illustrated in Fig. 4.22. Formally the corresponding reduction set S_2 would look as follows:

$$S_2 = \left\{ \begin{matrix} \left(6, \mathit{DIAGONAL\_LIST1}, \mathit{union}\left(s_0.\mathit{bounds}, s_1.\mathit{bounds}\right), \varepsilon, (0,1)\right), \\ \left(7, \mathit{DIAGONAL\_LIST1}, \mathit{union}\left(s_3.\mathit{bounds}, s_2.\mathit{bounds}\right), \varepsilon, (3,2)\right) \end{matrix} \right\}$$

Other permutations than $(\mathit{UA}, H, V, \mathit{D0}, \mathit{D1})$ or $(\mathit{UA}, H, V, \mathit{D1}, \mathit{D0})$ will result either in S_1 or S_2. Therefore they are redundant and can be omitted. See Alg. 13 for a detailed description of the redundancy check.

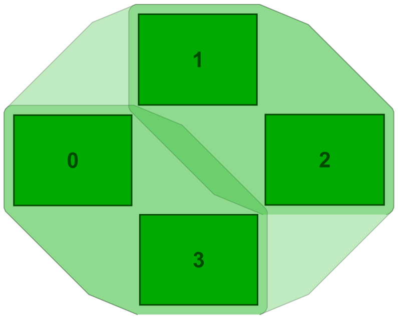


Figure 4.23: sample reduction set S_3

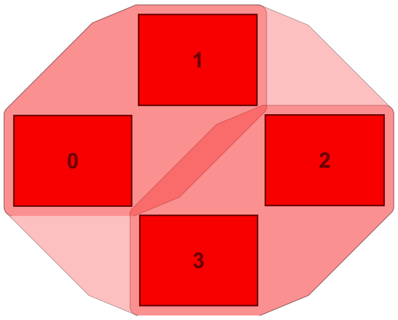


Figure 4.24: sample reduction set S_4

Further structures can be detected on the next abstraction level. Applying the same list detection mechanism on symbols $\in S_1$ will result in …

$$S_3 = \left\{ \left(8, \mathit{DIAGONAL\_LIST1}, \mathit{union} \begin{pmatrix} \mathit{union} \begin{pmatrix} s_0.\mathit{bounds}, \\ s_3.\mathit{bounds} \end{pmatrix}, \\ \mathit{union} \begin{pmatrix} s_1.\mathit{bounds}, \\ s_2.\mathit{bounds} \end{pmatrix} \end{pmatrix}, \varepsilon, (4,5) \right) \right\}$$

… which is illustrated in Fig. 4.23. S_2 can be reduced to …

$$S_4 = \left\{ \left(9, \mathit{DIAGONAL\_LIST0}, \mathit{union} \begin{pmatrix} \mathit{union} \begin{pmatrix} s_0.\mathit{bounds}, \\ s_1.\mathit{bounds} \end{pmatrix}, \\ \mathit{union} \begin{pmatrix} s_3.\mathit{bounds}, \\ s_2.\mathit{bounds} \end{pmatrix} \end{pmatrix}, \varepsilon, (6,7) \right) \right\}$$

… which is illustrated in Fig. 4.24.

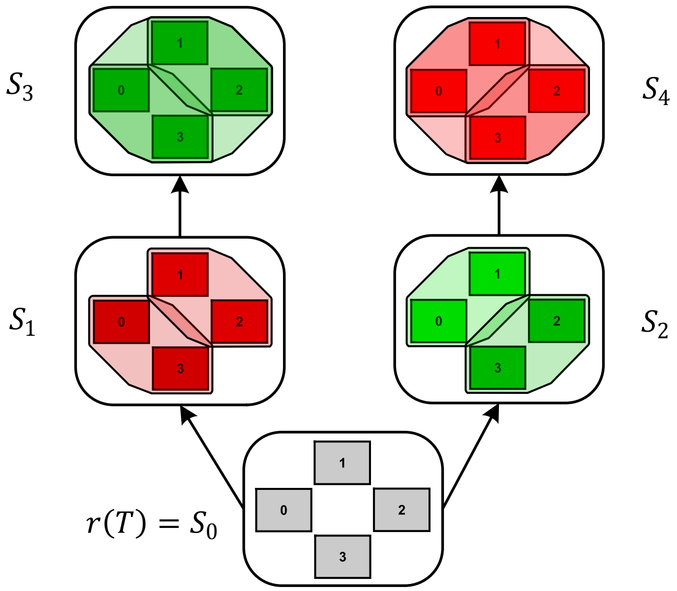


Figure 4.25: sample interpretation tree $T \in T'_I\langle\mathbb{N}_0, 16\rangle$ formed from terminals S_0 and non-terminals S_1, S_2 and S_3, S_4

Together, terminal symbols S_0 and their reduction sets S_1, S_2 and S_3, S_4 form an interpretation tree ...

$$T = \left(\left\{ \begin{matrix} S_3, S_4, \\ S_1, S_2, \\ S_0 \end{matrix} \right\}, \left\{ \begin{matrix} (S_0, S_1), (S_1, S_3), \\ (S_0, S_2), (S_2, S_4) \end{matrix} \right\}, S_0 \right) \in T'_I\langle\mathbb{N}_0, 16\rangle$$

... which is shown in Fig. 4.25.

For this, see also our definitions of $T_I\langle\Omega, k\rangle$ from Eq. 4.143 and its refined subset $T'_I\langle\Omega, k\rangle$ from Eq. 4.151.

This T forms the result of our bottom-up parsing algorithm and thus represents the function value of ...

$$\mathit{parse\_list\_structures}\,(S_0) = T$$

A detailed and complete, formal definition of *parse_list_structures* can be found on the following pages 122 to 138.

Interpretation Tree Expansion *parse_list_structures* (S) follows our original definition of $parse_S$ from Eq. 4.122 and maps collections of terminal symbols $S \in P_{terminal}\langle\Omega, k\rangle$ to interpretation trees $T \in T'_I\langle\Omega, k\rangle$:

$$\begin{aligned} parse\_list\_structures : P_{terminal}\langle\Omega, k\rangle &\to T'_I\langle\Omega, k\rangle, \\ S &\mapsto expand\_tree\Big(\big(\{S\}, \emptyset, S\big), S, 2\Big) \end{aligned} \tag{4.153}$$

Essentially, *parse_list_structures* (S) initiates the recursive expansion of a start-tree $T = (\{S\}, \emptyset, S)$, beginning at root-node $S \in P_{terminal}\langle\Omega, k\rangle$ with a minimal list size of two elements. The resulting tree is guaranteed to be $\in T'_I\langle\Omega, k\rangle$.

In detail we define the recursive expansion of trees $T \in T_I\langle\Omega, k\rangle$ as follows:

11 $expand\_tree :$
$\big(T_I\langle\Omega, k\rangle \times P_S\langle\Omega, k\rangle \times \mathbb{N}_0\big) \rightharpoonup T_I\langle\Omega, k\rangle,\ (T, S, minListSize) \mapsto T',$

```
Require: $\big(S \in leafs(T)\big) \wedge (minListSize \geq 2)$                ▷ see: Alg. 10
1: $T' \leftarrow reduce(T, S, minListSize)$                                  ▷ see: Alg. 12
2: for all $S' \in children\big(T', S\big)$ do                                ▷ see: Eq. 4.140
3:     $T' \leftarrow expand\_tree\big(T', S', minListSize\big)$
4: end for
5: return $T'$
```

The core of $expand\_tree(T, S, minListSize)$, as defined in Alg. 11, are single steps $T' \leftarrow reduce(T, S, minListSize)$ as can be seen in line 1. For reasons of simplicity, we decided to put the formal definition of such expansion steps into a separate algorithm which can be found in Alg. 12.

In $reduce(T, S, minListSize)$ we basically do the following: Firstly, we check whether $|S| > 1$, since for a reduction we need at least two symbols. If that is not the case then T remains unchanged (i. e., $reduce(T, S, minListSize) = T$). Otherwise we continue by setting up our helper data structure “Alignment-Accumulator”, which was mentioned already in our introductory example (on page 117). A complete formal definition of that “table”-structure can be found on pages 124 to 127.

Once we have created and filled such an accumulator-instance we continue with the main structure-detection loop, which basically operates as follows: For each permutation $(a_0, \ldots, a_4)$ of discrete set *Alignment* (see: Eq. 4.154) with $a_0 = UNALIGNED$, we reduce S to a set of non-terminals $S_{reduction}$ and expand the interpretation tree to $T \leftarrow add\_child(T, S, S_{reduction})$. Redundant reduction sets $S_{reduction}$ are rejected. See Alg. 13 for a detailed description of that pairwise redundancy check.

12 *reduce* :
$\big(T_I\langle\Omega,k\rangle \times P_S\langle\Omega,k\rangle \times \mathbb{N}_0\big) \rightharpoonup T_I\langle\Omega,k\rangle,\ (T,S,\mathit{minListSize}) \mapsto T'$,

```
Require: $\big(S \in \mathit{leafs}(T)\big) \wedge (\mathit{minListSize} \geq 2)$    ▷ see: Alg. 10
 1: $T' \leftarrow T$
 2: if $|S| > 1$ then
 3:     $\mathit{accu} \leftarrow \mathit{create\_accumulator}(S)$    ▷ see: Eq. 4.156
 4:     $\mathit{accu} \leftarrow \mathit{fill\_accumulator}(\mathit{accu})$    ▷ see: Alg. 14
 5:     for all permutations $(a_0,\ldots,a_n)$ of $\mathit{Alignment}$,
 6:             where $a_0 = \mathit{UNALIGNED}$ and $n = \big(|\mathit{Alignment}| - 1\big)$ do
 7:         $\begin{pmatrix} S_{reduction}, \\ \mathit{terminate} \end{pmatrix} \leftarrow \mathit{detect\_structures}\begin{pmatrix} S_\Sigma(T'), \\ (a_0,\ldots,a_n), \\ \mathit{accu}, \\ \mathit{minListSize} \end{pmatrix}$    ▷ see: Alg. 21
 8:         if $\mathit{terminate} = \mathit{TRUE}$ then
 9:             $T' \leftarrow \mathit{add\_child}\big(T', S, S_{reduction}\big)$    ▷ see: Eq. 4.142
10:             return $T'$
11:         end if
12:         $\mathit{isRedundant} \leftarrow \mathit{FALSE}$
13:         for all $S' \in \mathit{children}\big(T', S\big)$ do    ▷ see: Eq. 4.140
14:             if $\mathit{redundant}\big(S_{reduction}, S'\big) = \mathit{TRUE}$ then    ▷ see: Alg. 13
15:                 $\mathit{isRedundant} \leftarrow \mathit{TRUE}$
16:                 break
17:             end if
18:         end for
19:         if $\mathit{isRedundant} = \mathit{FALSE}$ then
20:             $T' \leftarrow \mathit{add\_child}\big(T', S, S_{reduction}\big)$    ▷ see: Eq. 4.142
21:         end if
22:     end for
23: end if
24: return $T'$
```

One integral part of Alg. 12, besides *create_accumulator* and *fill_accumulator* in lines 3, 4 and $(S_{reduction}, \mathit{terminate}) \leftarrow \mathit{detect\_structures}(\ldots)$ in line 7, is the redundancy check described from line 12 to line 18.

A given reduction set $S_{reduction}$ is redundant if it is equivalent to at least one reduction set from a previous run through the structure detection loop. That is, if it is equivalent to at least one $S' \in \mathit{children}(T', S)$, with T' being the partly extended interpretation tree T. Two symbol sets $S, S' \in P_S\langle\Omega,k\rangle$ are regarded as being equivalent, if and only if (1) they have the same number of elements (i. e., $|S| = \big|S'\big|$) and (2) for each $s \in S$ there exists exactly one $s' \in S'$ for which $s.\mathit{type} = s'.\mathit{type}$ and $s.\mathit{child\_ids} = s'.\mathit{child\_ids}$.

Reformulated into a pairwise redundancy check for symbol sets (S, S'):

13 $\mathit{redundant}\colon \big(P_S\langle\Omega,k\rangle \times P_S\langle\Omega,k\rangle\big) \to \{\mathit{TRUE}, \mathit{FALSE}\}\,,\ \big(S,S'\big) \mapsto r,$

```
1: if $|S| \neq |S'|$ then
2:     return $FALSE$
3: end if
4: for all $s \in S$ do
5:     $shared \leftarrow FALSE$
6:     for all $s' \in S'$ do
7:         if $(s.type = s'.type) \wedge (s.child\_ids = s'.child\_ids)$ then
8:             $shared \leftarrow TRUE$
9:             break
10:        end if
11:    end for
12:    if $shared = FALSE$ then
13:        return $FALSE$
14:    end if
15: end for
16: return $TRUE$
```

Alignment Accumulator Given the following two discrete sets ...

$$F = \left\{ \begin{array}{c} \mathit{NO\_CANDIDATE}, \\ \mathit{CANDIDATE}, \\ \mathit{STRUCTURE\_ELEMENT} \end{array} \right\} \mathit{Alignment} = \left\{ \begin{array}{c} \mathit{UNALIGNED}, \\ \mathit{HORIZONTAL}, \\ \mathit{VERTICAL}, \\ \mathit{DIAGONAL0}, \\ \mathit{DIAGONAL1} \end{array} \right\} \tag{4.154}$$

... we define the alignment "table", which was mentioned in our introductory example (on page 117), as ...

$$\mathit{AlignmentAccumulator}\langle\Omega,k\rangle = \left\{ (S,f) \;\middle|\; \begin{array}{c} S \in P_S\langle\Omega,k\rangle, \\ f : (S \times \mathit{Alignment}) \to F \end{array} \right\} \tag{4.155}$$

Quite obviously, $\mathit{AlignmentAccumulator}\langle\Omega,k\rangle$ is rather a *map* than simply a *table*. Nevertheless, it may be easier to visualize if you think of it as a two-dimensional table-like structure.

When instantiated, $\mathit{accu} \in \mathit{AlignmentAccumulator}\langle\Omega,k\rangle$ shall neither include *CANDIDATE*- nor *STRUCTURE_ELEMENT*-entries. Consequently, for each "cell" $(s,a) \in (S \times \mathit{Alignment})$ the value of $f(s,a)$ must be *NO_CANDIDATE*.

In order to ensure that, we introduce the following constructor function:

$$create\_accumulator : P_S\langle\Omega, k\rangle \rightarrow AlignmentAccumulator\langle\Omega, k\rangle,\ S \mapsto (S, f)\,, \\ f : (S \times Alignment) \rightarrow \{NO\_CANDIDATE\} \qquad (4.156)$$

A concrete example of usage can be found in Alg. 12 in line 3.

Adequate setter and getter operations for accessing single "cells" in such an alignment "table" are defined in Eq. 4.157 and Eq. 4.158.

$$\begin{aligned} set : &\begin{pmatrix} AlignmentAccumulator\langle\Omega, k\rangle \\ \times\ S\langle\Omega, k\rangle \times Alignment \times F \end{pmatrix} \rightarrow AlignmentAccumulator\langle\Omega, k\rangle, \\ &\big((S, f)\,, s, a, v\big) \mapsto \big(S, f'\big)\,, \\ &f' : (S \times Alignment) \rightarrow F,\ \forall (x, y) \in (S \times Alignment) : \\ &\qquad f'(x, y) = \begin{cases} v \text{ , if } (x = s) \wedge (y = a) \\ f(x, y) \text{ , else} \end{cases} \end{aligned} \qquad (4.157)$$

$$get : \begin{pmatrix} AlignmentAccumulator\langle\Omega, k\rangle \\ \times\ S\langle\Omega, k\rangle \times Alignment \end{pmatrix} \rightharpoonup F,\ \big((S, f)\,, s, a\big) \mapsto f(s, a),\ s \in S \qquad (4.158)$$

In addition to these basic operations on *AlignmentAccumulator*$\langle\Omega, k\rangle$, we introduce three more specific algorithms: (1) *fill_accumulator* (defined in Alg. 14); (2) *structure_candidates* (Alg. 15) and (3) *unassigned_symbols* (which can be found in Alg. 16).

fill_accumulator (*accu*) detects for each $s \in accu.S$ (that is, for all "table rows") the pairwise alignments a of s with its potential neighbors $s' \in accu.S$ and marks the respective "cells" (s, a) in *accu* with the flag *CANDIDATE*. A detailed description of alignment detection can be found in Alg. 17.

14 *fill_accumulator* : $AlignmentAccumulator\langle\Omega, k\rangle \rightarrow AlignmentAccumulator\langle\Omega, k\rangle$, $accu \mapsto accu'$,

```
1: $accu' \leftarrow accu$
2: for all $s \in accu'.S$ do
3:     $A \leftarrow detect\_alignment\big(s, accu'.S\big)$                ▷ see: Alg. 17
4:     for all $a \in A$ do
5:         $accu' \leftarrow set\big(accu', s, a, CANDIDATE\big)$          ▷ see: Eq. 4.157
6:     end for
7: end for
8: return $accu'$
```

Our second operation on $AlignmentAccumulator\langle\Omega, k\rangle$ accepts three arguments: (1) the $accu \in AlignmentAccumulator\langle\Omega, k\rangle$ to operate on; (2) some permutation of alignments $alignOrder = (a_0, \ldots, a_n)$ and (3) a selected $align = a_j$ for any $j \in \{0, 1, \ldots, n\}$.

Given these arguments, $structure\_candidates(accu, alignOrder, align)$ collects all $s \in accu.S$ for which …

$$\begin{gathered} get(accu, s, a_i) \neq STRUCTURE\_ELEMENT (\text{for } 0 \leq i < j) \\ \wedge \\ get\big(accu, s, a_j\big) \neq NO\_CANDIDATE \end{gathered}$$

It herewith identifies all symbols $\in accu.S$ which are candidates for a_j but are not marked as structure elements for preceding alignments $a_0 \ldots a_{j-1}$.

$structure\_candidates:$
15 $\big(AlignmentAccumulator\langle\Omega, k\rangle \times List\langle Alignment\rangle \times Alignment\big) \to P_S\langle\Omega, k\rangle,$
$(accu, alignOrder, align) \mapsto S_{candidates},$

```
1:  $S_{candidates} \leftarrow \emptyset$
2:  for all $s \in accu.S$ do
3:      for $i = 0$ to $\big(size(alignOrder) - 1\big)$ do                ▷ see: Eq. 4.93
4:          $a_i \leftarrow get(alignOrder, i)$                           ▷ see: Eq. 4.94
5:          if $a_i = align$ then
6:              if $get(accu, s, a_i) \neq NO\_CANDIDATE$ then       ▷ see: Eq. 4.158
7:                  $accu \leftarrow set(accu, s, a_i, CANDIDATE)$        ▷ see: Eq. 4.157
8:                  $S_{candidates} \leftarrow S_{candidates} \cup \{s\}$
9:              end if
10:             break
11:         end if
12:         if $get(accu, s, a_i) = STRUCTURE\_ELEMENT$ then    ▷ Eq. 4.158
13:             break
14:         end if
15:     end for
16: end for
17: return $S_{candidates}$
```

Our third and last operation on $AlignmentAccumulator\langle\Omega, k\rangle$ identifies "unassigned" symbols. That is, it collects all symbols $s \in accu.S$ which are not marked as structure elements. To achieve that, $unassigned\_symbols(accu)$ checks for all $s \in accu.S$ if there is at least one $a \in Alignment$ for which $get(accu, s, a) = STRUCTURE\_ELEMENT$ and returns all symbols that do not have such entries. In Alg. 16 we describe that in detail.

16 $\mathit{unassigned\_symbols}:$
$\mathit{AlignmentAccumulator}\langle\Omega,k\rangle \to P_S\langle\Omega,k\rangle,\ \mathit{accu} \mapsto S_{unassigned},$

```
1:  $S_{unassigned} \leftarrow \emptyset$
2:  **for all** $s \in accu.S$ **do**
3:      $isStructElement \leftarrow FALSE$
4:      **for all** $a \in Alignment$ **do**                                ▷ Eq. 4.154
5:          **if** $get(accu, s, a) = STRUCTURE\_ELEMENT$ **then**         ▷ Eq. 4.158
6:              $isStructElement \leftarrow TRUE$
7:              **break**
8:          **end if**
9:      **end for**
10:     **if** $isStructElement = FALSE$ **then**
11:         $S_{unassigned} \leftarrow S_{unassigned} \cup \{s\}$
12:     **end if**
13: **end for**
14: **return** $S_{unassigned}$
```

fill_accumulator, as defined in Alg. 14, requires in line 3 the detection of pairwise alignments of symbols s with their neighbors $s' \in S$. This is realized by $\mathit{detect\_alignment}(s, S)$ using the following procedure: In a first step all spatial neighbors of s are identified. These are all symbols $s' \in S$ that are close enough to have a spatial relation with s. For this we use $N \leftarrow \mathit{lookup\_neighbors}(s, S)$ which is defined in Eq. 4.159. If there are no such objects (i. e., $|N| = 0$) then we regard s as being *UNALIGNED*. Otherwise, we determine for all neighbors $s' \in N$ the dominating pairwise alignment between s and s'. For this we use $\mathit{detect\_alignment}(s, s')$ which is described in detail in Alg. 18 – Alg. 20.

17 $\mathit{detect\_alignment}: \big(S\langle\Omega,k\rangle \times P_S\langle\Omega,k\rangle\big) \to 2^{Alignment},\ (s,S) \mapsto A,$

```
1:  $A \leftarrow \emptyset$
2:  $N \leftarrow lookup\_neighbors(s, S)$                         ▷ see: Eq. 4.159
3:  **if** $|N| = 0$ **then**
4:      $A \leftarrow \{UNALIGNED\}$
5:  **else**
6:      **for all** $s' \in N$ **do**
7:          $(a, \beta) \leftarrow detect\_alignment(s, s')$       ▷ see: Alg. 18 – Alg. 20
8:          **if** $a \neq \varepsilon$ **then**
9:              $A \leftarrow A \cup \{a\}$
10:         **end if**
11:     **end for**
12: **end if**
13: **return** $A$
```

Finding Spatial Neighbors Formally we define the lookup of neighboring parser symbols as ...

$$\begin{aligned}\textit{lookup\_neighbors} : \big(S\langle\Omega,k\rangle \times P_S\langle\Omega,k\rangle\big) &\to P_S\langle\Omega,k\rangle,\\ (s,S) &\mapsto \Big(\textit{lookup}\left(\textit{neighborhood\_bounds}\left(s\right),S\right) \setminus \{s\}\Big)\end{aligned} \tag{4.159}$$

The included $\textit{lookup}\,(\textit{neighborhood\_bounds}\,(s), S)$ makes use of the bounding volume operation $\textit{intersects}\,(b_0, b_1)$ that was introduced alread with Alg. 4:

$$\begin{aligned}\textit{lookup} : \big(\textit{Bounds}\langle k\rangle \times P_S\langle\Omega,k\rangle\big) \to P_S\langle\Omega,k\rangle,\ (b,S) \mapsto S' \subseteq S,\\ \forall s \in S' : \textit{intersects}\,(b, s.\textit{bounds}) = \textit{TRUE}\end{aligned} \tag{4.160}$$

The exact definition of this search operation highly depends on how the parser is implemented. One obvious possibility would be to build on bounding volume hierarchies, such as *QuadTrees.* In fact, this is what we did in our implementation. However, for our theoretical model Eq. 4.160 is completely sufficient.

The search area $b = \textit{neighborhood\_bounds}\,(s)$ that is used for $\textit{lookup}\,(b, S)$ in Eq. 4.159 is calculated by three $\textit{Bounds}\langle k\rangle$-operations: *scale* from Eq. 4.82, $\textit{delta}_{\textit{min}}$ (Alg. 2) and $\textit{delta}_{\textit{max}}$ (Alg. 1):

$$\begin{aligned}&\textit{neighborhood\_bounds} : S\langle\Omega,k\rangle \to \textit{Bounds}\langle k\rangle,\\ &s \mapsto \textit{scale}\left(s.\textit{bounds}, \left(\frac{1}{5} \times \textit{delta}_{\textit{max}}\,(s.\textit{bounds}) + \frac{4}{5} \times \textit{delta}_{\textit{min}}\,(s.\textit{bounds})\right)\right)\end{aligned} \tag{4.161}$$

Note, that the given *offset* ...

$$\left(\frac{1}{5} \times \textit{delta}_{\textit{max}}\,(s.\textit{bounds}) + \frac{4}{5} \times \textit{delta}_{\textit{min}}\,(s.\textit{bounds})\right)$$

... that is used in Eq. 4.161 for scaling *s.bounds*, is crucial for the overall structure recognition procedure. The reason for this is, that the given offset essentially determines size and dimensions of $\textit{neighborhood\_bounds}\,(s)$, which in turn has significant impact on the detection of spatially related objects.

It is quite obvious, that using a fixed offset value, that does not take into account individual object proportions, would too often result in search areas that include the wrong neighbors. With “wrong” we mean, objects which were not intented to have a spatial relation. Consequently, although being the easiest this was not an option for us. At the other extreme, $\textit{neighborhood\_bounds}\,(s)$ that totally depend on the spatial environment of s (i. e., the context) would certainly result in highly accurate selections of neighbors, but their calculation would also be very time consuming. Thus this was not an option either.

With Eq. 4.161 we decided on a middle course between accuracy and efficiency. That is, search areas are dynamically adjusted to object size and proportions but do not require analysis of spatial context. Tests have shown, that our definition of *neighborhood_bounds* (s) allows for reasonable selection of neighbors.

Alignment Detection Pairwise alignment detection between symbols, as required in Alg. 17 (line 7), is realized by *detect_alignment* (Alg. 18 – Alg. 20). *detect_alignment* accepts a pair of parser symbols $s_0, s_1 \in S\langle\Omega, k\rangle$ and tries to determine their dominating (or most apparent) alignment. Results are encoded as tuples (a, β) with $a \in (\mathit{Alignment} \cup \{\varepsilon\})$ and $\beta \in \{0.0, \ldots, 1.0\}$. a represents the detected alignment. $a = \varepsilon$ means that a clear result could not be determined. The second tuple element β is a certainty value (ranging from zero to one-hundred percent) that indicates how obvious alignment a is. Note, that even though $\beta \in \{0.0, \ldots, 1.0\}$ it should not be confused with "Probability".

Given two symbols s_0, s_1, we expect for our pairwise alignment detection, that $s_0.\mathit{bounds}$ and $s_1.\mathit{bounds}$ are not nested inside each other. "Nesting" expresses rather explicit and formal than implicit an informal object relations and thus contradicts spatial hypertext's implicit and informal nature (Sect. 1.2). For this reason, we decided that *detect_alignment* (s_0, s_1) applied on nested symbols shall result in $(\varepsilon, 1.0)$. This way we exclude such relations from our overall structure detection procedure. In all other cases *detect_alignment* (s_0, s_1) continues with some analysis of symmetry. In concrete terms, *detect_alignment* (s_0, s_1) determines numerically how well s_0 and s_1 are aligned to x- and y-axis. This requires the calculation of a numerical *symmetry_indicator* (see Eq. 4.162):

```
18  $\mathit{detect\_alignment}:$
    $\big(S\langle\Omega,k\rangle \times S\langle\Omega,k\rangle\big) \to \Big(\big(\mathit{Alignment} \cup \{\varepsilon\}\big) \times \{0.0,\ldots,1.0\}\Big), (s_0,s_1) \mapsto r,$

1: if $\left(\begin{array}{l} \big(\mathit{contains}(s_0.\mathit{bounds}, s_1.\mathit{bounds}) = \mathit{TRUE}\big) \\ \vee\ \big(\mathit{contains}(s_1.\mathit{bounds}, s_0.\mathit{bounds}) = \mathit{TRUE}\big) \end{array}\right)$ then    ▷ see: Alg. 3
2:     return $(\varepsilon, 1.0)$    ▷ "Nesting" is explicit relation $\Rightarrow \notin \mathit{Alignment}$
3: end if
4: $ah \leftarrow \mathit{symmetry\_indicator}\left(\begin{array}{l} \mathit{min}_v(s_0.\mathit{bounds}), \mathit{max}_v(s_0.\mathit{bounds}), \\ \mathit{min}_v(s_1.\mathit{bounds}), \mathit{max}_v(s_1.\mathit{bounds}) \end{array}\right)$
5: $av \leftarrow \mathit{symmetry\_indicator}\left(\begin{array}{l} \mathit{min}_h(s_0.\mathit{bounds}), \mathit{max}_h(s_0.\mathit{bounds}), \\ \mathit{min}_h(s_1.\mathit{bounds}), \mathit{max}_h(s_1.\mathit{bounds}) \end{array}\right)$
   ...
```

Having calculated degrees of horizontal and vertical alignment ah and av, we can do the same for diagonal alignment $ad0, ad1$. For this we follow two basic principles: (1) "*If it is neither horizontal nor vertical then it must be diagonal*" and (2) "*Two objects can never be aligned from top-left to bottom-right and from top-right to bottom-left at the same time*".

Depending on the orientation of $centroid(s_0.bounds)$ and $centroid(s_1.bounds)$ we either get $ad0 = 0.0$ and $ad1 = (1.0 - |ah - av|)$ or vice versa:

19 $detect\_alignment(s_0, s_1)$ – part II

```
...
6:  $ad0 \leftarrow 0.0$
7:  $ad1 \leftarrow 0.0$
8:  $(x_0, y_0) \leftarrow centroid(s_0.bounds)$                    ▷ see: Alg. 5
9:  $(x_1, y_1) \leftarrow centroid(s_1.bounds)$
10: $\Delta x \leftarrow (x_0 - x_1)$
11: $\Delta y \leftarrow (y_0 - y_1)$
12: if $(\Delta x > 0 \wedge \Delta y < 0) \vee (\Delta x < 0 \wedge \Delta y > 0)$ then
13:     $ad1 \leftarrow \big(1.0 - |ah - av|\big)$
14: else if $(\Delta x > 0 \wedge \Delta y > 0) \vee (\Delta x < 0 \wedge \Delta y < 0)$ then
15:     $ad0 \leftarrow \big(1.0 - |ah - av|\big)$
16: end if
...
```

After that, $ah, av, ad0$, and $ad1$ are normalized to guarantee that they sum up to 1.0. Finally $amax = max(\{ah, av, ad0, ad1\})$ identifies the "winner" alignment. If there are several "winners", then horzontal and vertical alignment are preferred over diagonal alignments:

20 $detect\_alignment(s_0, s_1)$ – part III

```
...
17: $at \leftarrow (ah + av + ad0 + ad1)$                  ▷ make sure that ...
18: $ah \leftarrow \big(ah/at\big)$                 ▷ ... $ah, av, ad0, ad1$ sum up to 1.0
19: $av \leftarrow \big(av/at\big)$
20: $ad0 \leftarrow \big(ad0/at\big)$
21: $ad1 \leftarrow \big(ad1/at\big)$
22: $amax \leftarrow max\big(\{ah, av, ad0, ad1\}\big)$          ▷ detect "winner"-alignment
23: if $amax = ah$ then
24:     return $(HORIZONTAL, amax)$
25: else if $amax = av$ then
26:     return $(VERTICAL, amax)$
27: else if $amax = ad0$ then
28:     return $(DIAGONAL0, amax)$
29: end if
30: return $(DIAGONAL1, amax)$
```

An integral component of *detect_alignment* is the function *symmetry_indicator* (see lines 4 – 5 in Alg. 18). Like our definition of *neighborhood_bounds* in Eq. 4.161, also our definition of *symmetry_indicator* is a trade-off between performance and efficiency, which has resulted from informal studio-tests.

symmetry_indicator maps real-valued quadruples $(min_0, max_0, min_1, max_1)$ to strength values ranging from 0.0 to 1.0. A value of 1.0 indicates that the two sections described by min_0, max_0 and min_1, max_1 are perfectly symmetrical. A return value of 0.0, on the other hand, means that both sections are disjunct (i. e., $(max_1 < min_0) \vee (min_1 > max_0)$). A detailed definition of *symmetry_indicator* is given in Eq. 4.162:

$$\begin{aligned}
&symmetry\_indicator : \mathbb{R}^4 \rightharpoonup \{0.0, \ldots, 1.0\}, \\
&symmetry\_indicator \begin{pmatrix} min_0, \\ max_0, \\ min_1, \\ max_1 \end{pmatrix} = \begin{cases} 0 & \text{, if } \begin{pmatrix} (max_1 < min_0) \\ \vee \ (min_1 > max_0) \end{pmatrix} \\ -\dfrac{2\Delta}{d} + 1 & \text{, else if } \left(\Delta < \frac{1}{4}d\right) \\ -\dfrac{3\Delta}{d} + \dfrac{5}{4} & \text{, else if } \left(\frac{1}{4}d \leq \Delta < \frac{3}{8}d\right) \\ -\dfrac{\Delta}{d} + \dfrac{1}{2} & \text{, else} \end{cases} \\
&\Delta = \left| \frac{min_0 + max_0 - min_1 - max_1}{2} \right| \\
&d = max_0 - min_0 + max_1 - min_1 \\
&(min_0, max_0, min_1, max_1 \in \mathbb{R}), (max_0 > min_0) \wedge (max_1 > min_1) \qquad (4.162)
\end{aligned}$$

Structure Detection The algorithmic "heart" of *parse_list_structures*$(\ldots)$ is defined as ...

$$\begin{pmatrix} S_{reduction}, \\ terminate \end{pmatrix} \leftarrow detect\_structures \begin{pmatrix} S_{\Sigma}(T'), \\ (a_0, \ldots, a_n), \\ accu, \\ minListSize \end{pmatrix}$$

... which can be found in Alg. 12 in line 7.

The algorithm behind this is described formally in Alg. 21 – Alg. 25.

detect_structures$(S, alignOrder, accu, minListSize)$ creates pairs $(S', terminate)$, with S' being a collection of newly generated structure symbols (see: Eq. 4.132) and *terminate* as a boolean control flag $\in \{TRUE, FALSE\}$ which is exclusively used in the context of *reduce* (Alg. 12). In a nutshell, S' is the latest reduction of *accu.S* and the boolean flag *terminate* indicates whether to continue reducing *accu.S* or not. For details on usage of *terminate* see Alg. 12, lines 8 – 21.

More specifically, in *detect_structures* (*S*, *alignOrder*, *accu*, *minListSize*) we run sequentially through $alignOrder = (a_0, \dots, a_n)$ from a_0 to a_n, determine for each a_i the respective *structure_candidates* (*accu*, *alignOrder*, a_i) (see Alg. 15) and try to combine found candidate symbols in disjunct structures with alignment a_i. If $a_i = UNALIGNED$ then we create for each detected candidate a new structure symbol of type *UNALIGNED* and update the accumulator accordingly (i. e., we add *STRUCTURE_ELEMENT*-entries). All unaligned structure symbols generated this way are collected in a set $S_{unaligned}$. This first stage is described in detail in Alg. 21:

```
    detect_structures :

         ⎛  P_S⟨Ω, k⟩                        ⎞
    21   ⎜× List⟨Alignment⟩                  ⎟ ⇀ (P_S⟨Ω, k⟩ × {TRUE, FALSE}) ,
         ⎜× AlignmentAccumulator⟨Ω, k⟩       ⎟
         ⎝× ℕ_0                              ⎠
    (S, alignOrder, accu, minListSize) ↦ (S', terminate) ,

Require: $(accu.S \subseteq S) \wedge (minListSize \geq 2)$
 1: $S_{unaligned} \leftarrow \emptyset$
 2: $S_{list} \leftarrow \emptyset$
 3: for $i = 0$ to $\big(size(alignOrder) - 1\big)$ do                      ▷ Eq. 4.93
 4:     $a_i \leftarrow get(alignOrder, i)$                                  ▷ Eq. 4.94
 5:     $S_{candidates} \leftarrow structure\_candidates(accu, alignOrder, a_i)$   ▷ Alg. 15
 6:     if $a_i = UNALIGNED$ then
 7:         for all $s \in S_{candidates}$ do
 8:             $accu \leftarrow set(accu, s, UNALIGNED, STRUCTURE\_ELEMENT)$
 9:             $s' \leftarrow create\_structure\begin{pmatrix}(S \cup S_{unaligned} \cup S_{list}), \\ UNALIGNED, add(list_\varepsilon, s.id)\end{pmatrix}$   ▷ Alg. 8
10:             $S_{unaligned} \leftarrow S_{unaligned} \cup \{s'\}$
11:         end for
12:     else
    ...
```

If $a_i \neq UNALIGNED$ then we check whether there are enough candidates for assembling lists with at least *minListSize* elements. If this is not the case, then we can skip the current alignment a_i and continue with the next one from *alignOrder* instead. If, however, there are enough candidate symbols then we try to combine them to lists with alignment a_i. For this we use *detect_lists* ($S_{candidates}$, a_i, *minListSize*) which is described in detail in Alg. 26.

For each list $\in$ *detect_lists* $(S_{candidates}, a_i, minListSize)$ we then create a new structure symbol, either of type *HORIZONTAL_LIST*, *VERTICAL_LIST*, *DIAGONAL_LIST0* or *DIAGONAL_LIST1* (this depends on the current a_i) and for each list element we add a *STRUCTURE_ELEMENT*-entry to the accumulator. All list symbols created this way are collected in S_{list}. See Alg. 22 for a complete formal definition:

22 *detect_structures* $(S, alignOrder, accu, minListSize)$ – part II

```
    ...
13:         if $|S_{candidates}| < minListSize$ then
14:             continue
15:         end if
16:         for all $l \in detect\_lists(S_{candidates}, a_i, minListSize)$ do          ▷ Alg. 26
17:             $elementIds \leftarrow list_\varepsilon$
18:             for $j = 0$ to $\big(size(l) - 1\big)$ do                                ▷ Eq. 4.93
19:                 $s_j \leftarrow get(l, j)$                                            ▷ Eq. 4.94
20:                 $accu \leftarrow set\big(accu, s_j, a_i, STRUCTURE\_ELEMENT\big)$
21:                 $elementIds \leftarrow add\big(elementIds, s_j.id\big)$              ▷ Eq. 4.92
22:             end for
23:             $listType \leftarrow HORIZONTAL\_LIST$
24:             if $a_i = VERTICAL$ then
25:                 $listType \leftarrow VERTICAL\_LIST$
26:             else if $a_i = DIAGONAL0$ then
27:                 $listType \leftarrow DIAGONAL\_LIST0$
28:             else if $a_i = DIAGONAL1$ then
29:                 $listType \leftarrow DIAGONAL\_LIST1$
30:             end if
31:             $s' \leftarrow create\_structure\left(\begin{matrix}\big(S \cup S_{unaligned} \cup S_{list}\big),\\ listType, elementIds\end{matrix}\right)$   ▷ Alg. 8
32:             $S_{list} \leftarrow S_{list} \cup \{s'\}$
33:         end for
34:     end if
35: end for
    ...
```

Having detected at least one list (i. e., $|S_{list}| > 0$) we copy all remaining symbols into our reduction set, that is $S_{result} = S_{unaligned} \cup S_{list}$. Thus, they can be reused in another structure detection run. For this, we lookup all remaining symbols from *accu.S* which were not marked as structure elements (see: *unassigned_symbols* $(accu)$ from Alg. 16). These symbols are then “cloned” and added to S_{result} which is finally returned in form of $(S_{result}, FALSE)$. With $terminate = FALSE$ we signalize to continue reducing *accu.S* for other *alignOrder*s. The exact definition can be found in Alg. 23.

23 *detect_structures* $(S, alignOrder, accu, minListSize)$ – part III

```
    ...
36: if $|S_{list}| > 0$ then
37:     $S_{result} \leftarrow S_{unaligned} \cup S_{list}$
38:     for all $s \in unassigned\_symbols(accu)$ do                ▷ Alg. 16
39:         $s' \leftarrow create\_symbol\big((S \cup S_{result}), s.type, s.bounds, \varepsilon, add(list_\varepsilon, s.id)\big)$
40:         $S_{result} \leftarrow S_{result} \cup \{s'\}$
41:     end for
42:     return $(S_{result}, FALSE)$
43: end if
    ...
```

Note, however, that this does not automatically mean that we quit our bottom-up procedure when we cannot form lists (i. e., when $|S_{list}| = 0$). We rather continue reducing unaligned and unassigned symbols until only a single start-symbol is left. This way, we still get valid parse trees, even though there are no lists. This requires adequate reduction rules as those in Alg. 24 and Alg. 25.

Alg. 24 handles the case of $\left|unassigned\_symbols(accu)\right| = \left|S_{unaligned}\right| = 1$. In this crucial case we manually perform a "dummy"-reduction to ε. That means, we reduce both unaligned and unassigned symbol to a single start-symbol of type ε and signalize $terminate = TRUE$:

24 *detect_structures* $(S, alignOrder, accu, minListSize)$ – part IV

```
    ...
44: if $\Big(\left|S_{unaligned}\right| = 1\Big) \wedge \Big(\left|unassigned\_symbols(accu)\right| = 1\Big)$ then   ▷ Alg. 16
45:     $ids \leftarrow list_\varepsilon$
46:     for all $s \in S_{unaligned}$ do
47:         $ids \leftarrow add\big(ids, get(s.child\_ids, 0)\big)$       ▷ Eq. 4.92, Eq. 4.94
48:     end for
49:     for all $s \in unassigned\_symbols(accu)$ do                ▷ Alg. 16
50:         $ids \leftarrow add(ids, s.id)$                          ▷ Eq. 4.92
51:     end for
52:     return $\Big(\big\{create\_structure(S, \varepsilon, ids)\big\}, TRUE\Big)$      ▷ Alg. 8
53: end if
    ...
```

In all remaining cases we manually reduce unaligned symbols to a single parent symbol of type *UNALIGNED* and unassigned symbols to a single non-terminal with type ε. You could also say that we group unaligned symbols together under the label *UNALIGNED* and unassigned symbols under ε. Both are combined in S_{result} and are finally returned together with $terminate = TRUE$. See Alg. 25.

25 *detect_structures* $(S, alignOrder, accu, minListSize)$ – part V

```
    ...
54: $S_{result} \leftarrow \emptyset$
55: **if** $\left|S_{unaligned}\right| > 0$ **then**
56:     $ids \leftarrow list_{\varepsilon}$
57:     **for all** $s \in S_{unaligned}$ **do**
58:         $ids \leftarrow add\left(ids, get(s.child\_ids, 0)\right)$          ▷ Eq. 4.92, Eq. 4.94
59:     **end for**
60:     $S_{result} \leftarrow S_{result} \cup \left\{create\_structure(S, UNALIGNED, ids)\right\}$     ▷ Alg. 8
61: **end if**
62: **if** $\left|unassigned\_symbols(accu)\right| > 0$ **then**          ▷ Alg. 16
63:     $ids \leftarrow list_{\varepsilon}$
64:     **for all** $s \in unassigned\_symbols(accu)$ **do**          ▷ Alg. 16
65:         $ids \leftarrow add(ids, s.id)$          ▷ Eq. 4.92
66:     **end for**
67:     $S_{result} \leftarrow S_{result} \cup \left\{create\_structure\left((S \cup S_{result}), \varepsilon, ids\right)\right\}$     ▷ Alg. 8
68: **end if**
69: **return** $(S_{result}, TRUE)$
```

List Detection The core list detection algorithm that is described in Alg. 26 requires an adequate data structure for representing lists of symbols. For this we use the following derivative of $\mathit{List}\langle S\langle\Omega,k\rangle\rangle$ (see Eq. 4.91 for a generic definition of lists):

$$\mathit{SymbolList}\langle\Omega,k\rangle \subset \mathit{List}\langle S\langle\Omega,k\rangle\rangle :$$

$$\mathit{SymbolList}\langle\Omega,k\rangle = \left\{ \mathit{list} \;\middle|\; \begin{array}{c} \mathit{list} = (s_0, s_1, \ldots, s_n) \in \big(\mathit{List}\langle S\langle\Omega,k\rangle\rangle \setminus \{\mathit{list}_\varepsilon\}\big)\,, \\ \nexists\,\{s_i, s_j\} : \big(s_i \neq s_j\big) \wedge \big(s_i.\mathit{id} = s_j.\mathit{id}\big)\,, \\ 0 \leq i < j \leq n, \\ n \in \mathbb{N}_0 \end{array} \right\} \cup \{\mathit{list}_\varepsilon\} \quad (4.163)$$

What makes such lists $(s_0, s_1, \ldots, s_n) \in \mathit{SymbolList}\langle\Omega,k\rangle$ special is not only the fact that they include only symbols $s \in S\langle\Omega,k\rangle$ as list elements, but also that there may be no pair of symbols s_i, s_j for which $s_i.\mathit{id} = s_j.\mathit{id}$ but $s_i \neq s_j$. In a nutshell, list elements with same ids must also have the same attributes.

Subsets of $\mathit{SymbolList}\langle\Omega,k\rangle$ are used as return values for our list detection algorithm. *detect_lists*, as defined in Alg. 26, accepts a triple of arguments $(S, a, \mathit{minListSize})$, with potential list elements $S \in P_S\langle\Omega,k\rangle$, the required list alignment $a \in (\mathit{Alignment} \setminus \{\mathit{UNALIGNED}\})$ and $\mathit{minListSize} \in \mathbb{N}_0$. These three arguments are mapped to a (potentially empty) set of lists $L \subset \mathit{SymbolList}\langle\Omega,k\rangle$. Here it shall be ensured, that all detected lists $\in L$ are not empty and include at least *minListSize* elements. In order to achieve that, $\mathit{detect\_lists}(S, a, \mathit{minListSize})$ basically proceeds as follows:

In a first step all symbols $s \in S$ get sorted in ascending order and according to their lower horizontal, vertical or diagonal bounds. This depends on the given alignment argument a. See Eq. 4.164 for a detailed description of this sort operation.

In a second step, we run sequentially through that sorted list $(s_0, s_1, \ldots, s_n)$ from s_0 to s_n and check for each s_i which list $\in L$ fits best. In concrete terms, given that s_{last} denotes the last element of a list candidate, we filter out those lists from L for which, (1) s_i and s_{last} have the same type (i. e., $s_{last}.\mathit{type} = s_i.\mathit{type}$); (2) s_i and s_{last} are close enough to have a spatial relation (i. e., $\mathit{intersects}(\mathit{neighborhood\_bounds}(s_{last}), s_i.\mathit{bounds})$) (see: Alg. 4, Eq. 4.161) and (3) the pairwise alignment a' between s_i and s_{last} that is determined by $(a', \beta) \leftarrow \mathit{detect\_alignment}(s_{last}, s_i)$ fits the required alignment given by argument a (i. e., $a' = a$).

From those list candidates that fulfill these three conditions we then select the one with a maximal β. This "winner"-list gets extended by s_i. When no "winner" can be found (for instance, because this was our first run and hence L was still empty) then we extend L by a new list which includes only s_i.

Having repeated that for each s_i from our sorted list $(s_0, s_1, \ldots, s_n)$, we finally remove all list candidates from L which include less than *minListSize* elements. A detailed formal description is given in Alg. 26.

```
     $detect\_lists :$
26   $\Big(P_S\langle\Omega,k\rangle \times \big(Alignment \setminus \{UNALIGNED\}\big) \times \mathbb{N}_0\Big) \rightarrow 2^{SymbolList\langle\Omega,k\rangle},$
     $(S, a, minListSize) \mapsto L,$

Ensure: $\forall l \in L : \big(empty(l) = FALSE\big) \wedge \big(size(l) \geq minListSize\big)$
 1: $L \leftarrow \emptyset$
 2: $sortedSymbols \leftarrow sort(S, a)$                                        ▷ Eq. 4.164
 3: for $i = 0$ to $\big(size(sortedSymbols) - 1\big)$ do                          ▷ Eq. 4.93
 4:     $s_i \leftarrow get(sortedSymbols, i)$                                  ▷ Eq. 4.94
 5:     $\gamma \leftarrow 0.0$
 6:     $list_{winner} \leftarrow \varepsilon$
 7:     for all $list_{candidate} \in L$ do
 8:         $s_{last} \leftarrow last(list_{candidate})$                         ▷ Eq. 4.97
 9:         if $s_{last}.type = s_i.type$ then
10:             if $intersects\big(neighborhood\_bounds(s_{last}), s_i.bounds\big)$ then
11:                 $\big(a', \beta\big) \leftarrow detect\_alignment(s_{last}, s_i)$          ▷ Alg. 18
12:                 if $\big(a' = a\big) \wedge (\beta > \gamma)$ then
13:                     $\gamma \leftarrow \beta$
14:                     $list_{winner} \leftarrow list_{candidate}$
15:                 end if
16:             end if
17:         end if
18:     end for
19:     if $list_{winner} \neq \varepsilon$ then
20:         $L \leftarrow \Big(L \cup \big\{add(list_{winner}, s_i)\big\}\Big) \setminus \{list_{winner}\}$       ▷ Eq. 4.92
21:     else
22:         $L \leftarrow L \cup \big\{add(list_{\varepsilon}, s_i)\big\}$                       ▷ Eq. 4.92
23:     end if
24: end for
25: for all $list_{candidate} \in L$ do
26:     if $size(list_{candidate}) < minListSize$ then                          ▷ Eq. 4.93
27:         $L \leftarrow L \setminus \{list_{candidate}\}$
28:     end if
29: end for
30: return $L$
```

In line 2 Alg. 26 requires sorting of symbols by $sortedSymbols \leftarrow sort(S, a)$. $sort(S, a)$ is supposed to sort S in ascending order using the comparator function $\mathit{compare}(s_0, s_1, a)$ which is described in Alg. 27. The exact definition of *sort* highly depends on the selected sorting algorithm and hence on concrete system implementation. For our theoretical model, however, it does not matter what algorithm we choose. Therefore it is sufficient to define *sort* as follows:

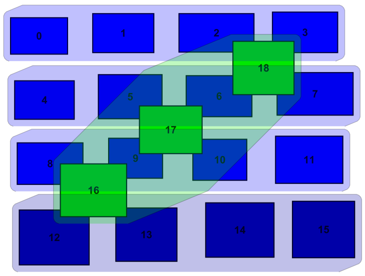
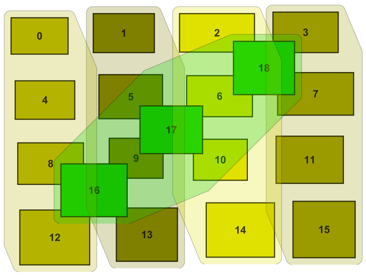
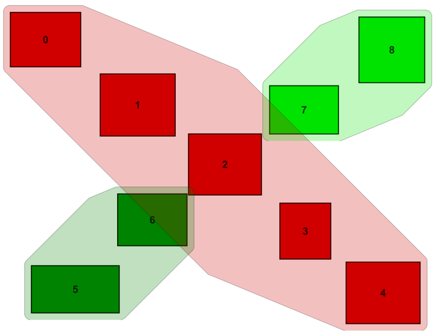
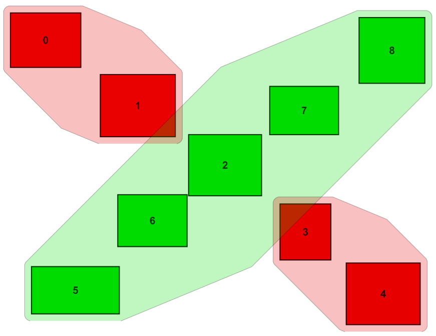
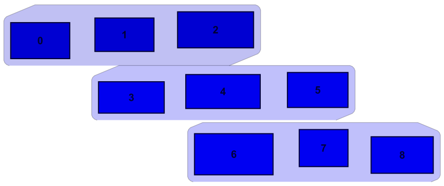
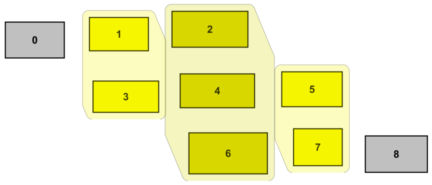
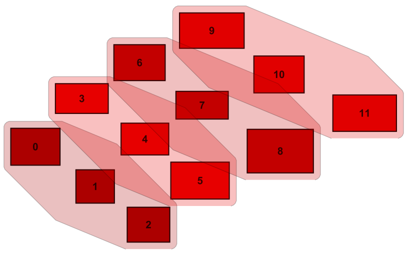
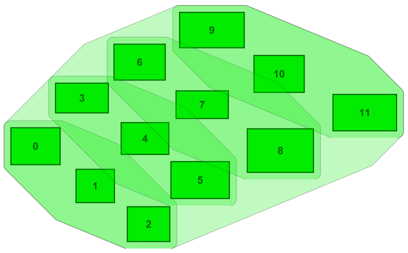
$$\begin{aligned}
\mathit{sort} \colon & \Big(P_S\langle\Omega,k\rangle \times \big(\mathit{Alignment} \setminus \{\mathit{UNALIGNED}\}\big)\Big) \to \mathit{SymbolList}\langle\Omega,k\rangle, \\
& (S,a) \mapsto l : \\
& \qquad \big(\mathit{size}(l) = |S|\big) \wedge \left(\bigcup_{i=0}^{\mathit{size}(l)-1} \{\mathit{get}(l,i)\}\right) = S \\
& \qquad \mathit{compare}\big(\mathit{get}(l,i), \mathit{get}(l,i+1), a\big) \in \{\mathit{LESS}, \mathit{EQUAL}\}, \\
& \qquad\qquad i \in \Big\{0, 1, \ldots, \big(\mathit{size}(l) - 2\big)\Big\}
\end{aligned} \tag{4.164}$$

Depending on the given alignment a, comparator function $\mathit{compare}(s_0, s_1, a)$ selects for s_0, s_1 lower horizontal, vertical or diagonal bounds $\mathit{min}_0, \mathit{min}_1$ and checks whether min_0 is *LESS* than, *GREATER* than or *EQUAL* to min_1:

27 $\mathit{compare} : \begin{pmatrix} S\langle\Omega,k\rangle \times S\langle\Omega,k\rangle \\ \times \big(\mathit{Alignment} \setminus \{\mathit{UNALIGNED}\}\big) \end{pmatrix} \to \begin{Bmatrix} \mathit{LESS}, \\ \mathit{EQUAL}, \\ \mathit{GREATER} \end{Bmatrix}, (s_0, s_1, a) \mapsto r,$

```
1: $r \leftarrow \mathit{EQUAL}$
2: if $a = \mathit{HORIZONTAL}$ then
3:     $\mathit{min}_0 \leftarrow \mathit{min}_h(s_0.\mathit{bounds})$        ▷ Eq. 4.77
4:     $\mathit{min}_1 \leftarrow \mathit{min}_h(s_1.\mathit{bounds})$
5: else if $a = \mathit{VERTICAL}$ then
6:     $\mathit{min}_0 \leftarrow \mathit{min}_v(s_0.\mathit{bounds})$        ▷ Eq. 4.78
7:     $\mathit{min}_1 \leftarrow \mathit{min}_v(s_1.\mathit{bounds})$
8: else if $a = \mathit{DIAGONAL0}$ then
9:     $\mathit{min}_0 \leftarrow \mathit{min}_{d0}(s_0.\mathit{bounds})$       ▷ Eq. 4.79
10:    $\mathit{min}_1 \leftarrow \mathit{min}_{d0}(s_1.\mathit{bounds})$
11: else if $a = \mathit{DIAGONAL1}$ then
12:    $\mathit{min}_0 \leftarrow \mathit{min}_{d1}(s_0.\mathit{bounds})$       ▷ Eq. 4.80
13:    $\mathit{min}_1 \leftarrow \mathit{min}_{d1}(s_1.\mathit{bounds})$
14: end if
15: if $\mathit{min}_0 < \mathit{min}_1$ then
16:    $r \leftarrow \mathit{LESS}$
17: else if $\mathit{min}_0 > \mathit{min}_1$ then
18:    $r \leftarrow \mathit{GREATER}$
19: end if
20: return $r$
```

The following snapshots from diverse interpretation trees will give you an impression of the power of the previously described list detection mechanism. (Note, that these snapshots were chosen randomly)

Spatial Parsing Algorithm – Stage 3 – Normalization

The last step in our three-stage spatial parsing algorithm, besides creating terminal symbols (Alg. 9) and detecting list structures (Eq. 4.153), is normalizing interpretation trees returned by *parse_list_structures* (*create_terminals* (...)). For this we define ...

$$\textit{normalize} : T'_I\langle\Omega, k\rangle \to I\langle\Omega\rangle$$

normalize (T) is supposed to transform ingoing interpretation trees $T \in T'_I\langle\Omega, k\rangle$ into weighted, "flat" graphs $\in I\langle\Omega\rangle$, as we defined them in Eq. 4.90. This mapping builds on the following considerations:

First, let us define simple Paths from root to leaf in interpretation trees $T'_I\langle\Omega, k\rangle$ as elements of the following generic set:

$$T_P\langle\Omega, k\rangle = \left\{ P \;\middle|\; \begin{array}{c} P \in T'_I\langle\Omega, k\rangle, \\ \forall v \in V(P) : d^+_P(v) \in \{0, 1\} \end{array} \right\} \subset T'_I\langle\Omega, k\rangle \tag{4.165}$$

Thus, (simple) root-to-leaf paths in interpretation trees are interpretation trees themselves.

In addition to operations that are defined on any $T \in T'_I\langle\Omega, k\rangle$, such as *leafs* (Eq. 4.139), *children* (Eq. 4.140), *level* (Eq. 4.141), ... etc., we introduce two additional operations for navigating paths $P \in T_P\langle\Omega, k\rangle$: (1) *has_next* (Eq. 4.166) and (2) *next* (Eq. 4.167).

has_next (P, S) takes a path $P \in T_P\langle\Omega, k\rangle$ and a vertex $S \in P_S\langle\Omega, k\rangle$ and checks whether S has a successor on P (i. e., if *children* $(P, S) \neq \emptyset$). If this is the case then *TRUE* is returned. Otherwise *has_next* (P, S) = *FALSE*. This also includes the case that $S \notin V(P)$. Thus, *has_next* (P, S) is totally defined:

$$\begin{aligned} \textit{has\_next} &: \big(T_P\langle\Omega, k\rangle \times P_S\langle\Omega, k\rangle\big) \to \{\textit{TRUE}, \textit{FALSE}\}, \\ \textit{has\_next}(P, S) &= \begin{cases} \textit{TRUE}, & \text{if } \textit{children}(P, S) \neq \emptyset \\ \textit{FALSE}, & \text{else} \end{cases} \end{aligned} \tag{4.166}$$

The second required operation for iterating through paths $P \in T_P\langle\Omega, k\rangle$ is a single jump from one path-node to its successor. For this we use the function *next*. *next* (P, S) accepts a path $P \in T_P\langle\Omega, k\rangle$ and a vertex $S \in P_S\langle\Omega, k\rangle$ and returns the child (or rather the successor) of S in P. This is only defined if *children* $(P, S) \neq \emptyset$. Thus, for navigating P we need both operations *has_next* and *next*.

$$\begin{aligned} \textit{next} &: \big(T_P\langle\Omega, k\rangle \times P_S\langle\Omega, k\rangle\big) \rightharpoonup P_S\langle\Omega, k\rangle, \\ &(P, S) \mapsto S' \in \textit{children}(P, S), \ \textit{children}(P, S) \neq \emptyset \end{aligned} \tag{4.167}$$

Additionally, let $P_T(T)$ be a function, that can split trees $T \in T'_I\langle\Omega, k\rangle$ up into collections of root-to-leaf paths $\{P_0, P_1, \ldots, P_n\}$, where $n = \left|\mathit{leafs}(T)\right| - 1$:

$$P_T : T'_I\langle\Omega, k\rangle \to 2^{T_P\langle\Omega,k\rangle},\ T \mapsto \left\{P_0, P_1, \ldots, P_{\left(\left|\mathit{leafs}(T)\right|-1\right)}\right\},$$
$$\left(\bigcup_{i=0}^{\left(\left|\mathit{leafs}(T)\right|-1\right)} V(P_i)\right) = V(T) \wedge \left(\bigcup_{i=0}^{\left(\left|\mathit{leafs}(T)\right|-1\right)} E(P_i)\right) = E(T) \tag{4.168}$$

From *parse_list_structures* (Eq. 4.153) we can conclude, that each ...

$$\{P_0, P_1, \ldots, P_n\} \in P_T\left(T'_I\langle\Omega, k\rangle\right)$$

... encodes a set of *alternative* interpretations ...

$$\{I_0, I_1, \ldots, I_n\} \in 2^{(I\langle\Omega\rangle)}\ ;\ (n \in \mathbb{N}_0)$$

Thus, there is a total mapping ...

$$P_T\left(T'_I\langle\Omega, k\rangle\right) \to 2^{\left(I\langle\Omega\rangle\right)},\ P_T(T) \mapsto I_P\left(P_T(T)\right)$$

... where function I_P is defined as:

$$I_P : T_P\langle\Omega, k\rangle \to I\langle\Omega\rangle,\ P \mapsto I = (U, A, w) \tag{4.169}$$

Here, U shall be the set of all information unit identifiers *s.info_unit* that can be found in the root node of P. A is the set of all binary subsets of U:

$$U = \left(\bigcup_{s \in r(P)} \{s.\mathit{info\_unit}\}\right)\ ;\ A = \binom{U}{2} \tag{4.170}$$

The total weighting function w is defined as a binary relation:

$$w : A \to \{\varepsilon, 0.0, \ldots, 1.0\},\ w = R_w\left(P, r(P)\right) \tag{4.171}$$

For this we introduce the recursive function $R_w(P, S)$:

$$R_w(P, S) = \begin{cases} \begin{pmatrix} R'_w(P, S) \\ \cup\, R_w\left(P, \mathit{next}(P, S)\right) \end{pmatrix}, & \text{if } \mathit{has\_next}(P, S) = \mathit{TRUE} \\ R'_w(P, S), & \text{else} \end{cases} \tag{4.172}$$

Illustrated with an example, if P is a simple path from root node $r(P) = S_0$ to leaf node S_n then $R_w\left(P, r(P)\right)$ from Eq. 4.172 would result in:

$$R_w\big(P, r(P)\big) = R'_w\left(P, S_0\right) \cup R'_w\left(P, S_1\right) \cup \ldots \cup R'_w\left(P, S_n\right)$$

Thus, $R_w\left(P, r(P)\right)$ simply reapplies $R'_w\left(P, S\right)$ on all nodes from root to leaf and calculates their union. The result is a binary relation that can be used as weighting function w, as defined in Eq. 4.171.

In this context, it shall be noted that ...

$$\left(\bigcap_{S \in V(P)} R'_w(P, S)\right) = \emptyset;\ P \in T_P\langle\Omega, k\rangle$$

$R'_w\left(P, S\right)$ takes each $s \in S$ and calculates for all pairwise combinations of child symbols $\{\mathit{child}(P, s, i), \mathit{child}(P, s, j)\}$ the relation $R''_w(P, S, s, i, j)$; which is defined in Eq. 4.174. The union of all these binary relations forms the function value of $R'_w\left(P, S\right)$:

$$R'_w(P, S) = \bigcup_{s \in S} \left(\bigcup_{i=0}^{\left(\mathit{child\_count}(s)-2\right)} \ \bigcup_{j=i+1}^{\left(\mathit{child\_count}(s)-1\right)} R''_w(P, S, s, i, j) \right) \tag{4.173}$$

The relation $R''_w\left(P, S, s, i, j\right)$ assigns values $\mathit{weight}(s.\mathit{type}, \mathit{level}(P, S), i, j)$ to unordered pairs $\{a.\mathit{info\_unit}, b.\mathit{info\_unit}\}$. Here, a is required to be a leaf node beneath $\mathit{child}(P, s, i)$ and b shall be a leaf node under $\mathit{child}(P, s, j)$:

$$R''_w(P, S, s, i, j) = \left\{ \left(\left\{ \begin{array}{c} a.\mathit{info\_unit}, \\ b.\mathit{info\_unit} \end{array} \right\}, \mathit{weight}\left(\begin{array}{c} s.\mathit{type}, \\ \mathit{level}(P, S), \\ i, j \end{array} \right) \right) \middle| \begin{array}{c} a \in \mathit{leafs}\big(P, \mathit{child}(P, s, i)\big) \\ \wedge \\ b \in \mathit{leafs}\big(P, \mathit{child}(P, s, j)\big) \end{array} \right\} \tag{4.174}$$

Due to Eq. 4.171, $\mathit{weight}(s.\mathit{type}, \mathit{level}(P, S), i, j)$, as we use it in Eq. 4.174, has to provide function values $\in \{\varepsilon, 0.0, \ldots, 1.0\}$. Only then $R''_w\left(P, S, s, i, j\right)$, $R'_w(P, S)$ and finally also $R_w\left(P, S\right)$ form valid weighting relations. Parameters that must be included in this calculation are: (1) the structure type $s.\mathit{type}$ of a parent symbol s; (2) the (structure-)level of node S in P (provided that $s \in S$) and (3) child symbol indices $i, j \in \mathbb{N}_0$.

In more detail, function *weight* looks as follows:

$$weight \colon \Big(\big(StructureType \cup \{\varepsilon\} \big) \times \mathbb{N}_0 \times \mathbb{N}_0 \times \mathbb{N}_0 \Big) \rightharpoonup \{\varepsilon, 0.0, \ldots, 1.0\},$$

$$\begin{pmatrix} structType, \\ structLevel, \\ i, j \end{pmatrix} \mapsto v = \begin{cases} \begin{pmatrix} weight_{type}\,(structType) \\ \times\, weight_{level}\,(structLevel) \\ \times\, weight_{distance}\,(i, j) \end{pmatrix}, & \text{if } structType \neq \varepsilon \\ \varepsilon, & \text{else} \end{cases} \tag{4.175}$$

Provided that *structType* $\neq \varepsilon$ we define $weight(structType, structLevel, i, j)$ as a product of three factors: (1) $weight_{type}$ (Eq. 4.176); (2) $weight_{level}$ (Eq. 4.177) and (3) $weight_{distance}$ (Eq. 4.178).

$weight_{type}$ maps structure types $\in StructureType$ to weightings $\in \{0.0, \ldots, 1.0\}$. It herewith represents an indicator for the general strength of object-relations in structures of certain types (such as horizontal, vertical lists etc.). Since function *weight* from Eq. 4.175 and thus also $weight_{type}$ is never used on root-node-level, we do not need to provide a definition of $weight_{type}\,(ATOM)$. Therefore $weight_{type}$ is partially defined:

$$weight_{type} : StructureType \rightharpoonup \{0.0, \ldots, 1.0\},$$

$$weight_{type} = \left\{ \begin{array}{r} (UNALIGNED, 0.0), \\ (HORIZONTAL\_LIST, 1.0), \\ (VERTICAL\_LIST, 1.0), \\ (DIAGONAL\_LIST0, 1.0), \\ (DIAGONAL\_LIST1, 1.0) \end{array} \right\} \tag{4.176}$$

In Eq. 4.176 we do not make any specific assumptions about the different strengths of object-relationships in horizontal, vertical or diagonal lists. The alignment of a list has no specific impact on the dependencies between its elements. Therefore $weight_{type}$ of a list is always 1.0. For structure type *UNALIGNED* we use a weight of 0.0 instead, since unaligned objects are certainly not associated with each other.

The second weighting factor we use for quantifying the strength of object relations is $weight_{level}$. Clearly it makes a significant difference whether two objects have a direct spatial relation or not. It makes a difference whether two objects are direct members of the same list or they rather lie in spatially separated structures and are related on a higher level of abstraction only. Objects with a direct spatial relation must have a stronger dependency than objects that are only associated indirectly. This gets reflected by the following formula:

$$\begin{aligned} \mathit{weight}_{\mathit{level}} : \mathbb{N}_0 \rightharpoonup \{0.0, \ldots, 1.0\}\,, \forall x \in \mathbb{N}^+ : \\ \mathit{weight}_{\mathit{level}}(x) = \frac{1}{2^{(x-1)}} \end{aligned} \tag{4.177}$$

According to this definition, interpretation tree levels (or rather structure-levels) $x = 1, 2, 3, \ldots, n$ get weighted as ...

$$\mathit{weight}_{\mathit{level}}(x) = 1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots, \frac{1}{2^{(n-1)}}$$

As we are working with ordered sequences of objects (i. e., lists), the strength of object relations is not only determined by structure *type* (Eq. 4.176) and structure *level* (Eq. 4.177) but also by the *distance* between objects inside a sorted structure. It is quite obvious, that consecutive list elements have a stronger spatial relation than list elements that are spatially separated by a sub-list. This is taken into consideration by the following weighting function:

$$\begin{aligned} \mathit{weight}_{\mathit{distance}} : (\mathbb{N}_0 \times \mathbb{N}_0) \rightharpoonup \{0.0, \ldots, 1.0\}\,, \\ (i, j) \mapsto 2^{(i-j+1)},\ j > i \end{aligned} \tag{4.178}$$

Here, arguments i and j represent positive indices $\in \mathbb{N}_0$ of two objects in a sequentially ordered structure, where $j > i$. Using the formula above, two consecutive structure elements (i. e., $j - i = 1$) get a weighting of 1.0, for $j - i = 2$ we get 0.5, for $j - i = 3$ the weight will be 0.25 and so forth.

With all these definitions at hand it finally becomes possible to retrieve implicitly encoded interpretations $\{I_0, I_1, \ldots, I_n\} \subset I\langle\Omega\rangle$ from interpretation trees $T \in T'_I\langle\Omega, k\rangle$. The only thing that remains to be done in order to realize the desired function *normalize* : $T'_I\langle\Omega, k\rangle \rightarrow I\langle\Omega\rangle$ is merging $I_0, I_1, \ldots, I_n$ together to a single result interpretation $\in I\langle\Omega\rangle$. Since we cannot know in general which of these interpretations to prefer we have rather decided on a maximal weighting strategy than mixing $I_0, I_1, \ldots, I_n$ in specific proportions. A desirable side-effect of taking over only maximal weightings into the final interpretation graph is that smaller structures dissolve in bigger ones.

Based on these considerations we define *normalize* as:

$$\begin{aligned} \mathit{normalize} : T'_I\langle\Omega, k\rangle \rightarrow I\langle\Omega\rangle,\ T \mapsto I = (U, A, w), \\ U = \left(\bigcup_{s \in r(T)} \{s.\mathit{info\_unit}\} \right) \\ A = \binom{U}{2} \\ w = \mathit{collect\_weights}\big(T, r(T)\big) \end{aligned} \tag{4.179}$$

collect_weights (T, S) generalizes $R_w(P, S)$ from Eq. 4.172 for usage on full interpretation trees $T \in T_I'\langle\Omega, k\rangle$. This is achieved by selecting only maximal weightings for the final interpretation graph (see: *max_weight* from Eq. 4.180).

28 *collect_weights* :

$$\big(T_I'\langle\Omega,k\rangle \times P_S\langle\Omega,k\rangle\big) \rightharpoonup 2^{\left(\binom{\Omega}{2}\times\{\varepsilon,0.0,\ldots,1.0\}\right)},\ (T,S)\mapsto R_w$$

```
Require: $S \in V(T)$
 1: $R \leftarrow \emptyset$
 2: for all $S' \in children(T, S)$ do                                    ▷ Eq. 4.140
 3:     if $R = \emptyset$ then
 4:         $R \leftarrow collect\_weights\big(T, S'\big)$
 5:     else
 6:         $R_{merge} \leftarrow \emptyset$
 7:         for all $(a, v) \in collect\_weights\big(T, S'\big)$ do
 8:             $R_{merge} \leftarrow R_{merge} \cup \Big\{\Big(a, max\_weight\big(R(a), v\big)\Big)\Big\}$      ▷ Eq. 4.180
 9:         end for
10:         $R \leftarrow R_{merge}$
11:     end if
12: end for
13: $R' \leftarrow \emptyset$
14: for all $s \in S$ do
15:     for $i = 0$ to $\big(child\_count(s) - 2\big)$ do                   ▷ Eq. 4.129
16:         for $j = i + 1$ to $\big(child\_count(s) - 1\big)$ do
17:             $v \leftarrow \varepsilon$
18:             if $s.type = UNALIGNED$ then
19:                 $v \leftarrow 0.0$
20:             else if $s.type \neq \varepsilon$ then
21:                 $v \leftarrow 2^{\left(i-j-level(T,S)+2\right)}$                ▷ Eq. 4.141
22:             end if
23:             for all $s_0 \in leafs\big(T, child(T, s, i)\big)$ do    ▷ Alg. 10, Eq. 4.152
24:                 for all $s_1 \in leafs\big(T, child(T, s, j)\big)$ do
25:                     $R' \leftarrow R' \cup \left\{\left(\left\{\begin{matrix} s_0.info\_unit, \\ s_1.info\_unit \end{matrix}\right\}, v\right)\right\}$
26:                 end for
27:             end for
28:         end for
29:     end for
30: end for
31: return $R \cup R'$
```

$$
\begin{aligned}
\mathit{max\_weight} \colon \big(\{\varepsilon, 0.0, \ldots, 1.0\} \times \{\varepsilon, 0.0, \ldots, 1.0\}\big) &\to \{\varepsilon, 0.0, \ldots, 1.0\}\,, \\
(w_0, w_1) \mapsto w_{max} = &\begin{cases} \varepsilon & \text{, if } (w_0 = \varepsilon) \wedge (w_1 = \varepsilon) \\ w_0 & \text{, else if } (w_0 \neq \varepsilon) \wedge (w_1 = \varepsilon) \\ w_1 & \text{, else if } (w_0 = \varepsilon) \wedge (w_1 \neq \varepsilon) \\ \mathit{max}(w_0, w_1) & \text{, else} \end{cases}
\end{aligned}
\tag{4.180}
$$

For a better understanding, let us finally apply *normalize* on our introductory list-detection example from pages 115 – 121.

The resulting interpretation tree we got on page 121 was defined as ...

$$
T = \left(\left\{ \begin{matrix} S_3, S_4, \\ S_1, S_2, \\ S_0 \end{matrix} \right\}, \left\{ \begin{matrix} (S_0, S_1)\,, (S_1, S_3)\,, \\ (S_0, S_2)\,, (S_2, S_4) \end{matrix} \right\}, S_0 \right) \in T'_I\langle \mathbb{N}_0, 16 \rangle
$$

... where the root set of terminal symbols S_0 was given as ...

$$
S_0 = \left\{ \begin{matrix} (0, \mathit{ATOM}, b_0, 0, \mathit{list}_\varepsilon)\,, \\ (1, \mathit{ATOM}, b_1, 1, \mathit{list}_\varepsilon)\,, \\ (2, \mathit{ATOM}, b_2, 2, \mathit{list}_\varepsilon)\,, \\ (3, \mathit{ATOM}, b_3, 3, \mathit{list}_\varepsilon) \end{matrix} \right\} \in P_{terminal}\langle \mathbb{N}_0, 16 \rangle
$$

When we calculate now *normalize*(T), using our definitions from Eq. 4.179, then we get a triple $(U, A, w) \in I\langle \Omega \rangle$ with ...

$$
U = \left(\bigcup_{s \in S_0} \{s.\mathit{info\_unit}\} \right) = \{0, 1, 2, 3\}
$$

$$
A = \binom{U}{2} = \left\{ \begin{matrix} \{0,1\}\,, \{0,2\}\,, \{0,3\}\,, \\ \{1,2\}\,, \{1,3\}\,, \\ \{2,3\} \end{matrix} \right\}
$$

... and a weighting function w which is calculated by ...

$$
w = \mathit{collect\_weights}(T, S_0)
$$

Essentially, *collect_weights*(T, S_0), as we defined it in Alg. 28, generates two binary relations, which are . . .

$$collect\_weights\,(T, S_1) = \left\{ \begin{array}{l} (\{0,3\}, 1.0)\,, \\ (\{1,2\}, 1.0) \end{array} \right\} \cup collect\_weights\,(T, S_3)$$

$$= \left\{ \begin{array}{l} (\{0,3\}, 1.0)\,, \\ (\{1,2\}, 1.0) \end{array} \right\} \cup \left\{ \begin{array}{l} (\{0,1\}, 0.5)\,, \\ (\{0,2\}, 0.5)\,, \\ (\{3,1\}, 0.5)\,, \\ (\{3,2\}, 0.5) \end{array} \right\} = \left\{ \begin{array}{l} (\{0,1\}, 0.5)\,, \\ (\{0,2\}, 0.5)\,, \\ (\{3,1\}, 0.5)\,, \\ (\{3,2\}, 0.5)\,, \\ (\{0,3\}, 1.0)\,, \\ (\{1,2\}, 1.0) \end{array} \right\}$$

. . . and . . .

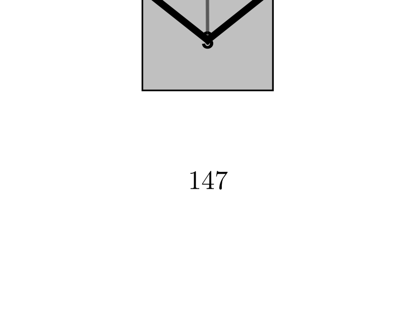
$$collect\_weights\,(T, S_2) = \left\{ \begin{array}{l} (\{0,1\}, 1.0)\,, \\ (\{3,2\}, 1.0) \end{array} \right\} \cup collect\_weights\,(T, S_4)$$

$$= \left\{ \begin{array}{l} (\{0,1\}, 1.0)\,, \\ (\{3,2\}, 1.0) \end{array} \right\} \cup \left\{ \begin{array}{l} (\{0,3\}, 0.5)\,, \\ (\{0,2\}, 0.5)\,, \\ (\{1,3\}, 0.5)\,, \\ (\{1,2\}, 0.5) \end{array} \right\} = \left\{ \begin{array}{l} (\{0,3\}, 0.5)\,, \\ (\{0,2\}, 0.5)\,, \\ (\{1,3\}, 0.5)\,, \\ (\{1,2\}, 0.5)\,, \\ (\{0,1\}, 1.0)\,, \\ (\{3,2\}, 1.0) \end{array} \right\}$$

Both alternatives are merged together choosing maximal weightings:

$$collect\_weights\,(T, S_0) = \left\{ \begin{array}{l} (\{0,3\}, max\_weight\,(1.0, 0.5))\,, \\ (\{0,2\}, max\_weight\,(0.5, 0.5))\,, \\ (\{1,3\}, max\_weight\,(0.5, 0.5))\,, \\ (\{1,2\}, max\_weight\,(1.0, 0.5))\,, \\ (\{0,1\}, max\_weight\,(0.5, 1.0))\,, \\ (\{3,2\}, max\_weight\,(0.5, 1.0)) \end{array} \right\} = \left\{ \begin{array}{l} (\{0,3\}, 1.0)\,, \\ (\{0,2\}, 0.5)\,, \\ (\{1,3\}, 0.5)\,, \\ (\{1,2\}, 1.0)\,, \\ (\{0,1\}, 1.0)\,, \\ (\{3,2\}, 1.0) \end{array} \right\}$$

Illustrated as an overlay on Fig. 4.19, where weightings are drawn as black lines with varying brightness, thickness and opacity:



4.4.3 Visual Parser

Dynamic parser systems, as we defined them in Sect. 4.4.1 (Eq. 4.106), analyze the strength of pairwise object relations and generate output in form of weighted graphs $\in I\langle\Omega\rangle$. To achieve that parsers focus on different structural aspects. A good example for this could be seen in previous Sect. 4.4.2 where we discussed a spatial parsing algorithm. However, spatial dependency is not the only crucial factor for successful structure detection. As pointed out already in Sect. 1.4 (on page 8) visual structure is not only determined by position, alignment etc. but also by visual (dis-)similarities. Parsers that take this into account in their analysis are called "visual parsers".

Let Φ_V be a set of event categories that indicate when to perform a full visual parse. For this see our definitions of *parse* in Eq. 4.110 and Φ in Eq. 4.107.

$$\Phi_V = \left\{ \begin{array}{c} \mathit{CREATE}, \\ \mathit{MODIFY\_SPATIAL}, \\ \mathit{MODIFY\_VISUAL}, \\ \mathit{DELETE} \end{array} \right\} \tag{4.181}$$

In direct comparison, Φ_V extends Φ_S from Eq. 4.119 by an additional flag *MODIFY_VISUAL*. Thus, a full reparse is triggered not only when new information units were created or deleted or when spatial properties got modified (e. g., shape, proportions, dimensions etc.) but also when purely visual attributes have changed, such as color.

Provided that our visual parsing algorithm is defined by a function $\mathit{parse}_V : (I\langle\Omega\rangle \times E\langle\Omega\rangle \times D\langle\Omega, k\rangle) \to I\langle\Omega\rangle$, we can use Φ_V from Eq. 4.181 to partially refine our generic parser model from Eq. 4.106 into a parameterized model for visual parsers $A_V\langle\Omega, n, \alpha, \beta\rangle$:

$$\begin{aligned} A_V\langle\Omega, n, \alpha, \beta\rangle &:= A_P\langle\Omega, n, \alpha, \beta, \Phi_V, \mathit{parse}_V\rangle \\ T_V\langle\Omega, n, \alpha, \beta\rangle &:= T_P\langle\Omega, n, \alpha, \beta, \Phi_V, \mathit{parse}_V\rangle \\ G_V\langle\Omega, n, \alpha, \beta\rangle &:= G_P\langle\Omega, n, \alpha, \beta, \Phi_V, \mathit{parse}_V\rangle \end{aligned} \tag{4.182}$$

When using the same default configuration as given in Eq. 4.121 we get:

$$\begin{aligned} A_V &:= A_V\langle\mathbb{N}_0, 16, 0.95, 0.05\rangle \\ T_V &:= T_V\langle\mathbb{N}_0, 16, 0.95, 0.05\rangle \\ G_V &:= G_V\langle\mathbb{N}_0, 16, 0.95, 0.05\rangle \end{aligned} \tag{4.183}$$

Following the blockdiagram in Fig. 4.15 visual parsers A_V can be illustrated as seen in Fig. 4.26.

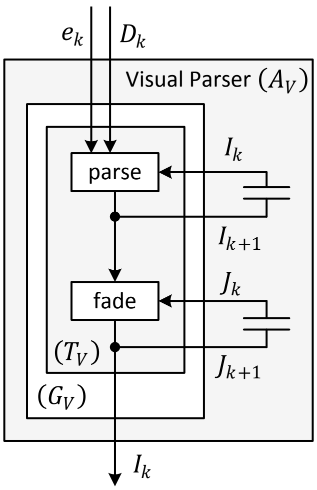


Figure 4.26: Blockdiagram of visual parser model A_V as defined in Eq. 4.183

$parse_V$ follows the definition of φ from Eq. 4.107 and maps triples (I, e, D) to interpretation graphs I'. Here, $I, I' \in I\langle\Omega\rangle$, $e \in E\langle\Omega\rangle$ and $D \in D\langle\Omega, k\rangle$.

Unlike the previously defined spatial parser (see: Eq. 4.122) our visual parser includes all three arguments I, e and D into the structure analysis process. That is, $parse_V$ utilizes not only spatial and visual properties D, but also the preceding edit event e and the latest visual parse result I:

$$\begin{aligned} parse_V : \big(I\langle\Omega\rangle \times E\langle\Omega\rangle \times D\langle\Omega, k\rangle\big) &\to I\langle\Omega\rangle, \\ \big((U, A, w), e, D\big) &\mapsto \big(U', A', w'\big) \end{aligned} \tag{4.184}$$

When an ingoing event e indicates that new information units $e.objects$ were added to the workspace (i. e., $e.operation = CREATE$) then U has to be extended to $U' = (U \cup e.objects)$. If, however, $e.objects$ have been removed (i. e., $e.operation = DELETE$) then U must be reduced to $U' = (U \setminus e.objects)$. In all other cases, such as $MODIFY\_SPATIAL$, $MODIFY\_VISUAL$ etc., U remains unchanged, that is $U' = U$. Expressed as piecewise function:

$$U' = \begin{cases} U \cup e.objects \text{ , if } e.operation = CREATE \\ U \setminus e.objects \text{ , if } e.operation = DELETE \\ U \text{ , else} \end{cases} \tag{4.185}$$

Both cases $e.operation = CREATE$ and $e.operation = DELETE$ require not only to adjust U, as described in Eq. 4.185, but also to update A, which builds on U. For this we use the following rules: If $e.operation = CREATE$ then the previous A must be extended by new associations $\{u, u'\} \notin A$ for which either $u, u' \in e.objects$ or $u \in U \wedge u' \in e.objects$ holds. In the opposite case, that is $e.operation = DELETE$, we remove all $\{u, u'\}$ from A where u or u' are not included in U' anymore, which makes them redundant. Otherwise $A' = A$:

$$A' = \begin{cases} A \cup \binom{e.objects}{2} \cup \left\{ \{u, u'\} \;\middle|\; \begin{matrix} u \in U \\ \wedge \\ u' \in e.objects \end{matrix} \right\} & \text{, if } e.operation = CREATE \\ \binom{U'}{2} & \text{, if } e.operation = DELETE \\ A & \text{, else} \end{cases} \tag{4.186}$$

Note, that in Eq. 4.185 and Eq. 4.186 we check only whether $e.operation = CREATE$ or $e.operation = DELETE$. Flags, such as *MODIFY_SPATIAL*, *MODIFY_VISUAL* etc., are not included explicitely. The advantage of this is, that we can easily extend $E\langle\Omega\rangle$ from Eq. 4.71 by additional (and possibly more specific) operation flags without the need to adjust our visual parsing algorithm. This is especially beneficial in application development.

The last element in result triples $(U', A', w') \in I\langle\Omega\rangle$, besides U' (Eq. 4.185) and A' (Eq. 4.186), is the weighting function w':

$$\begin{aligned} & w' : A' \rightarrow \{\varepsilon, 0.0, \ldots, 1.0\}, \forall a \in A' : \\ & \quad w'(a) = \begin{cases} detect\_visual\_relation(a, D) & \text{, if } a \in A_{recalc} \\ w(a) & \text{, else} \end{cases} \end{aligned} \tag{4.187}$$

(Re-)calculation of weights $w'(a) = detect\_visual\_relation(a, D)$ is only required for associations $a \in A_{recalc}$. These are defined as follows:

$$A_{recalc} = \binom{e.objects}{2} \cup \left\{ \{u, u'\} \;\middle|\; \begin{matrix} u \in e.objects \\ \wedge \\ u' \in \big(U \setminus e.objects\big) \end{matrix} \right\} \tag{4.188}$$

According to this, (re-)calculation happens only for (1) associations with at least one *modified* object and (2) for associations which are completely *new*. In case of *DELETE*, however, weights $w(a)$ do not need to be updated, since for no remaining $a \in A' : a \in A_{recalc}$. Therefore $\forall a \in A' : w'(a) = w(a)$; which results in $w' \subset w$.

Note that if e is an empty event (i. e., $e = (\emptyset, \varepsilon) \in E\langle\Omega\rangle$; see Eq. 4.71), then we set $U' = U$, $A' = A$ and thus also $w' = w$. Consequently, $parse_V(I, (\emptyset, \varepsilon), D)$ returns an unchanged $I = (U, A, w)$. When combined with fading of interpretations (see Sect. 4.4.1; pages 93–94) this allows to simulate passing of discrete time and hence continuous aging of structure; even *without* active involvement of human users. That is, theoretically empty events $e = (\emptyset, \varepsilon)$ could be used as fixed system clock. Currently we do not make use of this option.

The most crucial part of our visual parsing algorithm is detection of visual object relations. In the current version of our algorithm the strength of visual relations equates with the degree of visual similarity. This means, how strong the visual relation between two objects is, is only determined by how optically similar they are (regarding color, shape, proportions etc.). Other notions of visual relation are not considered in our visual parser.

In order to detect the strength of visual connections between information units, $\mathit{detect\_visual\_relation}\,(\{u,u'\},(V,p))$ takes two units $u,u' \in \Omega$ and a set of properties $(V,p) \in D\langle\Omega,k\rangle$ and calculates $\mathit{detect\_visual\_similarity}\,(p(u),p(u'))$. This requires that $u,u' \in V$. Otherwise ε is returned:

$$\begin{aligned}
&\mathit{detect\_visual\_relation} : \left(\binom{\Omega}{2} \times D\langle\Omega,k\rangle\right) \to \{\varepsilon, 0.0, \ldots, 1.0\}\,, \\
&\Big(\{u,u'\}\,,(V,p)\Big) \mapsto y = \begin{cases} \mathit{detect\_visual\_similarity}\begin{pmatrix} p(u), \\ p\left(u'\right) \end{pmatrix} & \text{, if } u,u' \in V \\ \varepsilon & \text{, else} \end{cases}
\end{aligned} \tag{4.189}$$

We express the degree of visual similarity between two information units as numerical values $\in \{0.0, \ldots, 1.0\}$. A value of 1.0 indicates that both objects are optically identical, whereas 0.0 means that they have nothing in common visually. Informal laboratory tests resulted in the following weighted formula:

$$\begin{aligned}
\mathit{detect\_visual\_similarity} &: \big(\mathit{InfoUnitData}\langle k\rangle\big)^2 \to \{0.0, \ldots, 1.0\}\,, \\
(d_0, d_1) &\mapsto \left(\frac{\alpha \times w_p \times w_d + \beta \times w_s + \gamma \times w_c}{\alpha + \beta + \gamma}\right), \\
&\alpha = 8,\ \beta = 2,\ \gamma = 4
\end{aligned} \tag{4.190}$$

w_p is a numerical indicator for how similar proportions of $d_0.\mathit{bounds}$ and $d_1.\mathit{bounds}$ are. Here, proportions are defined by *width* and *height* from Eq. 4.81.

$$w_p = \mathit{proportion\_similarity}\begin{pmatrix} \mathit{width}\,(d_0.\mathit{bounds})\,, \\ \mathit{height}\,(d_0.\mathit{bounds})\,, \\ \mathit{width}\,(d_1.\mathit{bounds})\,, \\ \mathit{height}\,(d_1.\mathit{bounds}) \end{pmatrix} \tag{4.191}$$

The respective formula is:

$$proportion\_similarity\left(w_0, h_0, w_1, h_1\right) = 1 - \frac{|w_0 h_1 - w_1 h_0|}{w_0 h_1 + w_1 h_0} \tag{4.192}$$

As an example, when two objects have exactly the same proportions and hence $w_0 = w_1 = w$ and $h_0 = h_1 = h$ then $proportion\_similarity\left(w, h, w, h\right)$ gets:

$$\left(1 - \frac{|wh - wh|}{wh + wh}\right) = \left(1 - \frac{0}{2wh}\right) = (1 - 0) = 1.0$$

In a similar fashion we determine w_d which is a numerical indicator for how similar two objects are regarding their dimensions (or sizes):

$$w_d = dimension\_similarity\begin{pmatrix} width\left(d_0.bounds\right), \\ height\left(d_0.bounds\right), \\ width\left(d_1.bounds\right), \\ height\left(d_1.bounds\right) \end{pmatrix} \tag{4.193}$$

For this calculation we approximate object dimensions by rectangular surface areas *width* × *height* (Eq. 4.81). Hence, the respective formula looks slightly different than the one in Eq. 4.192:

$$dimension\_similarity\left(w_0, h_0, w_1, h_1\right) = 1 - \frac{|w_0 h_0 - w_1 h_1|}{w_0 h_0 + w_1 h_1} \tag{4.194}$$

Illustrated with an example, when two objects have the same approximated surface areas (i. e., $w_0 h_0 = w_1 h_1$) then $dimension\_similarity\left(w_0, h_0, w_1, h_1\right)$ will result in:

$$\left(1 - \frac{|w_0 h_0 - w_1 h_1|}{w_0 h_0 + w_1 h_1}\right) = \left(1 - \frac{0}{w_0 h_0 + w_1 h_1}\right) = (1 - 0) = 1.0$$

Another factor that can play an essential role for calculating visual similarities is the similarity of geometrical shapes w_s:

$$w_s = shape\_similarity\left(d_0.shape, d_1.shape\right) \tag{4.195}$$

As only rectangular and ellipsoidal shapes are used (see: Eq. 4.84) we restrict our definition of *shape_similarity* on equality checks for shapes. In concrete terms, if two shapes $s_0, s_1 \in Shape$ are equal then $shape\_similarity\left(s_0, s_1\right)$ returns a similarity value of 1.0. Otherwise $shape\_similarity\left(s_0, s_1\right) = 0.0$:

$$shape\_similarity : \left(Shape \times Shape\right) \rightarrow \left\{0.0, \ldots, 1.0\right\}, \forall s_0, s_1 \in Shape :$$

$$shape\_similarity\left(s_0, s_1\right) = \begin{cases} 1 \text{ , if } s_0 = s_1 \\ 0 \text{ , else} \end{cases} \tag{4.196}$$

The last factor included in *detect_visual_similarity* (Eq. 4.190), besides w_p (Eq. 4.191), w_d(Eq. 4.193) and w_s(Eq. 4.195), is w_c, where $\underline{c}$ stands for $\underline{c}$olor:

$$w_c = \mathit{color\_similarity}\,(d_0.\mathit{color}, d_1.\mathit{color}) \tag{4.197}$$

After having conducted several informal experiments with different color spaces and weighted formulas we finally developed the following piecewise definition of *color_similarity*:

$$\begin{gathered}\mathit{color\_similarity} : (\mathit{Color} \times \mathit{Color}) \to \{0.0, \ldots, 1.0\}\,, \forall c_0, c_1 \in \mathit{Color} : \\ \mathit{color\_similarity}\,(c_0, c_1) = \\ \begin{cases} 1 \,, & \text{if } 0 \leq \Delta E\,(c_0, c_1) < 5 \\ \left(-\frac{1}{95} \times \Delta E\,(c_0, c_1) + \frac{100}{95}\right) \,, & \text{if } 5 \leq \Delta E\,(c_0, c_1) < 100 \\ 0 \,, & \text{else} \end{cases}\end{gathered} \tag{4.198}$$

$\Delta E\,(c_0, c_1)$, as we use it in Eq. 4.198, is defined as the Euclidean distance between two colors $\mathit{Lab}\,(c_0)\,, \mathit{Lab}\,(c_1)$ in multidimensional L*a*b* color space [38]. L*a*b* colors were designed to approximate human vision and are therefore perfectly suited for detecting perceivable color similarities. Since there are no simple formulas for conversion between *Color* (i. e., RGB) and L*a*b* we do not provide a full definition of *Lab* here. We rather limit ourselves to ...

$$\begin{gathered}\Delta E\,(c_0, c_1) = \left\|\mathit{Lab}\,(c_0) - \mathit{Lab}\,(c_1)\right\|_2 \\ \mathit{Lab} : \mathit{Color} \to \begin{pmatrix} \{0, \ldots, 100\} \\ \times\ \{-170, \ldots, 100\} \\ \times\ \{-100, \ldots, 150\} \end{pmatrix}, \ (r, g, b) \mapsto (L^*, a^*, b^*)\end{gathered} \tag{4.199}$$

As an example, let us assume that there are two RGB-colors given:

$$c_0 = (255, 255, 255) \ \text{ and } \ c_1 = (0, 0, 0)$$

Transformed into L*a*b* c_0 becomes $\mathit{Lab}\,(c_0) = (100, 0, 0)$ and c_1 turns into $\mathit{Lab}\,(c_1) = (0, 0, 0)$. The Euclidean distance between $\mathit{Lab}\,(c_0)$ and $\mathit{Lab}\,(c_1)$ is ...

$$\Delta E\,(c_0, c_1) = \left\|(100, 0, 0) - (0, 0, 0)\right\|_2 = 100$$

$\Delta E = 100$ falls in section number three in Eq. 4.198. Thus the *color_similarity* of *black* and *white* is 0.0. For our purposes this makes perfect sense.

Finally, let us illustrate the effects of combining these similarity values in an example. Here we follow our definition from Eq. 4.190.

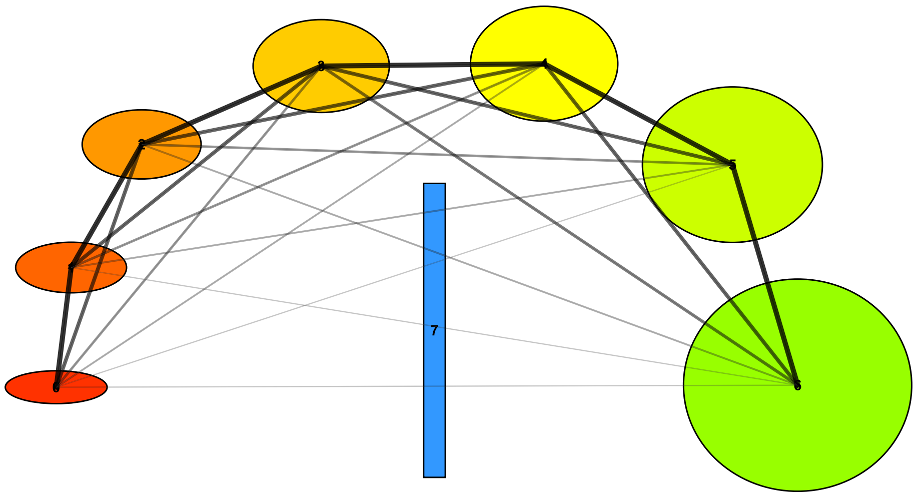


Figure 4.27: Visual parse result illustrated as a network of objects with varying size, proportions, shape and fill color connected by black lines with varying brightness, thickness and opacity. The more similar two objects are the thicker the connecting line and the stronger the line color.

Let us assume that eight visual objects are given, seven of type *ELLIPSE* and one of type *RECTANGLE*. Elliptical objects shall be arranged in a bow shape gradually morphing from left to right from small, red, horizontal ellipses to big green circles. In contrast to this, object number eight shall be a narrow upright rectangle with light blue fill color. The visual interpretation graph we would get back by applying $parse_V$ on these objects is illustrated in Fig. 4.27.

In Fig. 4.27 weightings are drawn as black lines with varying brightness, thickness and opacity. The bigger the weight the thicker the line and the stronger the color. Weights of 1.0, for instance, would be drawn as opaque black bars. Smaller weights are illustrated as thin lines with weak color. This has the optical effect of vanishing edges when weights converge towards zero. This also explains why there are no visible connections between the blue rectangle in the middle and the elliptical objects arranged around: Our visual parser recognizes that rectangle and ellipses have nothing in common visually.

4.4.4 Content Parser

Normally spatial parsers do not deal with content of information units. They rather infer object relations from spatial and visual properties than from the (non-)verbal information included in hypertext nodes (Sect. 1.3). This, however, does not necessarily mean that it makes no sense or that there is no need for evaluating an information unit's payload. In fact, this was considered already in one of the first publications on spatial parsing, in [12]. But it has never been implemented.

Even though analysing contentual relationships contradicts the basic operating principle of spatial parsers, it still can be useful for resolving ambiguities. The importance of disambiguation for successful spatial parsing was pointed out already in Chapter 3. In order to make use of content to support disambiguation, and hence structure detection, we need an appropriate way to combine visual and contentual analysis. Interpretation systems, as described in Sect. 4.2, together with their linked parsers (Sect. 4.2.4) are ideally suited for this purpose. All that needs to be done, is to introduce an additional parsing component which is able to detect contentual instead of spatial or visual relations. Such a "content parser" is then coupled with spatial and visual parser by weighted merging, as described in Sect. 4.2.4.

Except for Φ_C and the weighting formula for object relations, content parsers correspond to visual parsers from Sect. 4.4.3. Therefore Eq. 4.203 to Eq. 4.208 are almost identical to our previous definitions from Eq. 4.184 to Eq. 4.189.

Provided that content relations shall be detected only when new objects were created, when the content of information units has changed or when objects got deleted ...

$$\Phi_C = \left\{ \begin{array}{c} \mathit{CREATE}, \\ \mathit{MODIFY\_CONTENT}, \\ \mathit{DELETE} \end{array} \right\} \tag{4.200}$$

... and assuming, analogous to Eq. 4.120 and Eq. 4.182, that our content parsing algorithm is defined by a function $\mathit{parse}_C : (I\langle\Omega\rangle \times E\langle\Omega\rangle \times D\langle\Omega, k\rangle) \to I\langle\Omega\rangle$ we can partially refine Eq. 4.106 into a content parser model $A_C\langle\Omega, n, \alpha, \beta\rangle$:

$$\begin{aligned} A_C\langle\Omega, n, \alpha, \beta\rangle &:= A_P\langle\Omega, n, \alpha, \beta, \Phi_C, \mathit{parse}_C\rangle \\ T_C\langle\Omega, n, \alpha, \beta\rangle &:= T_P\langle\Omega, n, \alpha, \beta, \Phi_C, \mathit{parse}_C\rangle \\ G_C\langle\Omega, n, \alpha, \beta\rangle &:= G_P\langle\Omega, n, \alpha, \beta, \Phi_C, \mathit{parse}_C\rangle \end{aligned} \tag{4.201}$$

When applying the same default model parameters as we used in Eq. 4.121 and Eq. 4.183 we get:

$$\begin{aligned} A_C &:= A_C\langle\mathbb{N}_0, 16, 0.95, 0.05\rangle \\ T_C &:= T_C\langle\mathbb{N}_0, 16, 0.95, 0.05\rangle \\ G_C &:= G_C\langle\mathbb{N}_0, 16, 0.95, 0.05\rangle \end{aligned} \tag{4.202}$$

Illustrated as a blockdiagram, system components A_C, T_C and G_C become what can be seen in Fig. 4.28.

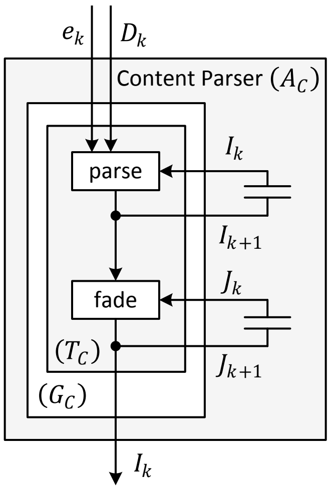


Figure 4.28: Blockdiagram of content parser model A_C as defined in Eq. 4.202

Just like parse_V in Eq. 4.184, also parse_C maps triples (I, e, D) to interpretations I' and therefore matches the definition of φ from Eq. 4.107:

$$\begin{aligned} \mathit{parse}_C : \big(I\langle\Omega\rangle \times E\langle\Omega\rangle \times D\langle\Omega, k\rangle\big) &\rightarrow I\langle\Omega\rangle, \\ \big((U, A, w), e, D\big) &\mapsto \big(U', A', w'\big) \end{aligned} \tag{4.203}$$

With Eq. 4.204 we ensure that the resulting interpretation graph (U', A', w') always includes the latest collection of information units (or rather their identifiers $\in \Omega$):

$$U' = \begin{cases} U \cup e.\mathit{objects} \text{ , if } e.\mathit{operation} = \mathit{CREATE} \\ U \setminus e.\mathit{objects} \text{ , if } e.\mathit{operation} = \mathit{DELETE} \\ U \text{ , else} \end{cases} \tag{4.204}$$

Eq. 4.205 keeps A' consistent with U':

$$A' = \begin{cases} A \cup \binom{e.\mathit{objects}}{2} \cup \left\{ \{u, u'\} \,\middle|\, \begin{matrix} u \in U \\ \wedge \\ u' \in e.\mathit{objects} \end{matrix} \right\} \text{ , if } e.\mathit{operation} = \mathit{CREATE} \\ \binom{U'}{2} \text{ , if } e.\mathit{operation} = \mathit{DELETE} \\ A \text{ , else} \end{cases} \tag{4.205}$$

Apparently, both mappings from U to U' (Eq. 4.204) and from A to A' (Eq. 4.205) are identical to those defined in Eq. 4.185 and Eq. 4.186.

The same applies to the mapping of w to w'. Also w' together with the included set A_{recalc} can be found again in Eq. 4.187 and Eq. 4.188. The only difference between w' of the visual parser and the content parser's w' lies in the function that is used for detecting object relations, now called *detect_content_relation*:

$$\begin{aligned} &w' : A' \to \{\varepsilon, 0.0, \ldots, 1.0\}, \; \forall a \in A' : \\ &\quad w'(a) = \begin{cases} \mathit{detect\_content\_relation}(a, D) & \text{, if } a \in A_{recalc} \\ w(a) & \text{, else} \end{cases} \end{aligned} \tag{4.206}$$

$$A_{recalc} = \binom{e.objects}{2} \cup \left\{ \{u, u'\} \;\middle|\; \begin{matrix} u \in e.objects \\ \wedge \\ u' \in \big(U \setminus e.objects\big) \end{matrix} \right\} \tag{4.207}$$

Just like the strength of visual relations equated with the degree of visual similarity (in Eq. 4.189) also contentual relationships shall depend only on content-related proximity. In concrete terms, only the similarity of content should determine how strong the relation between two information units is. Thus, when two carriers of information include exactly the same content they must have a stronger connection than objects with completely different information payload. Other notions of contentual relation are not considered for our parsing algorithm.

In Sect. 4.1.4 we specified to use string-based content (i. e., text) in our prototypical spatial hypermedia system. According to Eq. 4.87 in Sect. 4.2.2 and Eq. 4.103 in Sect. 4.3 this is available via *d.text*, for any $d \in \mathit{InfoUnitData}\langle k \rangle$.

In order to detect the strength of contentual relationships between two carriers of information u, u', $\mathit{detect\_content\_relation}(\{u, u'\}, (V, p))$ accepts information units $u, u' \in \Omega$ and their assigned properties $(V, p) \in D\langle \Omega, k \rangle$ and calculates the degree of topic related similarity between $p(u).\mathit{text}$ and $p(u').\mathit{text}$ using $\mathit{detect\_topic\_similarity}(p(u).\mathit{text}, p(u').\mathit{text})$. Like the previous calculation of $\mathit{detect\_visual\_similarity}(p(u), p(u'))$ in Eq. 4.189 this requires that $u, u' \in V$. Otherwise ε is used as return value:

$$\begin{aligned} &\mathit{detect\_content\_relation} : \left(\binom{\Omega}{2} \times D\langle \Omega, k \rangle \right) \to \{\varepsilon, 0.0, \ldots, 1.0\}, \\ &\Big(\{u, u'\}, (V, p) \Big) \mapsto y = \begin{cases} \mathit{detect\_topic\_similarity} \begin{pmatrix} p(u).\mathit{text}, \\ p(u').\mathit{text} \end{pmatrix} & \text{, if } u, u' \in V \\ \varepsilon & \text{, else} \end{cases} \end{aligned} \tag{4.208}$$

What really sets the content parser apart from the visual parser is the pairwise detection of thematic similarities, defined by *detect_topic_similarity*.

The approach we have chosen for inferring topic-related similarities builds on probabilistic inference and information theory. In detail, generative topic models are used for computing topic distributions over content and divergence functions calculate how close such distributions come to each other. The model we have selected for statistical inference is the Latent Dirichlet Allocation (LDA) [39]. For measuring the difference, or rather divergence, between topic distributions we use the well-known Kullback Leibler (KL) divergence [40].

With both tools at hand, we compute topic distributions for u and u' ...

$$\theta_0 = topic\_distribution\left(p(u).text\right)$$
$$\theta_1 = topic\_distribution\left(p(u').text\right)$$

... use them as input for a symmetrized KL (since original KL is *asymmetric*)

$$KL_{symmetric}\left(\theta_0, \theta_1\right) = \frac{1}{2}\left(KL\left(\theta_0, \theta_1\right) + KL\left(\theta_1, \theta_0\right)\right) \tag{4.209}$$

... and finally we map the result (which is ≥ 0) to value range $]0, 1]$:

$$e^{-KL_{symmetric}(\theta_0,\theta_1)} = e^{-\frac{1}{2}\left(KL(\theta_0,\theta_1)+KL(\theta_1,\theta_0)\right)}$$

Note, that this value converges towards zero for an increasing divergence.

This gives us a numerical indicator for topic distribution similarity and hence an indicator for similarity of text-based content. Thus we can define:

$$\begin{aligned} &detect\_topic\_similarity : \left(String \times String\right) \to \{0.0, \ldots, 1.0\}, \\ &\qquad (s_0, s_1) \mapsto e^{-\frac{1}{2}\left(KL\left((\theta_0,\theta_1\right)+KL(\theta_1,\theta_0)\right)}, \quad \begin{matrix} \theta_0 = topic\_distribution\left(s_0\right) \\ \theta_1 = topic\_distribution\left(s_1\right) \end{matrix} \end{aligned} \tag{4.210}$$

In a nutshell, our content parser takes a set of information units with text-based content (i. e., documents) as input, creates a thematic profile for each document, measures the pairwise degree of topic-related similarities between them and finally delivers that result in form of a complete, undirected, weighted graph, where weights assigned to edges indicate the strength of thematic relationships. A weight of 0.0 refers to “documents have nothing in common” whereas a value of 1.0 means that “with respect to covered topics both documents are identical”.

Initial tests suggest, that the content parser’s performance strongly depends on settings used for LDA model estimation and inference. We assume that quality of parser output is mainly determined by (1) quality and quantity of training corpus; (2) number of sampling iterations and (3) number of topics. From this we conclude, that there is no default configuration which can always provide us with reasonable parse results. Consequently, in order to use the content parser in a productive system it must be tailored to individual user-requirements.

4.4.5 Temporal Parser

In Chapter 3 we identified three categories of structures that cannot be detected properly by conventional spatial parsers: (1) ambiguous structures (Sect. 3.1); (2) destroyed structures (Sect. 3.2) and (3) temporal structures (Sect. 3.3).

For solving the "destroyed structures problem" from Sect. 3.2 we suggested in Sect. 4.4.1 to extend our generic parser model by some short-term memory. For this see our considerations on the subject "fading" on pages 93–94.

For the detection of ambiguous (Sect. 3.1) and temporal structures (Sect. 3.3), however, it is not sufficient to simply extend existing parser models by additional temporal parsing functionality. Instead we need a separate parsing system with a different system architecture than the one described in Sect. 4.4.1. We need a parser which can process temporal dependencies, a *Temporal Parser*.

Like spatial and visual parser, also temporal parsers measure the strength of relationships between information objects. What differentiates such new parsing systems from their spatial and visual counterparts is, that they do not operate in geometrical 2d-space or in color space, but in *time*. Temporal parsers do not parse a canvas or a workspace. They rather analyze streams of edit events in order to detect how strongly related objects are. In doing so, however, they do not search for pre-defined activity patterns. Just like our spatial and visual parser, also temporal parsers do not perform pattern matching. Since there is no one natural order in which people create spatial hypertext, it is simply not possible to predict user behaviour. We therefore cannot specify default-activities which were universally valid (such as, for instance, constructor or destructor patterns for lists, tables etc.). Thus, we must limit ourselves to temporal heuristics. In a nutshell, our algorithm design was driven by the following simple rule:

Pairs of objects frequently modified at short time intervals have a stronger temporal connection than objects touched infrequently or in long time intervals.

Following this principle it becomes possible to recognize temporal dependencies between information objects, even without considering knowledge about system users or context of application. A complete formal definition of our temporal parsing algorithm can be found on the following pages 160–168. An example is presented on pages 168–176.

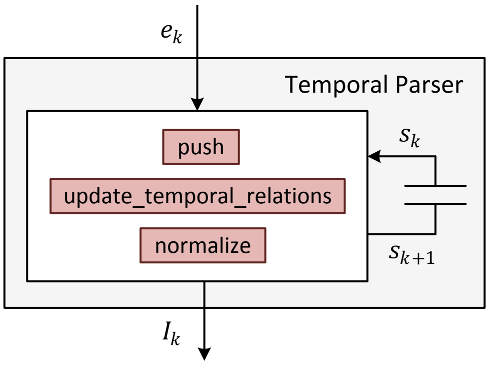


Figure 4.29: temporal parsers are dynamic systems determined by three core functions: *push* (Eq. 4.213), *update_ temporal_ relations* (Eq. 4.214) and *normalize* (Eq. 4.230)

Our temporal parser transforms ingoing sequences of events $e_k \in E\langle\Omega\rangle$ (Eq. 4.71) into outgoing sequences of interpretations $I_k \in I\langle\Omega\rangle$ (Eq. 4.90):

$$\begin{tikzcd}(e_0, e_1, \ldots, e_{k_e}) \arrow[r] & \boxed{\text{Temporal Parser}} \arrow[r] & (I_0, I_1, \ldots, I_{k_e})\end{tikzcd}$$

As illustrated in Fig. 4.29 the core of such dynamic systems is determined by three functions: (1) *push* (Eq. 4.213); (2) *update_ temporal_ relations* (Eq. 4.214) and (3) *normalize* (Eq. 4.230).

In a nutshell, *push* adds ingoing events $e_k \in E\langle\Omega\rangle$ to an internal buffer, from which *update_ temporal_ relations* infers temporal dependencies that are finally brought into form $I_k \in I\langle\Omega\rangle$ by *normalize*.

The buffers used for this are defined by the following set of event tuples:

$$B\langle\Omega, \Delta k\rangle = \left\{ (e_{\Delta k}, \ldots, e_1, e_0) \;\middle|\; \begin{matrix} e_k \in E\langle\Omega\rangle, \\ k = 0, 1, \ldots, \Delta k, \\ \Delta k \in \mathbb{N}_0 \end{matrix} \right\} = (E\langle\Omega\rangle)^{\Delta k+1} \tag{4.211}$$

In simple terms $B\langle\Omega, \Delta k\rangle$ can be understood as the set of all non-empty buffers of length $\Delta k + 1$ whose elements are $\in E\langle\Omega\rangle$. Mathematically this corresponds to the (Δk+1)-ary cartesian product $(E\langle\Omega\rangle)^{\Delta k+1}$. Buffers $(e_{\Delta k}, \ldots, e_1, e_0)$ that include only empty events $e_k = (\emptyset, \varepsilon)$ (for $k = 0, 1, \ldots, \Delta k$) can be identified by the following subset of $B\langle\Omega, \Delta k\rangle$:

$$B_\varepsilon\langle\Delta k\rangle = \left\{ (e_{\Delta k}, \ldots, e_1, e_0) \;\middle|\; \begin{matrix} e_k = (\emptyset, \varepsilon), \\ k = 0, 1, \ldots, \Delta k, \\ \Delta k \in \mathbb{N}_0 \end{matrix} \right\} \subset B\langle\Omega, \Delta k\rangle \tag{4.212}$$

The reason why we explicitly separate these empty buffers $B_\varepsilon\langle\Delta k\rangle$ from their superset $B\langle\Omega, \Delta k\rangle$ is, that we will later on use them as default settings. For the following definition of *push*$\langle\Omega, \Delta k\rangle$, however, only $B\langle\Omega, \Delta k\rangle$ is relevant.

push$\langle\Omega, \Delta k\rangle$ accepts an event buffer $(e_{\Delta k}, \ldots, e_1, e_0) \in B\langle\Omega, \Delta k\rangle$ and an ingoing event $e \in E\langle\Omega\rangle$ and maps both to a new buffer $(e'_{\Delta k}, \ldots, e'_1, e) \in B\langle\Omega, \Delta k\rangle$, with $e'_k = e_{k-1}$, for $k = 1, 2, \ldots, \Delta k$:

$$\begin{aligned} push\,\langle\Omega, \Delta k\rangle : (B\langle\Omega, \Delta k\rangle \times E\langle\Omega\rangle) &\to B\langle\Omega, \Delta k\rangle, \\ \Big((e_{\Delta k}, \ldots, e_1, e_0), e\Big) &\mapsto \left(e'_{\Delta k}, \ldots, e'_1, e\right), \\ e'_k &= e_{k-1},\ k = 1, 2, \ldots, \Delta k \end{aligned} \tag{4.213}$$

In a nutshell, *push*$\langle\Omega, \Delta k\rangle$ performs a left-shift. That is, we insert a new element at the right end of of the event tuple and remove one from the left. Hence, the buffer length remains unchanged. This way *push*$\langle\Omega, \Delta k\rangle$ turns simple event tuples $(e_{\Delta k}, \ldots, e_1, e_0)$ into ring buffers of capacity $\Delta k + 1$.

Such buffers are processed by *update_ temporal_ relations*$\langle\Omega, \Delta k, \alpha\rangle$ (Eq. 4.214).

update_ temporal_ relations(J, B) accepts two arguments: $J \in I\langle\Omega, \mathbb{R}_0^+\rangle$ and $B \in B\langle\Omega, \Delta k\rangle$. J is an interpretation graph with weightings $\in \mathbb{R}_0^+$. Note, that this should not be confused with the parser's final result interpretation (which is $\in I\langle\Omega, \{\varepsilon, 0.0, \ldots, 1.0\}\rangle$ instead). J is rather meant to be an internal structure that is used to keep track of how frequently pairs of objects are modified. We will see afterwards how that works in detail. The second argument B shall be the latest event buffer generated by *push*$\langle\Omega, \Delta k\rangle$. That is, we expect B to be an event tuple $(e_{\Delta k}, \ldots, e_1, e_0) \in B\langle\Omega, \Delta k\rangle$ with e_0 being the latest edit event. We define *update_ temporal_ relations*$\langle\Omega, \Delta k, \alpha\rangle$ as follows:

$$\begin{aligned} update\_\ temporal\_\ relations\,\langle\Omega, \Delta k, \alpha\rangle : \Big(I\langle\Omega, \mathbb{R}_0^+\rangle \times B\langle\Omega, \Delta k\rangle\Big) &\to I\langle\Omega, \mathbb{R}_0^+\rangle, \\ (J, B) &\mapsto trim\Big(add\_\ weights(J, B)\Big) \end{aligned} \tag{4.214}$$

To put it briefly, what happens in Eq. 4.214 is, that we infer the latest temporal weightings from buffer B, add them to the weights in J (via *add_ weights*(J, B)) and finally cut the resulting values down in order to prevent them from becoming arbitrarily large. The latter is achieved by *trim* which is defined as ...

$$\begin{aligned} trim : I\langle\Omega, \mathbb{R}_0^+\rangle &\to I\langle\Omega, \mathbb{R}_0^+\rangle,\ I = (U, A, w) \mapsto I' = \left(U, A, w'\right), \\ w' &: A \to \mathbb{R}_0^+,\ \forall a \in A : \\ w'(a) &= \begin{cases} w(a) - w^+_{min}(I) + 1.0 \text{ , if } \Big(w^+_{min}(I) > 1.0\Big) \wedge \big(w(a) > 0.0\big) \\ w(a) \text{ , else} \end{cases} \end{aligned} \tag{4.215}$$

In short, when the smallest positive weight in I has exceeded a threshold of 1.0 (i. e., $w^{+}_{min}(I) > 1.0$) then $trim(I)$ reduces all $w(a) > 0.0$ by $(w^{+}_{min}(I) - 1.0)$.

The smallest positive weight $w(a)$ in $(U, A, w) \in I\langle\Omega, \mathbb{R}^{+}_{0}\rangle$ can be identified by:

$$w^{+}_{min} : I\langle\Omega, \mathbb{R}^{+}_{0}\rangle \to \mathbb{R}^{+}_{0},$$
$$w^{+}_{min}\Big((U, A, w)\Big) = \begin{cases} min\Big(w\,(A) \setminus \{0.0\}\Big) \text{ , if } \Big(w\,(A) \setminus \{0.0\}\Big) \neq \emptyset \\ 0.0 \text{ , else} \end{cases} \tag{4.216}$$

The second operation our definition of *update_ temporal_ relations* $\langle\Omega, \Delta k, \alpha\rangle$ builds on, is *add_ weights* $\langle\Omega, \Delta k, \alpha\rangle$. *add_ weights* (J, B) takes an interpretation graph $J = (U, A, w) \in I\langle\Omega, \mathbb{R}^{+}_{0}\rangle$, analyses the temporal dependencies arising from a given event buffer $B \in B\langle\Omega, \Delta k\rangle$ and generates a new graph $(U', A', w') \in I\langle\Omega, \mathbb{R}^{+}_{0}\rangle$ with (possibly) increased temporal weightings:

$$add\_\,weights\,\langle\Omega, \Delta k, \alpha\rangle : \Big(I\langle\Omega, \mathbb{R}^{+}_{0}\rangle \times B\langle\Omega, \Delta k\rangle\Big) \to I\langle\Omega, \mathbb{R}^{+}_{0}\rangle,$$
$$\Big((U, A, w), (e_{\Delta k}, \ldots, e_1, e_0)\Big) \mapsto \Big(U', A', w'\Big) \tag{4.217}$$

Provided that e_0 represents the *latest* edit event, we define U' as ...

$$U' = \begin{cases} U \cup e_0.objects \text{ , if } e_0.operation = CREATE \\ U \setminus e_0.objects \text{ , if } e_0.operation = DELETE \\ U \text{ , else} \end{cases} \tag{4.218}$$

With this definition we ensure, that graphs $(U', A', w') \in I\langle\Omega, \mathbb{R}^{+}_{0}\rangle$ that were generated by *add_ weights* $\langle\Omega, \Delta k, \alpha\rangle$ always include the latest collection of information units U'. Since graphs $\in I\langle\Omega, \mathbb{R}^{+}_{0}\rangle$ are required to be complete, any modification of U must be accompanied by the following update of A:

$$A' = \begin{cases} A \cup \binom{e_0.objects}{2} \cup \left\{ \{u, u'\} \;\middle|\; \begin{matrix} u \in U \\ \wedge \\ u' \in e_0.objects \end{matrix} \right\} & \text{, if } e_0.operation \\ & \quad = CREATE \\ \binom{U'}{2} & \text{, if } e_0.operation \\ & \quad = DELETE \\ A & \text{, else} \end{cases} \tag{4.219}$$

Apparently, aside from the fact that both U' and A' depend on the latest event buffer entry e_0, their definitions are identical to Eq. 4.185 and Eq. 4.186 or Eq. 4.204 and Eq. 4.205 respectively.

This is not the case for w'. Unlike U' and A' the definition of w' differs fundamentally from Eq. 4.187 and Eq. 4.206, not least because of a different target set $\mathbb{R}_0^+$ but also because of different semantics. In contrary to Eq. 4.187 and Eq. 4.206 the weighting function w', as we define it here, assigns to associations $a \in A'$ *absolute* rather than *relative* values. More precisely, temporal weightings $w'(a)$ are rather counters for how frequently pairs of objects $a = \{u, u'\}$ were modified than relative strength values $\in \{0.0, \dots, 1.0\}$. Relative values are first generated by function *normalize* which will be described afterwards on page 166 (Eq. 4.230).

We define w' as follows:

$$w' : A' \to \mathbb{R}_0^+,\ \forall a \in A' :$$
$$w'(a) = \begin{cases} w(a)\ , \text{if} \begin{pmatrix} (e_0.\mathit{operation} = \mathit{DELETE}) \\ \vee \\ (e_0.\mathit{operation} = \mathit{CREATE} \wedge a \in A) \end{pmatrix} \\ \Delta w(a)\ , \text{if} \begin{pmatrix} (e_0.\mathit{operation} = \mathit{CREATE}) \\ \wedge \\ \big(a \in (A' \setminus A)\big) \end{pmatrix} \\ w(a) + \Delta w(a)\ , \text{else} \end{cases} \tag{4.220}$$

In Eq. 4.220 we map associations $a \in A'$ either to $w(a)$, $\Delta w(a)$ or $w(a) + \Delta w(a)$. Which term we choose depends on the value of $e_0.\mathit{operation}$ and whether $a \in A$. There are three specific cases:

In case of $e_0.\mathit{operation} = \mathit{CREATE}$ (when $A' \supseteq A$) an explicit distinction should be made between *new* associations $a \in (A' \setminus A)$ and $a \in A$. New associations $a \in (A' \setminus A)$ do not have an assigned value $w(a)$ since w is only defined on A. This is why we "initialize" a with $w'(a) = \Delta w(a)$. If, otherwise, $a \in A$ then assigned weights remain unchanged, that is $w'(a) = w(a)$. The reason for this is, that the creation of new objects should not have a direct impact on relations between objects that already existed.

The same applies in case of $e_0.\mathit{operation} = \mathit{DELETE}$. When objects were removed, and hence $A' \subseteq A$, weights of remaining associations $a \in A'$ should not change. Therefore, $\forall a \in A' : w'(a) = w(a)$ which results in $w' \subseteq w$.

In all other cases, when $U' = U$ and thus also $A' = A$, temporal weightings $w(a)$ get increased by $\Delta w(a)$, more precisely: $\forall a \in A' : w'(a) = w(a) + \Delta w(a)$. This has the effect of strengthening temporal connections.

We define this weighting delta $\Delta w(a)$ as ...

$$\Delta w(a) = \begin{cases} \Delta w'(a) \text{ , } a \in \mathit{Def}\big(\Delta w'\big) \\ 0 \text{ , else} \end{cases} \tag{4.221}$$

Essentially, $\Delta w'(a)$ assigns to each association $a = \{u, u'\}$ for which we can find at least one unordered pair of events $\{e_0, e_k\}$ $(0 \leq k \leq \Delta k)$ in the event buffer, with $u \in e_0.\mathit{objects}$ and $u' \in e_k.\mathit{objects}$, a weight inferred from the temporal distance between e_0 and e_k (that is, from k). We will see afterwards how such weightings are calculated. For all other associations the weighting delta becomes zero, hence $\Delta w(a) = 0$.

We define $\Delta w'$ as a binary relation $R_{\Delta w}$ that assigns weighting deltas $\in \mathbb{R}_0^+$ to associations $a \in \Sigma_{A_\Delta}(\Delta k)$ (see Eq. 4.228):

$$\Delta w' : \Big(\Sigma_{A_\Delta}\left(\Delta k\right)\Big) \to \mathbb{R}_0^+ \text{ , } \Delta w' = R_{\Delta w} \tag{4.222}$$

For assembling $R_{\Delta w}$ we determine for each event buffer entry e_k (for $k = 0$ to $k = \Delta k$) the latest temporal connections between $e_0.\mathit{objects}$ and $e_k.\mathit{objects}$, collect them in binary relations $R'_{\Delta w}(k)$ and finally calculate ...

$$R'_{\Delta w}(0) \cup R'_{\Delta w}(1) \cup \ldots \cup R'_{\Delta w}\left(\Delta k\right)$$

Hence, $R_{\Delta w}$ can be defined as:

$$R_{\Delta w} = \bigcup_{k=0}^{\Delta k} \Big(R'_{\Delta w}(k)\Big) \tag{4.223}$$

Such binary relations $R'_{\Delta w}(k)$ assign to all $a \in A'_\Delta(k)$ (see Eq. 4.226) positive real-valued weights; hence $A'_\Delta(k) \to \mathbb{R}_0^+$. The weighting formula used for this depends on two parameters α and k. It is defined as:

$$2^{-\alpha k} \tag{4.224}$$

Here, α is a real-valued tuning parameter that determines how temporal weights evolve for an increasing $k \geq 0$. With this determining factor it can be controlled whether object events occurring in long time intervals get more $(\alpha < 0)$, less $(\alpha > 0)$ or the same weight $(\alpha = 0)$ as events happening at short time intervals. Here we refer to events in discrete time. That is, our temporal weightings are determined by logical order only. The time that really ellapsed between pairs of events is not taken into consideration. This is intended to avoid temporal over-interpretation.

Thus, when we go back from “present time” ($k = 0$) to the earliest event the temporal parser can “remember” ($k = \Delta k$) our weighting formula will turn discrete time steps $k = 0, 1, 2, \ldots, \Delta k$ into weighting deltas:

$$1, \frac{1}{2^{\alpha}}, \frac{1}{2^{2\alpha}}, \ldots, \frac{1}{2^{\Delta k \alpha}} \quad \text{(for any } \alpha \in \mathbb{R}\text{)}$$

Binary relations formed by $R'_{\Delta w}(k)$ assign such weighting deltas $2^{-\alpha k}$ to all $a \in A'_{\Delta}(k)$ (see Eq. 4.226). Thus, they can be defined as:

$$R'_{\Delta w}(k) = \bigcup_{a \in A'_{\Delta}(k)} \left\{ \left(a, 2^{-\alpha k} \right) \right\} \tag{4.225}$$

$A'_{\Delta}(k)$ determines all associations $a = \{u, u'\}$ between objects $u \in e_0.\mathit{objects}$ and $u' \in e_k.\mathit{objects}$ (i. e., $a \in A_{\Delta}(k)$) for which there is no more recent event e_j in the event buffer (with $j < k$) where $u' \in e_j.\mathit{objects}$ (i. e., $a \notin \Sigma_{A_{\Delta}}(k-1)$):

$$A'_{\Delta}(k) = \left\{ a \;\middle|\; \begin{array}{c} a \in A_{\Delta}(k) \\ \wedge \\ a \notin \Sigma_{A_{\Delta}}(k-1) \end{array} \right\} = \Big(A_{\Delta}(k) \setminus \Sigma_{A_{\Delta}}(k-1) \Big) \subseteq A_{\Delta}(k) \tag{4.226}$$

With $A_{\Delta}(k)$ we identify all $\{u, u'\}$ between objects u that are affected by the latest event in the event buffer (i. e., $u \in e_0.\mathit{objects}$) and objects u' for which an event occurred k discrete time steps in the past (i. e., $u' \in e_k.\mathit{objects}$):

$$A_{\Delta}(k) = \left\{ \{u, u'\} \;\middle|\; \begin{array}{c} u \in e_0.\mathit{objects} \\ \wedge \\ u' \in e_k.\mathit{objects} \\ \wedge \\ u \neq u' \end{array} \right\} \tag{4.227}$$

When we generalize $A_{\Delta}(k)$ from Eq. 4.227 for full ranges of discrete time steps $0 \leq k \leq n$ so that $A_{\Delta}(0) \cup A_{\Delta}(1) \cup \ldots \cup A_{\Delta}(n)$ then we get for $n \leq \Delta k$:

$$\Sigma_{A_{\Delta}}(n) = \begin{cases} \left(\displaystyle\bigcup_{k=0}^{n} A_{\Delta}(k) \right) & , n \geq 0 \\ \emptyset & , \text{else} \end{cases} \tag{4.228}$$

Eq. 4.222 to Eq. 4.228 allow to simplify our original definition of $\Delta w(a)$ from Eq. 4.221 to:

$$\Delta w(a) = \sum_{k=0}^{\Delta k} \left(\Big| \{a\} \cap \big(A_{\Delta}(k) \setminus \Sigma_{A_{\Delta}}(k-1) \big) \Big| \times 2^{-\alpha k} \right) \tag{4.229}$$

Expressed in simple terms, *update_ temporal_ relations* $\langle\Omega,\Delta k,\alpha\rangle$, as we defined it in Eq. 4.214, counts how frequently pairs of objects $\{u,u'\}$ are modified and stores these values in a complete graph $\in I\langle\Omega,\mathbb{R}_0^+\rangle$.

Even though the temporal parser does not build on the generic parser model we defined in Sect. 4.4.1, it still has to be compatible with our parser definition from Sect. 4.2.4. Otherwise, it would not be possible to integrate it into our prototype system. Consequently, the temporal parser must accept (parts of) ingoing edit steps $(e,D)\in(E\langle\Omega\rangle\times D\langle\Omega,k\rangle)$ (see Eq. 4.71; Eq. 4.85) and has to generate temporal interpretations $I\in I\langle\Omega\rangle$ (Eq. 4.90) as output. The first condition is met by our definition of *push* $\langle\Omega,\Delta k\rangle$ in Eq. 4.213. The second one is guaranteed to be satisfied by the following definition of *normalize*:

normalize (I) accepts interpretation graphs $I\in I\langle\Omega,\mathbb{R}_0^+\rangle$, as generated by *update_ temporal_ relations* $\langle\Omega,\Delta k,\alpha\rangle$, transforms included temporal weightings (or rather frequency values) into relative strength values $\in\{0.0,\ldots,1.0\}$ and provides that result in form of an interpretation $I'\in I\langle\Omega\rangle$:

$$\begin{aligned} \mathit{normalize}: I\langle\Omega,\mathbb{R}_0^+\rangle &\to I\langle\Omega\rangle,\ (U,A,w)\mapsto\big(U,A,w'\big)\,,\\ w' &: A\to\{0.0\ldots1.0\}\,,\forall a\in A:\\ w'(a) &= \begin{cases} \dfrac{w(a)}{\mathit{max}\Big(w\,(A)\Big)} & ,\ \text{if } \mathit{max}\Big(w\,(A)\Big)>0\\ 0 & ,\ \text{else}\end{cases} \end{aligned} \tag{4.230}$$

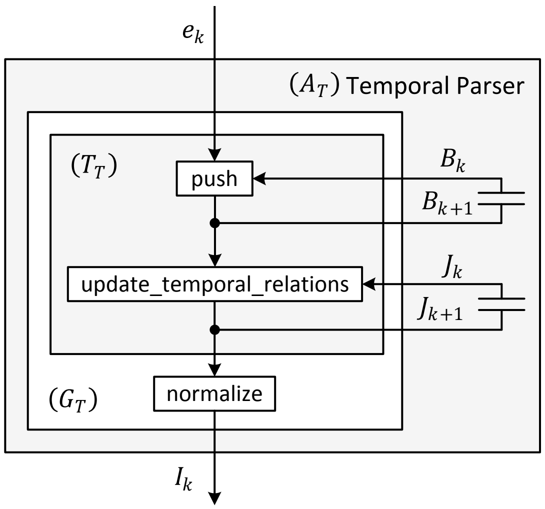
The rationale behind this weight transformation is, that temporal dependencies are something *relative*. This means, unlike spatial, visual and content parsing, where it is (theoretically) sufficient to view pairs of objects in isolation, temporal parsing cannot be performed without considering the context (i. e. the relations between all objects in U). Only when we know which (of all) $u,u'\in U$ were touched most frequently we can infer reasonable strength values $\in\{0.0\ldots1.0\}$ for all pairs of objects. This is why in Eq. 4.230 we map $\mathit{max}(w(A))$ to 1.0 and $w(a)<\mathit{max}(w(A))$ to weightings <1.0.

Provided that *push* $\langle\Omega,\Delta k\rangle$ (Eq. 4.213), *update_ temporal_ relations* $\langle\Omega,\Delta k,\alpha\rangle$ (Eq. 4.214) and *normalize* (Eq. 4.230) are sequentially connected, sharing the same event buffer $B\in B\langle\Omega,\Delta k\rangle$ and frequency graph $J\in I\langle\Omega,\mathbb{R}_0^+\rangle$, we can understand temporal parsers as dynamic systems, that accept ingoing sequences of events $e_k\in E\langle\Omega\rangle$ and generate (depending on latest buffer entries in B and collected weights in J) outgoing sequences of interpretations $I_k\in I\langle\Omega\rangle$. As with spatial parser (Eq. 4.120), visual parser (Eq. 4.182) and content parser (Eq. 4.201) also such temporal parsing systems can be described using deterministic automata. Such a model with initial state (B,I_ε), for $B\in B_\varepsilon\langle\Delta k\rangle$ (Eq. 4.212), is given in the following Eq. 4.231. For a definition of the empty interpretation I_ε see Eq. 4.89.

$$A_T\langle\Omega,\Delta k,\alpha\rangle = \begin{pmatrix} S_T\langle\Omega,\Delta k\rangle, \\ E_T\langle\Omega\rangle, \\ O_T\langle\Omega\rangle, \\ T_T\langle\Omega,\Delta k,\alpha\rangle, \\ G_T\langle\Omega,\Delta k,\alpha\rangle, \\ s_{init}\langle\Delta k\rangle \end{pmatrix}, \Delta k \in \mathbb{N}_0,\ \alpha \in \mathbb{R}$$

$$\begin{aligned}
&S_T\langle\Omega,\Delta k\rangle = B\langle\Omega,\Delta k\rangle \times I\langle\Omega,\mathbb{R}_0^+\rangle \\
&E_T\langle\Omega\rangle = E\langle\Omega\rangle \\
&O_T\langle\Omega\rangle = I\langle\Omega\rangle \\
&T_T\langle\Omega,\Delta k,\alpha\rangle : \big(S_T\langle\Omega,\Delta k\rangle \times E_T\langle\Omega\rangle\big) \to S_T\langle\Omega,\Delta k\rangle, \Big(\left(B,J\right),e\Big) \mapsto \left(B',J'\right), \\
&\quad B' = push\,\langle\Omega,\Delta k\rangle\,(B,e) \\
&\quad J' = update\_\,temporal\_\,relations\,\langle\Omega,\Delta k,\alpha\rangle\,\left(J,B'\right) \\
&G_T\langle\Omega,\Delta k,\alpha\rangle : \big(S_T\langle\Omega,\Delta k\rangle \times E_T\langle\Omega\rangle\big) \to O_T\langle\Omega\rangle, \Big(\left(B,J\right),e\Big) \mapsto I, \\
&\quad I = normalize\left(J'\right), \\
&\quad \left(B',J'\right) = T_T\langle\Omega,\Delta k,\alpha\rangle\Big(\left(B,J\right),e\Big) \\
&s_{init}\langle\Delta k\rangle = (B,I_\varepsilon) \in S_T\langle\Omega,\Delta k\rangle, B \in B_\varepsilon\langle\Delta k\rangle
\end{aligned} \tag{4.231}$$

Illustrated as a block diagram:



We can conclusively summarize the behaviour of temporal parsers as follows:

Temporal parsers $A_T\langle\Omega, \Delta k, \alpha\rangle$, as they are defined by Eq. 4.231, accept ingoing sequences of edit events $E(0 \dots k_e)$,

$$E\,(0 \dots k_e) = \left(e_0, e_1, \dots, e_{k_e}\right), \qquad e_k \in E_T\langle\Omega\rangle,\; k = 0, 1, \dots, k_e$$

Driven by $E(0 \dots k_e)$ they pass through states $S(0 \dots k_e + 1)$,

$$S\,(0 \dots k_e + 1) = \left(s_0, s_1, \dots, s_{k_e+1}\right) = \left(\binom{B_0}{J_0}, \binom{B_1}{J_1}, \dots, \binom{B_{k_e+1}}{J_{k_e+1}}\right),$$

$$s_0 = s_{init}\langle\Delta k\rangle$$
$$s_{k+1} = T_T\langle\Omega, \Delta k, \alpha\rangle\,(s_k, e_k)\,,\; k = 0, 1, \dots, k_e$$

Sequences of interpretations $I(0 \dots k_e)$ are provided as output:

$$I\,(0 \dots k_e) = \left(I_0, I_1, \dots, I_{k_e}\right), \qquad I_k = G_T\langle\Omega, \Delta k, \alpha\rangle\,(s_k, e_k)\,,\; k = 0, 1, \dots, k_e$$

For a better understanding of Eq. 4.231, let us finally go through an example.

As with spatial and visual parser, default parameter settings have emerged also from our practical experience with the temporal parser. Those settings allow us to refine the generic temporal parser model from Eq. 4.231 as follows:

$$\begin{aligned} A_T &:= A_T\langle\mathbb{N}_0, 6, 0.5\rangle \\ T_T &:= T_T\langle\mathbb{N}_0, 6, 0.5\rangle \\ G_T &:= G_T\langle\mathbb{N}_0, 6, 0.5\rangle \end{aligned} \tag{4.232}$$

However, for the sake of simplicity, we do not use this configuration for the following demonstration. In order to simplify calculation and thus to keep mathematical expressions as short as possible we rather set ...

$$\Omega = \mathbb{N}_0\;;\; \Delta k = 3\;;\; \alpha = 1.0$$

Hence, for our sample run the temporal parser becomes ...

$$A_T\langle\mathbb{N}_0, 3, 1.0\rangle$$

In brief, this means that (1) information units are identified by integers ≥ 0; (2) the parser keeps a maximum of $\Delta k + 1 = 4$ edit events in "memory" (Eq. 4.211) and (3) pairs of events $\{e_0, e_k\}$ get weighted with 2^{-k} (see Eq. 4.224).

According to Eq. 4.231 the initial state s_0 of $A_T\langle\mathbb{N}_0, 3, 1.0\rangle$ becomes:

$$s_0 = \begin{pmatrix} B_0 \\ J_0 \end{pmatrix} = s_{init}\langle 3\rangle = \begin{pmatrix} \big((\emptyset,\varepsilon), (\emptyset,\varepsilon), (\emptyset,\varepsilon), (\emptyset,\varepsilon)\big) \\ (\emptyset,\emptyset,\emptyset) \end{pmatrix}$$

So we start out with two components, an event buffer B_0 with capacity 4 that includes only empty events ...

$$B_0 = \big((\emptyset,\varepsilon), (\emptyset,\varepsilon), (\emptyset,\varepsilon), (\emptyset,\varepsilon)\big) \in B_\varepsilon\langle 3\rangle$$

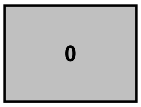
... and an empty frequency graph J_0:

$$J_0 = I_\varepsilon = (\emptyset,\emptyset,\emptyset) \in I\langle\mathbb{N}_0, \mathbb{R}_0^+\rangle$$

Let us now assume the following scenario: Three rectangular information units are generated in sequential steps one after the other. For the sake of simplicity, we number them consecutively as $0, 1, 2$. Position, alignment, color etc. are of no importance in this example. When all three rectangles exist the shape of objects 1 und 2 shall change from *RECTANGLE* to *ELLIPSE* (that is, both objects are modified spatially; for details on this see Eq. 4.101 and Eq. 4.65). Finally object 1 is deleted. This simple five-stage editing process can be approximately described with the following sequence of events:

$$E(0\ldots 4) = (e_0, e_1, e_2, e_3, e_4) = \begin{pmatrix} (\{0\}, \mathit{CREATE}), \\ (\{1\}, \mathit{CREATE}), \\ (\{2\}, \mathit{CREATE}), \\ (\{1,2\}, \mathit{MODIFY\_SPATIAL}), \\ (\{1\}, \mathit{DELETE}) \end{pmatrix}$$

Let us now demonstrate how $A_T\langle\mathbb{N}_0, 3, 1.0\rangle$ reacts to this ingoing sequence of events. We begin by generating the first rectangular information unit 0 ...



... which is represented symbolically by $e_0 = (\{0\}, \mathit{CREATE})$.

As we know already, single temporal parser runs always take place in three sequential steps: (1) buffering the latest input event e_k: $B_{k+1} = \mathit{push}(B_k, e_k)$; (2) updating frequency graph J_k: $J_{k+1} = \mathit{update\_temporal\_relations}(J_k, B_{k+1})$ and (3) transforming J_{k+1} into $I_k \in I\langle\Omega\rangle$: $I_k = \mathit{normalize}(J_{k+1})$.

In our example, the first two steps (buffering e_0 and updating J_0) result in:

$$\begin{aligned} B_1 &= \mathit{push}\left(B_0, e_0\right) = \Big(\left(\emptyset, \varepsilon\right), \left(\emptyset, \varepsilon\right), \left(\emptyset, \varepsilon\right), \left(\{0\}, \mathit{CREATE}\right)\Big) \\ J_1 &= \mathit{update\_temporal\_relations}\left(J_0, B_1\right) = \left(\{0\}, \emptyset, \emptyset\right) \end{aligned}$$

Therefore, driven by $e_0 = (\{0\}, \mathit{CREATE})$, the temporal parser $A_T\langle \mathbb{N}_0, 3, 1.0\rangle$ switches from it's initial state ...

$$\begin{pmatrix} B_0 \\ J_0 \end{pmatrix} = \begin{pmatrix} \Big(\left(\emptyset, \varepsilon\right), \left(\emptyset, \varepsilon\right), \left(\emptyset, \varepsilon\right), \left(\emptyset, \varepsilon\right)\Big) \\ \left(\emptyset, \emptyset, \emptyset\right) \end{pmatrix}$$

... to a first successor state:

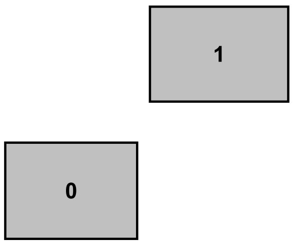
$$\begin{pmatrix} B_1 \\ J_1 \end{pmatrix} = \begin{pmatrix} \Big(\left(\emptyset, \varepsilon\right), \left(\emptyset, \varepsilon\right), \left(\emptyset, \varepsilon\right), \left(\{0\}, \mathit{CREATE}\right)\Big) \\ \left(\{0\}, \emptyset, \emptyset\right) \end{pmatrix}$$

Normalizing $J_1 = (\{0\}, \emptyset, \emptyset)$ results in an identical I_0:

$$I_0 = \mathit{normalize}\left(J_1\right) = \left(\{0\}, \emptyset, \emptyset\right)$$

From $U(I_0) = \{0\}$ it becomes obvious, that all the temporal parser has recognized so far is a single information unit labeled with 0. Weighted associations are not included yet in I_0; thus $A(I_0) = \emptyset$.

This changes with the second edit step when we introduce another rectangular information unit 1 and hence complete the first pair of objects $\{0, 1\}$. Let us assume the following updated workspace:



When we add the creation event for object 1 (i. e., $e_1 = (\{1\}, \mathit{CREATE})$) to our event buffer B_1 we get (similar to the first edit step e_0):

$$B_2 = \mathit{push}\left(B_1, e_1\right) = \Big(\left(\emptyset, \varepsilon\right), \left(\emptyset, \varepsilon\right), \left(\{0\}, \mathit{CREATE}\right), \left(\{1\}, \mathit{CREATE}\right)\Big)$$

Thus, the first two cells of B_2 are now assigned with non-empty events. Using this updated event buffer, previous frequency graph $J_1 = (\{0\}, \emptyset, \emptyset)$ becomes:

$$\begin{aligned} J_2 &= \mathit{update\_temporal\_relations}\,(J_1, B_2) \\ &= \left(\{0,1\}, \{\{0,1\}\}, \left\{\left(\{0,1\}, \Delta w\left(\{0,1\}\right)\right)\right\}\right) \\ &= \left(\{0,1\}, \{\{0,1\}\}, \left\{\left(\{0,1\}, 2^{-1}\right)\right\}\right) \\ &= \left(\{0,1\}, \{\{0,1\}\}, \left\{\left(\{0,1\}, 0.5\right)\right\}\right) \end{aligned}$$

Here, the temporal parser has assigned the first pair of objects a weighting: $\{0,1\}$ is initialized with $\Delta w\left(\{0,1\}\right)$. Since $1 \in e_0.\mathit{objects}$, $0 \in e_1.\mathit{objects}$ and $\alpha = 1.0$ this initial temporal weight becomes $2^{-\alpha k} = 2^{-k} = 2^{-1} = 0.5$. For details see Eq. 4.220 and Eq. 4.229.

Therefore, $A_T\langle \mathbb{N}_0, 3, 1.0\rangle$ switches from ...

$$\begin{pmatrix} B_1 \\ J_1 \end{pmatrix} = \begin{pmatrix} \left((\emptyset, \varepsilon), (\emptyset, \varepsilon), (\emptyset, \varepsilon), (\{0\}, \mathit{CREATE})\right) \\ (\{0\}, \emptyset, \emptyset) \end{pmatrix}$$

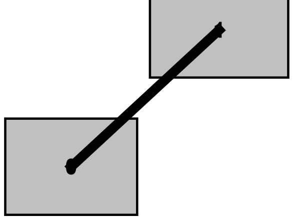
... to consecutive state:

$$\begin{pmatrix} B_2 \\ J_2 \end{pmatrix} = \begin{pmatrix} \left((\emptyset, \varepsilon), (\emptyset, \varepsilon), (\{0\}, \mathit{CREATE}), (\{1\}, \mathit{CREATE})\right) \\ \left(\{0,1\}, \{\{0,1\}\}, \left\{\left(\{0,1\}, 0.5\right)\right\}\right) \end{pmatrix}$$

Because $\Delta w\left(\{0,1\}\right) = 0.5$ is the first and only weight in J_2, it is necessarily also the biggest one. Therefore, the strongest temporal connection in J_2 exists between object 0 und object 1. Consequently *normalize* transforms J_2 into the following I_1:

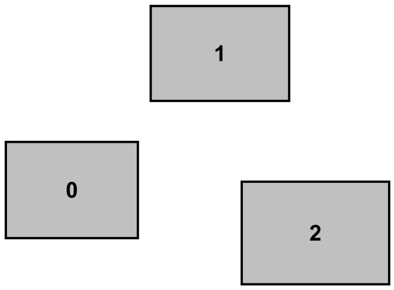
$$I_1 = \mathit{normalize}\,(J_2) = \left(\{0,1\}, \{\{0,1\}\}, \left\{\left(\{0,1\}, 1.0\right)\right\}\right)$$

Illustrated graphically I_1 becomes:



Like in previous illustrations of interpretation graphs, also here weightings are drawn as black lines with varying brightness, thickness and opacity. The higher weightings are the thicker the connecting line and the stronger the line color. In this simple example weight 1.0 is drawn as an opaque black bar.

In a similar fashion $A_T\langle \mathbb{N}_0, 3, 1.0\rangle$ reacts to the creation of the third and last object, symbolized by $(\{2\}, \mathit{CREATE})$. For this let us assume the following updated information space:



Now, the event buffer includes all three creation events (for 0, 1 and 2):

$$\begin{aligned} B_3 &= \mathit{push}\,(B_2, e_2) \\ &= \Big((\emptyset, \varepsilon), (\{0\}, \mathit{CREATE}), (\{1\}, \mathit{CREATE}), (\{2\}, \mathit{CREATE})\Big) \end{aligned}$$

update_temporal_relations (J_2, B_3) leaves the weighting of $\{0,1\}$ unchanged and initializes new associations $\{0,2\}$ with $\Delta w(\{0,2\}) = 2^{-2} = 0.25$ and $\{1,2\}$ with $\Delta w(\{1,2\}) = 2^{-1} = 0.5$:

$$\begin{aligned} J_3 &= \mathit{update\_temporal\_relations}\,(J_2, B_3) \\ &= \left(\{0,1,2\}, \left\{\begin{matrix}\{0,1\}, \\ \{0,2\}, \\ \{1,2\}\end{matrix}\right\}, \left\{\begin{matrix}\Big(\{0,1\}, w(\{0,1\})\Big), \\ \Big(\{0,2\}, \Delta w(\{0,2\})\Big), \\ \Big(\{1,2\}, \Delta w(\{1,2\})\Big)\end{matrix}\right\}\right) \\ &= \left(\{0,1,2\}, \left\{\begin{matrix}\{0,1\}, \\ \{0,2\}, \\ \{1,2\}\end{matrix}\right\}, \left\{\begin{matrix}(\{0,1\}, 0.5), \\ (\{0,2\}, 2^{-2}), \\ (\{1,2\}, 2^{-1})\end{matrix}\right\}\right) \\ &= \left(\{0,1,2\}, \left\{\begin{matrix}\{0,1\}, \\ \{0,2\}, \\ \{1,2\}\end{matrix}\right\}, \left\{\begin{matrix}(\{0,1\}, 0.50), \\ (\{0,2\}, 0.25), \\ (\{1,2\}, 0.50)\end{matrix}\right\}\right) \end{aligned}$$

Thus, $A_T\langle\mathbb{N}_0, 3, 1.0\rangle$ switches from ...

$$\begin{pmatrix} B_2 \\ J_2 \end{pmatrix} = \begin{pmatrix} \Big((\emptyset, \varepsilon), (\emptyset, \varepsilon), (\{0\}, \mathit{CREATE}), (\{1\}, \mathit{CREATE}) \Big) \\ \Big(\{0,1\}, \{\{0,1\}\}, \big\{ (\{0,1\}, 0.5) \big\} \Big) \end{pmatrix}$$

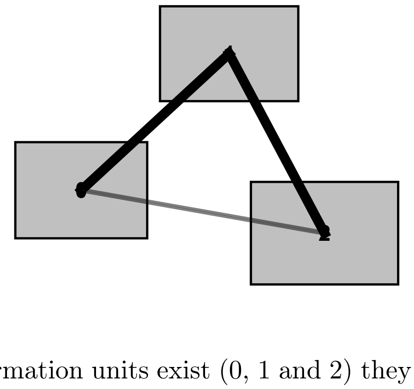
... to successor state:

$$\begin{pmatrix} B_3 \\ J_3 \end{pmatrix} = \begin{pmatrix} \Big((\emptyset, \varepsilon), (\{0\}, \mathit{CREATE}), (\{1\}, \mathit{CREATE}), (\{2\}, \mathit{CREATE}) \Big) \\ \left(\{0,1,2\}, \left\{ \begin{array}{c} \{0,1\}, \\ \{0,2\}, \\ \{1,2\} \end{array} \right\}, \left\{ \begin{array}{c} (\{0,1\}, 0.50), \\ (\{0,2\}, 0.25), \\ (\{1,2\}, 0.50) \end{array} \right\} \right) \end{pmatrix}$$

Now the strongest temporal connections can be found between $0, 1$ and $1, 2$. Both pairs of objects $\{0, 1\}$ and $\{1, 2\}$ have the same maximal weight 0.5. Objects 0 and 2, however, were not created immediately one after the other but with event $(\{1\}, \mathit{CREATE})$ in between. For this reason, the temporal connection between 0 and 2 is only half as strong as between $0, 1$ and $1, 2$. This gets also reflected by the strength values calculated by $\mathit{normalize}(J_3)$:

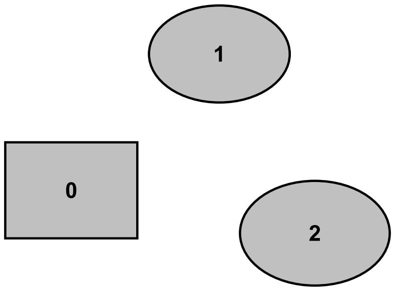
$$I_2 = \mathit{normalize}(J_3) = \left(\{0,1,2\}, \left\{ \begin{array}{c} \{0,1\}, \\ \{0,2\}, \\ \{1,2\} \end{array} \right\}, \left\{ \begin{array}{c} (\{0,1\}, 1.0), \\ (\{0,2\}, 0.5), \\ (\{1,2\}, 1.0) \end{array} \right\} \right)$$

Illustrated graphically:



As now all three information units exist (0, 1 and 2) they can be modified. According to our editing process described at the beginning, there should be only a single *MODIFY*-operation: that is, the last two objects 1 and 2 shall change their shape from *RECTANGLE* to *ELLIPSE*. Position, size, proportions etc. should remain unaltered.

This might look as follows:



As defined in Eq. 4.101 and Eq. 4.65, altering shapes is treated as spatial modification (i. e., *MODIFY_SPATIAL*). Thus $e_3 = (\{1,2\}, \mathit{MODIFY\_SPATIAL})$. The updated event buffer B_4 no longer contains empty events:

$$\begin{aligned} B_4 &= \mathit{push}(B_3, e_3) \\ &= \begin{pmatrix} (\{0\}, \mathit{CREATE}), (\{1\}, \mathit{CREATE}), (\{2\}, \mathit{CREATE}), \\ (\{1,2\}, \mathit{MODIFY\_SPATIAL}) \end{pmatrix} \end{aligned}$$

Spatially modifying units 1 and 2 affects all three pairs of objects in J_3, $\{0,1\}, \{0,2\}$ and $\{1,2\}$. According to our definitions from Eq. 4.220 their temporal connections must be strengthened as follows:

$$\begin{aligned} J_4 &= \mathit{update\_temporal\_relations}(J_3, B_4) \\ &= \left(\{0,1,2\}, \left\{ \begin{matrix} \{0,1\}, \\ \{0,2\}, \\ \{1,2\} \end{matrix} \right\}, \left\{ \begin{matrix} \left(\{0,1\}, w(\{0,1\}) + \Delta w(\{0,1\})\right), \\ \left(\{0,2\}, w(\{0,2\}) + \Delta w(\{0,2\})\right), \\ \left(\{1,2\}, w(\{1,2\}) + \Delta w(\{1,2\})\right) \end{matrix} \right\} \right) \\ &= \left(\{0,1,2\}, \left\{ \begin{matrix} \{0,1\}, \\ \{0,2\}, \\ \{1,2\} \end{matrix} \right\}, \left\{ \begin{matrix} (\{0,1\}, 0.50 + 2^{-3}), \\ (\{0,2\}, 0.25 + 2^{-3}), \\ (\{1,2\}, 0.50 + 2^{-0}) \end{matrix} \right\} \right) \\ &= \left(\{0,1,2\}, \left\{ \begin{matrix} \{0,1\}, \\ \{0,2\}, \\ \{1,2\} \end{matrix} \right\}, \left\{ \begin{matrix} (\{0,1\}, 0.50 + 0.125), \\ (\{0,2\}, 0.25 + 0.125), \\ (\{1,2\}, 0.50 + 1.000) \end{matrix} \right\} \right) \\ &= \left(\{0,1,2\}, \left\{ \begin{matrix} \{0,1\}, \\ \{0,2\}, \\ \{1,2\} \end{matrix} \right\}, \left\{ \begin{matrix} (\{0,1\}, 0.625), \\ (\{0,2\}, 0.375), \\ (\{1,2\}, 1.500) \end{matrix} \right\} \right) \end{aligned}$$

$A_T\langle\mathbb{N}_0, 3, 1.0\rangle$ switches from …

$$\begin{pmatrix} B_3 \\ J_3 \end{pmatrix} = \begin{pmatrix} \big((\emptyset, \varepsilon), (\{0\}, \mathit{CREATE}), (\{1\}, \mathit{CREATE}), (\{2\}, \mathit{CREATE})\big) \\ \left(\{0,1,2\}, \begin{Bmatrix} \{0,1\}, \\ \{0,2\}, \\ \{1,2\} \end{Bmatrix}, \begin{Bmatrix} (\{0,1\}, 0.50), \\ (\{0,2\}, 0.25), \\ (\{1,2\}, 0.50) \end{Bmatrix}\right) \end{pmatrix}$$

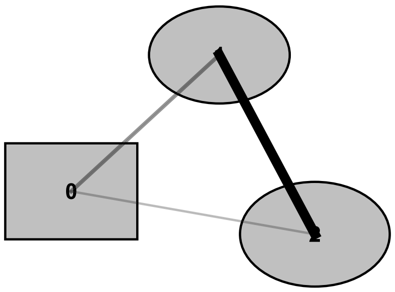
… to:

$$\begin{pmatrix} B_4 \\ J_4 \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} (\{0\}, \mathit{CREATE}), (\{1\}, \mathit{CREATE}), (\{2\}, \mathit{CREATE}), \\ (\{1,2\}, \mathit{MODIFY\_SPATIAL}) \end{pmatrix} \\ \left(\{0,1,2\}, \begin{Bmatrix} \{0,1\}, \\ \{0,2\}, \\ \{1,2\} \end{Bmatrix}, \begin{Bmatrix} (\{0,1\}, 0.625), \\ (\{0,2\}, 0.375), \\ (\{1,2\}, 1.500) \end{Bmatrix}\right) \end{pmatrix}$$

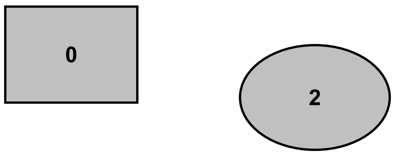
Now the strongest temporal connection exists between modified objects $1, 2$:

$$I_3 = \mathit{normalize}(J_4) = \left(\{0,1,2\}, \begin{Bmatrix} \{0,1\}, \\ \{0,2\}, \\ \{1,2\} \end{Bmatrix}, \begin{Bmatrix} (\{0,1\}, 0.4167), \\ (\{0,2\}, 0.2500), \\ (\{1,2\}, 1.0000) \end{Bmatrix}\right)$$

Illustrated graphically:



The fifth and last step in our sample editing process is *deleting* object 1:



Pushing the respective event $e_4 = (\{1\}, \mathit{DELETE})$ to our current event buffer B_4 results in an overflow. This means, the oldest event $(\{0\}, \mathit{CREATE})$ is removed from the buffer. Thus B_4 turns into:

$$\begin{aligned} B_5 &= \mathit{push}(B_4, e_4) \\ &= \begin{pmatrix} (\{1\}, \mathit{CREATE}), (\{2\}, \mathit{CREATE}), \\ (\{1,2\}, \mathit{MODIFY\_SPATIAL}), (\{1\}, \mathit{DELETE}) \end{pmatrix} \end{aligned}$$

According to Eq. 4.220 *DELETE*-operations leave the weightings of remaining associations unchanged. This means in our case:

$$\begin{aligned} J_5 &= \mathit{update\_temporal\_relations}(J_4, B_5) \\ &= \Big(\{0,2\}, \{\{0,2\}\}, \big\{(\{0,2\}, 0.375)\big\}\Big) \end{aligned}$$

Thus, $A_T\langle \mathbb{N}_0, 3, 1.0\rangle$ switches with the last event of our five-stage editing process from state ...

$$\begin{pmatrix} B_4 \\ J_4 \end{pmatrix} = \left(\begin{array}{c} \begin{pmatrix} (\{0\}, \mathit{CREATE}), (\{1\}, \mathit{CREATE}), (\{2\}, \mathit{CREATE}), \\ (\{1,2\}, \mathit{MODIFY\_SPATIAL}) \end{pmatrix} \\ \left(\{0,1,2\}, \begin{Bmatrix} \{0,1\}, \\ \{0,2\}, \\ \{1,2\} \end{Bmatrix}, \begin{Bmatrix} (\{0,1\}, 0.625), \\ (\{0,2\}, 0.375), \\ (\{1,2\}, 1.500) \end{Bmatrix} \right) \end{array} \right)$$

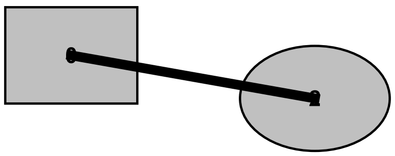
... to final state:

$$\begin{pmatrix} B_5 \\ J_5 \end{pmatrix} = \left(\begin{array}{c} \begin{pmatrix} (\{1\}, \mathit{CREATE}), (\{2\}, \mathit{CREATE}), \\ (\{1,2\}, \mathit{MODIFY\_SPATIAL}), (\{1\}, \mathit{DELETE}) \end{pmatrix} \\ \Big(\{0,2\}, \{\{0,2\}\}, \big\{(\{0,2\}, 0.375)\big\}\Big) \end{array} \right)$$

Since J_5 includes only a single pair of objects $\{0,2\}$ $\mathit{normalize}(J_5)$ results in:

$$I_4 = \mathit{normalize}(J_5) = \Big(\{0,2\}, \{\{0,2\}\}, \big\{(\{0,2\}, 1.0)\big\}\Big)$$

Illustrated graphically I_4 becomes:



Chapter 5

Test

As we know from Sect. 1.2 lack of clarity is inherently part of spatial hypertext. This makes disambiguation an important core feature of spatial parsers. However, resolving ambiguities in spatial hypertext requires knowledge which only hypertext authors can have. Typically, such knowledge is not encoded in single spatial hypertext artifacts. Therefore, conventional (non-adaptive) parsers cannot include such information into their analysis. Due to this conceptional limitation there are several types of structures that cannot be recognized properly, including: (1) ambiguous structures (Sect. 3.1); (2) destroyed structures (Sect. 3.2) and (3) temporal structures (Sect. 3.3).

In order to overcome this issue we suggested in Sect. 3.4 to consider not only spatial and visual properties, but also temporal aspects in spatial parser designs. A spatial parser that is "aware" of previous structures (see "fading" in Sect. 4.4.1 ; pages 93–94) and "knows" about temporal dependencies between information units (see temporal parser in Sect. 4.4.5) could (a) filter out discrete structures most likely seen by human users; (b) complete corrupted structures and (c) detect associations that are purly temporal.

Both, temporal parser as well as fading-feature were implemented in a prototypical spatial hypermedia system, strictly following our theoretical system model from Chapter 4. According to that model, spatial hypermedia systems are compositions of editing systems supporting in the creation of spatial hypertext (Sect. 4.1) and interpretation systems performing structural analysis (Sect. 4.2). Editing systems are mainly determined by workspace models as described in Sect. 4.1.3, whereas interpretation systems are primarily defined by parsing algorithms (Sect. 4.4).

For our research prototype we specified in Sect. 4.1.4 several refinements regarding visual language, object keys, content etc. This has led to default configurations for both workspace model A_W (Eq. 4.66) as well as for parsers A_S (Eq. 4.121), A_V (Eq. 4.183), A_C (Eq. 4.202) and A_T (Eq. 4.232). A full block-diagram of our prototypical spatial hypermedia system can be found in the appendix on page 216. A screenshot of the user interface is given on page 217. This freely configurable reference implementation could be used for researching diverse topics, such as parser performance, optimal system configuration, user behaviour etc.

In this thesis we focus on a single aspect only, namely, parser performance. More specifically, we examine synergies between spatial parser (Sect. 4.4.2), visual parser (Sect. 4.4.3) and temporal parser (Sect. 4.4.5).

Content parsers are not included in our analysis. Initial tests with the content parser from Sect. 4.4.4 suggested, that using it in a productive system would require adaption to user-requirements. For our investigations of parsing performance, however, we expect that tuning of parser settings is not required. For this reason we decided to exclude the content parser from our test.

A second restriction concerns our previously mentioned implementation of structure fading (Sect. 4.4.1 ; pages 93–94). Just like the content parser also fading is not included in our performance analysis. The reason for this is as follows: The longer you are restructuring a spatial hypertext the higher the risk of accidentally or unnoticeably destroying structure. Only then it makes sense to include previous object relations into the analysis (or to reject them if they do not get refreshed). Only then it makes sense to include the fading-effect. Or in other words, the fading-feature will show its real added value only after long time use. Thus, for validation, long-term studies should be preferred over short-term tests. In our analysis, however, we build on short surveys.

In principle this limitation also applies to the evolution of temporal dependencies and therefore on the temporal parser. Most likely, many structures will emerge only after a certain period of time. Nevertheless, there were reasons to believe that the temporal parser would show its benefit also in short-term tests. Perhaps not as obvious as it might be possible after really long working sessions in a productive environment, but still verifiable.

In Sect. 3.4 we claimed, that considering not only spatial and visual properties but also temporal aspects in spatial parser design can lead to a significant increase in parsing accuracy, detection of richer structures and herewith higher parser performance. What still remains to be done, however, is delivering the proof that such a *spatio-temporal parser* really performs better than a conventional spatial parser. This chapter is intended to change that.

We want to show, that spatial parser performance (i. e., accuracy) can be significantly increased when taking into account not only spatial and visual but also temporal object relations.

We expect that the addition of a temporal parser (as defined in Sect. 4.4.5) will shift machine detected structures (encoded in interpretations ; Sect. 4.2.3) significantly closer to what target users (knowledge workers with technical background) intend to express.

5.1 Reference Data Collection

Parsers as we defined them in Sect. 4.4.1 and Sect. 4.4.5 generate formal interpretations $I \in I\langle\Omega\rangle$ (Sect. 4.2.3). Technically, such parse results are complete, weighted graphs of connected information objects. This makes it possible to compare them *numerically.* However, interpretations generated by parsers do not include information about their *accuracy.* Therefore, putting them in direct relation allows no conclusions regarding differences in parser performance. This required us to collect reference data.

Whether something is structure or not is highly subjective, hence reference interpretations cannot be generated artificially. There is also no set of standard structures that could be used as a basis for comparison. For this reason we decided to collect our reference data by surveys in a laboratory. This, however, required an adaption of our research prototype.

5.1.1 Adapted Prototype

When linked together, editing systems and interpretation systems realise interactive structure creation loops that are driven by user activities. This represents the functional core of spatial hypermedia systems and thus formed the basis for our theoretical system model from Chapter 4. We mentioned that already on page 47.

Fig. 5.1 illustrates that for our prototype. According to that, editing systems accept user interface activities (such as keyboard or mouse events etc.) and transform them into edit events $e_k \in E_W$ and workspaces $W_k \in O_W$ (Eq. 4.66). Both e_k and W_k are then converted into “edit steps” (e'_k, D_k), with $e'_k \in E\langle\mathbb{N}_0\rangle$ and $D_k \in D\langle\mathbb{N}_0, 16\rangle$ (Eq. 4.71, Eq. 4.85). Interpretation systems react on such ingoing (e'_k, D_k) by generating interpretations $I_k \in I\langle\mathbb{N}_0\rangle$ (Eq. 4.90). These parse results are finally transmitted back to presentation level as I'_k where they are displayed as graphical overlays on the workspace.

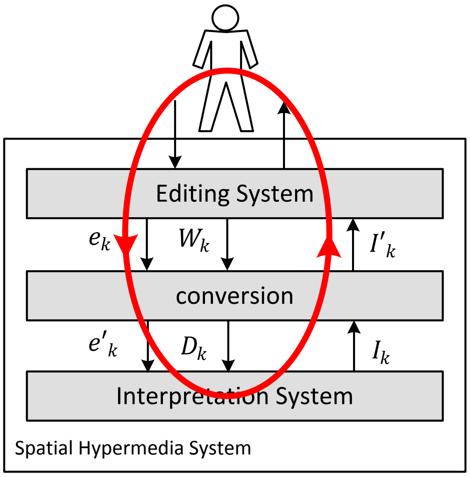


Figure 5.1: When linked together, editing systems and interpretation systems realise interactive structure creation loops, or visual feedback loops, that are initiated by system users. This is the expected behaviour in productive working environments.

This forms a visual feedback-loop where human and machine "interact" to gradually develop meaningful visual structure. Although this is the expected behaviour in a productive working environment it is not what we want when collecting reference data. For this reason, two essential modifications to our research prototype were needed.

Firstly, we enhanced our system with logging-functionality. This is why the adjusted system model illustrated in Fig. 5.2 includes, in addition to the already known components, another module labeled with "log". Plugged in between conversion layer and interpretation system this logging module records ingoing streams of edit steps (e'_k, D_k). This mechanism can be used to store full working sessions in persistent memory (e. g., in a log-file). Loading and "replaying" recorded editing processes finally allows to "simulate" user behaviour in a virtual test environment. This makes it possible to repeat one an the same user session with different parser configurations.

Secondly, we had to make sure that test persons were not influenced by machine generated feedback. This was achieved by deactivating or rather by excluding the interpretation system from the test prototype. Thus, the prototype application we used in our surveys included no parser functionality. The system model given in Fig. 5.3 illustrated that. For the sake of completeness, it should be noted that this stripped-down version of our system model rather describes visual editors with integrated logging functionality than full-fledged spatial hypermedia systems. For this see also our considerations in Chapter 3 or Sect. 4.2. However, for our survey this was of no importance.

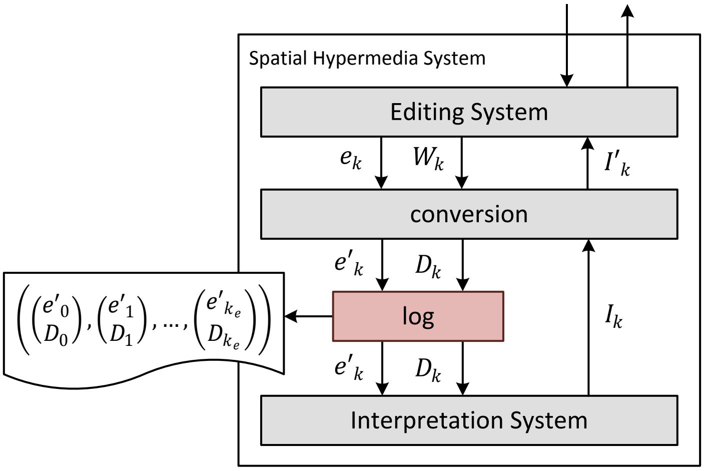

Spatial Hypermedia System
Editing System
e_k
W_k
I'_k
conversion
e'_k
D_k
log
I_k
$\left(\binom{e'_0}{D_0},\binom{e'_1}{D_1},\ldots,\binom{e'_{k_e}}{D_{k_e}}\right)$
Interpretation System

Figure 5.2: Spatial hypermedia system model extended by logging functionality. Streams of edit steps (e'_k, D_k) passing the logging system on their way from conversion to interpretation system are recorded in persistent memory. Our prototype application writes them to log-files.

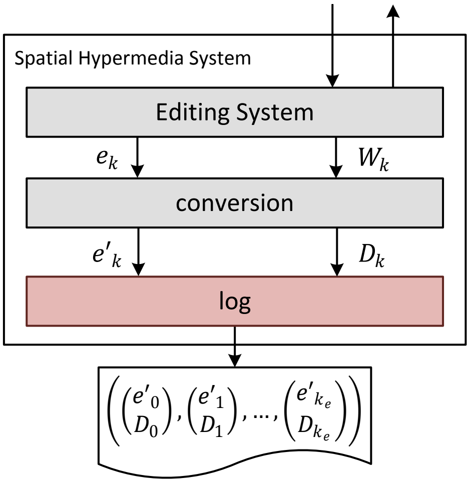

Spatial Hypermedia System
Editing System
e_k
W_k
conversion
e'_k
D_k
log
$\left(\binom{e'_0}{D_0},\binom{e'_1}{D_1},\ldots,\binom{e'_{k_e}}{D_{k_e}}\right)$

Figure 5.3: Stripped-down version of our system model from Fig. 5.2 without interpretation system. This is rather an advanced visual editor with integrated logging functionality than a fully-fledged spatial hypermedia system.

5.1.2 Survey

The second challenge, besides adapting our research prototype, was to establish realistic test conditions. This required to find persons who were willing to participate in a laboratory test. Only if probands participate voluntarily in a survey, there is a chance to get realistic and therefore useful results. When subjects, however, feel unwell in a given test situation, then free development of thoughts is excluded right from the start, which inevitably falsifies test results. This is especially true for *creative processes*, such as developing spatial hypertexts. Thus, particular attention had to be paid to ensuring that subjects had, right from the beginning, a positive attitude to their task. There are lots of factors which can have a negative impact on a test person's motivation, including biorhythm, stress or test duration. This makes the creation of optimal test conditions a challenge.

For our survey we proceeded as follows: Firstly, potential test candidates were asked to attend a small experiment, on a voluntary basis. We did that only on working days with low time pressure and at a biorhythmically optimal time. It turned out that our estimated test duration seemed to play an essential role for probands, when deciding whether to participate or not. An average duration of 20 minutes was accepted by most test persons. Longer tests, however, had a rather deterrent effect. For this reason, and because we believed that such a short timeframe still would be enough for showing the desired temporal effects, we decided to adhere to an average test duration of 20 minutes and an upper time limit of half an hour.

In total, we were able to convince 50 persons to join our survey. All participants had a technical background in computer sciences, ranging from Bachelor- to PhD-level. Therefore, it could be assumed that they knew how to use a visual modelling tool. However, it should be pointed out that none of these 50 people had used a spatial hypermedia system before.

Once a person had agreed to participate, he was asked to take a seat in a special laboratory room prepared for the test run. Fig. 5.4 shows a photo of the setup. The participant sat down in front of a 65-inches computer monitor (that was still switched off at that point in time), and was then informed about the basic test conditions. After that, we had to make sure that the test person was familiar with the prototype's user interface. Simply playing around with or (even worse) incorrect usage of the prototype's visual tools would have unnecessarily falsified our test results. Even though we had implemented only a small number of common GUI-features in our test application (such as zooming, panning, scaling etc.), we decided to still demonstrate each of those features to every single test person. For these purposes we prepared a simple demo hypertext that was formed from random objects only. These objects neither had any visual nor semantic relation with the hypertext used for the test. During

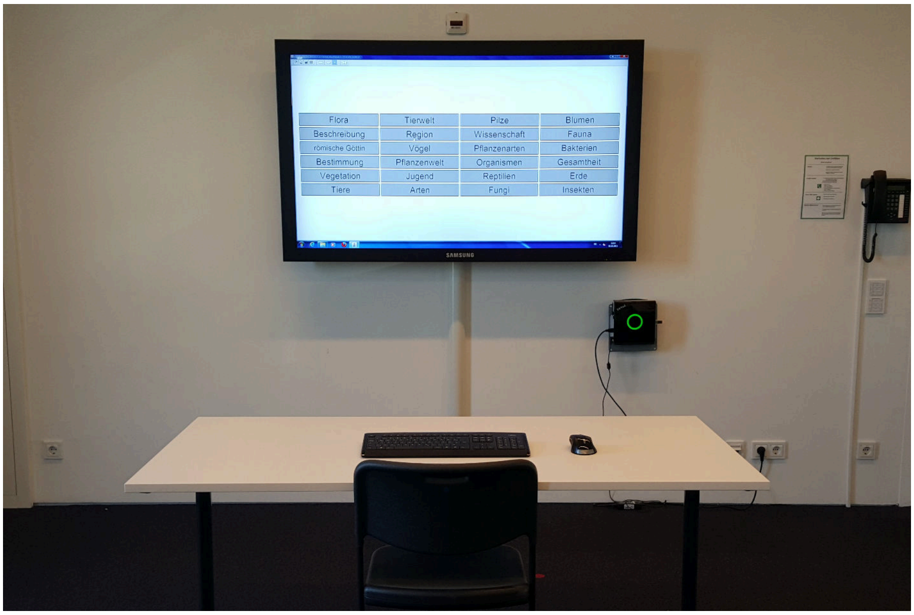
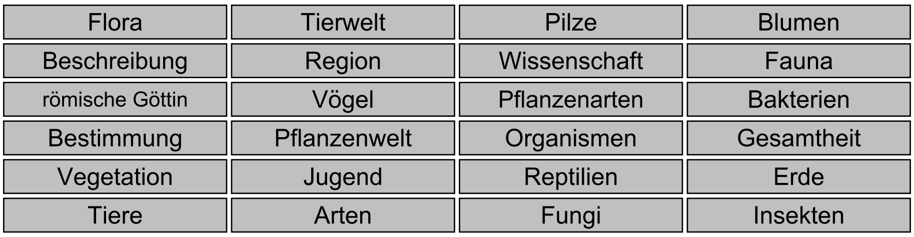


Figure 5.4: experimental setup – core hardware components: wireless keyboard/mouse, mini-PC (Dual-Core Processor 1.65 GHz; 2GB DDR3; HDMI), OS: Windows 7 (32 Bit), 65-inches LCD-monitor (diagonal 163cm) with an optimal resolution of 1920 × 1080 at 60Hz

demonstration, subjects were requested to actively use each interface feature to ensure that they had really understood its functionality. After probands had explicitly confirmed that there were no open questions regarding interface usage, the actual test hypertext was loaded into the prototype application. Fig. 5.5 shows the initial display as seen by our subjects.

Each test person was presented 24 rectangular objects, each labeled with a term related to a common subject area. Participants were then asked to re-structure those objects so that the resulting diagram reflected their basic comprehension of the given terms and their relations. This means, subjects should re-arrange,



Flora	Tierwelt	Pilze	Blumen
Beschreibung	Region	Wissenschaft	Fauna
römische Göttin	Vögel	Pflanzenarten	Bakterien
Bestimmung	Pflanzenwelt	Organismen	Gesamtheit
Vegetation	Jugend	Reptilien	Erde
Tiere	Arten	Fungi	Insekten

Figure 5.5: initial display as seen by subjects

scale, color etc. the given terms to express their individual, structural understanding of the given topic area. Here, participants were explicitly told to continue with their work until the result would make most sense to them.

Main design decisions for the sample hypertext in Fig. 5.5 included the following considerations:

Firstly, we intended to avoid unnecessary cognitive overhead and tried to keep up motivation of test candidates. For these reasons we selected only a relatively small number of terms (only 24) from a common and, to our understanding, easy to understand subject area ("flora and fauna") and translated them into the participant's native language (here: German). In concrete terms, we took the first random Wikipedia-article which included only general, basic knowledge and selected 24 characteristic terms.

Another essential design criterion refers to the task's level of difficulty. The given structuring problem should not have a common solution (in form of clusters, trees or tables, for instance). Otherwise there would have been the danger of getting results of a single type only. This is why we selected partly ambiguous terms which allowed for alternative interpretations. A good example is the term "Bestimmung", which could mean both "measurement" but also "destiny".

In addition we intended to make sure, that test persons only started interacting with the visual medium when they had already established a first structural understanding of the given "flora/fauna"-topic. Interface activities without a real (structural) meaning would have led to wrong results (at least in the specified timeslot of at maximum half an hour). This is why all 24 terms were initially arranged in a two-dimensional, rectangular grid. That grid of terms touched both left and right border of the display, but still left enough space in the top and the bottom region of the screen (see Fig. 5.4). Thus, users did not need to zoom out to view all 24 terms or to create empty workspace. Therefore, we could assume with high probability that test persons would begin structuring right from the start. So, except for purely "cosmetic" changes made to the display (in order to make it look "nice"), most of our participant's activities must have implied structural meaning.

Finally it was intended to avoid, that the initial display suggests a potential solution to the term-structuring problem. To achieve that, object shape and color were chosen as neutral as possible. Thus our grey rectangles should not have any semantic association with the subject "flora/fauna". In addition, all 24 terms were arranged such that none of them had an immediate semantic relationship with its surrounding neighbors. Thus, the initial spatial layout should not allow for drawing any conclusions about semantic relationships. With this we tried to maximize the visual difference between initial display and potential solution-hypertexts.

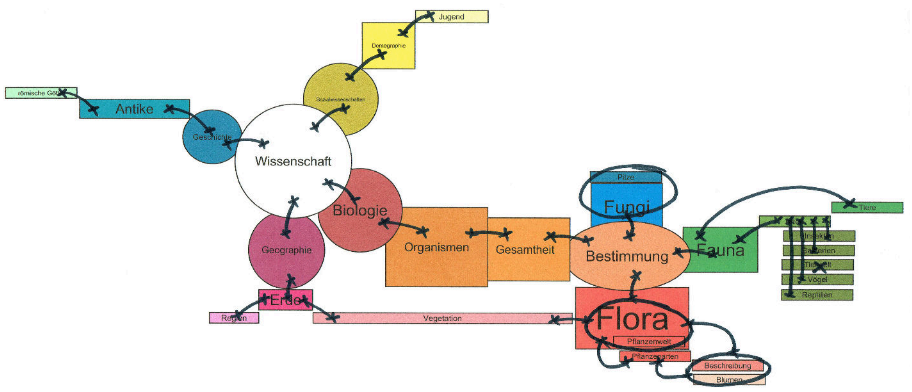


Figure 5.6: sample feedback provided by one of our participants

Having finished their structuring task, test persons were finally asked to tell the researcher explicitly what they intended to express. For this purpose they received a printout of their work and a pen. They were then asked to highlight where they can see associations between objects (by drawing lines, circles etc.). Fig. 5.6 shows an example of such a marked printout. Note, that Fig. 5.6 includes no explicit numerical information about the strength of object relations. The drawing only describes general connections between (groups of) objects. The reason for this lies in spatial hypertext's *implicit* nature (Sect. 1.2). Unlike our parsers, humans may find it cognitively difficult to express implicit structure as explicit numbers. Thus, precision of human user feedback is necessarily lower than precision of machine-generated feedback. This had significant impact on our evaluation algorithm that will be discussed in Sect. 5.2.

User feedback that we received in this graphical form had to be quantified. For these purposes we wrote a small visual analysis tool that allowed us to manually transform handwritten user feedback into digital, weighted graphs; that is, into interpretations $I\langle\mathbb{N}_0\rangle$. In these graphs we used numerical weightings > 0.0 to express object connections and hence structures. Different weightings indicated different levels of abstraction. That is, the higher the abstraction-level the smaller the assigned weighting and vice versa. When required, this allowed us to encode multi-level structures. Note, that our spatial parser does the same. This formed the reference data for our automated analysis.

Finally, we can summarize the survey-process as follows: First, test persons interact with our adapted prototype from Sect. 5.1.1 to solve a term-structuring task. Corresponding streams of edit steps (e'_k, D_k) are recorded in log-files. Having finished their assignment participants provide explicit graphical feedback on their intended structures. This handwritten user feedback is finally quantified as reference interpretations I_{ref}. Fig. 5.7 illustrates that graphically.

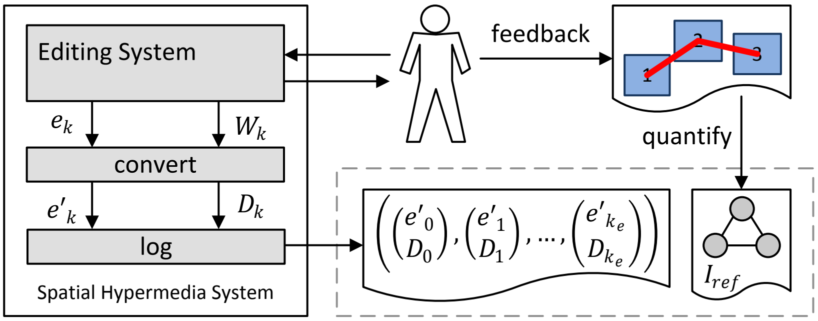

Editing System
e_k
W_k
convert
e'_k
D_k
log
Spatial Hypermedia System
feedback
1
2
3
quantify
$\left(\binom{e'_0}{D_0},\binom{e'_1}{D_1},\ldots,\binom{e'_{k_e}}{D_{k_e}}\right)$
I_{ref}

Figure 5.7: Survey-process overview

5.2 Virtual Test Run

Once we had collected both from a test person, logged edit process (i. e., what the participant did) and reference interpretation (i. e., what the participant intended to express), we proceeded as follows:

Firstly, an interpreter was instantiated that was build from only temporal, spatial and visual parser. Fading and implicit merging were deactiated, since those features were not needed for the test run. Once we had a runnable interpretation system, both, reference interpretation graph as well as the recorded stream of edit steps were loaded into memory. The stream was then passed through all three parser components. Expressed formally . . .

29 Virtual Test Studio - main procedure

```
1: $I_{ref} \leftarrow$ load reference interpretation from file
2: $\left(\binom{e_0}{D_0},\binom{e_1}{D_1},\ldots,\binom{e_n}{D_n}\right) \leftarrow$ load edit process from file
3: $I_T, I_S, I_V \leftarrow I_\varepsilon$
4: for $i = 0$ to $n$ do
5:     $I_T \leftarrow \textit{interprete}\left(A_T, (e_i, D_i)\right)$                                   ▷ Eq. 4.232
6:     $I_S \leftarrow \textit{interprete}\left(A_S\langle\mathbb{N}_0, 16, 0.0, 1.0\rangle, (e_i, D_i)\right)$        ▷ Eq. 4.120
7:     $I_V \leftarrow \textit{interprete}\left(A_V\langle\mathbb{N}_0, 16, 0.0, 1.0\rangle, (e_i, D_i)\right)$        ▷ Eq. 4.182
8: end for
9: $\left(\binom{(T_0, S_0, V_0),}{d_0},\binom{(T_1, S_1, V_1),}{d_1},\ldots,\binom{(T_{230}, S_{230}, V_{230}),}{d_{230}}\right) \leftarrow \textit{test}\begin{pmatrix}I_{ref},\\ I_T,\\ I_S,\\ I_V\end{pmatrix}$
```

In Alg. 29 single parser runs are symbolized by $I \leftarrow \mathit{interprete}(A, (e_i, D_i))$. Here, A is a parser instance and (e_i, D_i) is the $(i+1)$-th step in an edit process.

Resulting temporal, spatial and visual interpretations I_T, I_S, I_V were merged together in varying ratios $(T, S, V) \in \Delta^2$ with an increment of five percent. This means, weighting factors T, S, V were multiples of 0.05 and summed up to 1.0. Each merge result I_{merge} was then compared with the given reference interpretation I_{ref}. For this comparison we used a statistical divergence function which could tell us numerically how close a merged interpretation graph I_{merge} (i. e., the model) came to the respective reference interpretation graph I_{ref} (i. e., our observation). The smaller such numerical divergence values were, the closer the parse result came to what a user intended to express. For example, an optimal value of 0.0 indicated that model and observation were identical; that is, our merge result represented exactly what the user had in mind.

30 $\mathit{test}\colon \big(I\langle\Omega\rangle\big)^4 \rightharpoonup \mathit{List}\langle\Delta^2 \times \mathbb{R}_0^+\rangle,\ \big(I_{ref}, I_T, I_S, I_V\big) \mapsto \mathit{list}_{result}$

```
1: $list_{result} \leftarrow list_\varepsilon$
2: for $i = 0$ to 100 step 5 do
3:     for $j = 0$ to $(100 - i)$ step 5 do
4:         $k \leftarrow 100 - i - j$
5:         $I_{merge} \leftarrow merge\Big(\big(I_T, I_S, I_V\big), \big(i/100, j/100, k/100\big)\Big)$    ▷ Alg. 7
6:         $d \leftarrow compare\big(I_{ref}, I_{merge}\big)$    ▷ Alg. 31
7:         $list_{result} \leftarrow add\Big(list_{result}, \Big(\big(i/100, j/100, k/100\big), d\Big)\Big)$    ▷ Eq. 4.92
8:     end for
9: end for
10: return $list_{result}$
```

Our definition of $I\langle\Omega\rangle$ from Eq. 4.90 allows us to identify different structure levels, that is, we can distinguish between collections of structures with different weight ranges. This makes it possible to iterate through interpretations from the strongest to the weakest associations and to step-by-step extract collections of structures with increasing extension level. We denote the number of such levels as *struct_level_count* and formally define it as follows:

$$\begin{aligned} &\mathit{struct\_level\_count}\colon I\langle\Omega\rangle \to \mathbb{R}_0^+, \\ &\qquad (U, A, w) \mapsto \Big|\big\{w(a)\big|a \in A \wedge w(a) \notin \{0, \varepsilon\}\big\}\Big| \end{aligned} \tag{5.1}$$

Essentially, *struct_level_count* accepts an interpretation-graph $(U, A, w) \in I\langle\Omega\rangle$ and determines the number of different weightings $w(a) \notin \{0, \varepsilon\}$ assigned to association $a \in A$. Thus, each weight > 0 identifies an extension level of structure.

As an example, for an $I_{sample} = (U, A, w) \in I\langle\{u_0, u_1, u_2\}\rangle$ defined as ...

$$U = \{u_0, u_1, u_2\} \; ; \; A = \binom{U}{2} = \left\{ \begin{matrix} \{u_0, u_1\}, \\ \{u_0, u_2\}, \\ \{u_1, u_2\} \end{matrix} \right\} \; ; \; w = \left\{ \begin{matrix} \left(\{u_0, u_1\}, 0.50\right), \\ \left(\{u_0, u_2\}, 0.00\right), \\ \left(\{u_1, u_2\}, 1.00\right) \end{matrix} \right\} \tag{5.2}$$

... *struct_level_count*(I_{sample}) would identify 2 levels of weightings $\notin \{0, \varepsilon\}$. These are: a top-level with weight 1.00 and a bottom-level with weight 0.50.

Index-based access on these levels can be achieved via *struct*(I_{sample}, i):

$$\begin{aligned} &\mathit{struct}\colon \left(I\langle\Omega\rangle \times \mathbb{R}_0^+\right) \to I\langle\Omega, \{0,1\}\rangle, \; \left((U, A, w), i\right) \mapsto \left(U, A, w'\right), \\ &\quad w' : A \to \{0, 1\}, \\ &\quad \forall a \in A : w'(a) = \begin{cases} 0 \text{ , if } w(a) \in \{0, \varepsilon\} \\ 1 \text{ , else if } \left| \left\{ w\left(a'\right) \;\middle|\; \begin{matrix} a' \in A \\ \wedge\, w\left(a'\right) \notin \{0, \varepsilon\} \\ \wedge\, w\left(a'\right) < w(a) \end{matrix} \right\} \right| \geq i \\ 0 \text{ , else} \end{cases} \end{aligned} \tag{5.3}$$

Essentially, *struct*(I, i) takes an interpretation $I \in I\langle\Omega\rangle$ and a structure level index $i \geq 0$ and filters out connected sub-graphs with weights that are among the $($*struct_level_count*$(I) - i)$ - biggest weightings in I. Results are encoded as binary $I\langle\Omega, \{0, 1\}\rangle$.

Let us also illustrate this with an example: when we take I_{sample} from Eq. 5.2 again, then for structure level indices ...

$$0 \leq i < \mathit{struct\_level\_count}(I_{sample}) = 2$$

... *struct* $\left(I_{sample}, i\right)$ will return:

$$\mathit{struct}\left(I_{sample}, 0\right) = \left(\{u_0, u_1, u_2\}, \left\{ \begin{matrix} \{u_0, u_1\}, \\ \{u_0, u_2\}, \\ \{u_1, u_2\} \end{matrix} \right\}, \left\{ \begin{matrix} \left(\{u_0, u_1\}, 1\right), \\ \left(\{u_0, u_2\}, 0\right), \\ \left(\{u_1, u_2\}, 1\right) \end{matrix} \right\} \right)$$

$$\mathit{struct}\left(I_{sample}, 1\right) = \left(\{u_0, u_1, u_2\}, \left\{ \begin{matrix} \{u_0, u_1\}, \\ \{u_0, u_2\}, \\ \{u_1, u_2\} \end{matrix} \right\}, \left\{ \begin{matrix} \left(\{u_0, u_1\}, 0\right), \\ \left(\{u_0, u_2\}, 0\right), \\ \left(\{u_1, u_2\}, 1\right) \end{matrix} \right\} \right)$$

For structure indices $i \geq$ *struct_level_count*(I_{sample}) function *struct* $\left(I_{sample}, i\right)$ would return graphs with zero-weightings only (i. e., $\forall a \in A : w'(a) = 0$). This case, however is of no relevance for our following definition of *compare*.

Based on *struct_level_count* (Eq. 5.1) and *struct* (Eq. 5.3) we can define the *compare*-algorithm used in Alg. 30 (line 6) as follows:

31 $compare : \big(I\langle\Omega\rangle \times I\langle\Omega\rangle\big) \rightharpoonup \mathbb{R}_0^+,\ \big(I_{ref}, I_{compare}\big) \mapsto div$

```
Require: $struct\_level\_count\big(I_{ref}\big) > 0 \wedge struct\_level\_count\big(I_{compare}\big) > 0$
 1: $d_{sum} \leftarrow 0$
 2: $i \leftarrow struct\_level\_count\big(I_{compare}\big) - 1$    ▷ Eq. 5.1
 3: for $j = \Big(struct\_level\_count\big(I_{ref}\big) - 1\Big)$ to 0 step −1 do
 4:     $d_{min} \leftarrow divergence\Big(struct\big(I_{ref}, j\big), struct\big(I_{compare}, i\big)\Big)$    ▷ Alg. 32
 5:     for $k = (i-1)$ to 0 step −1 do    ▷ Eq. 5.3
 6:         $d \leftarrow divergence\Big(struct\big(I_{ref}, j\big), struct\big(I_{compare}, k\big)\Big)$
 7:         if $d < d_{min}$ then
 8:             $d_{min} \leftarrow d$
 9:             $i \leftarrow k$
10:         end if
11:     end for
12:     $d_{sum} \leftarrow d_{sum} + d_{min}$
13: end for
14: return $\Big(d_{sum}/struct\_level\_count\big(I_{ref}\big)\Big)$    ▷ Eq. 5.1
```

This is best explained with an example. Let our previous I_{sample} from Eq. 5.2 be our reference interpretation. Let us also assume, that $I_{compare} = (U, A, w)$ is defined by ...

$$U = \{u_0, u_1, u_2\}\ ;\ A = \binom{U}{2} = \begin{Bmatrix} \{u_0, u_1\}, \\ \{u_0, u_2\}, \\ \{u_1, u_2\} \end{Bmatrix}\ ;\ w = \begin{Bmatrix} \big(\{u_0, u_1\}, 0.25\big), \\ \big(\{u_0, u_2\}, 0.00\big), \\ \big(\{u_1, u_2\}, 0.75\big) \end{Bmatrix} \tag{5.4}$$

$compare\,(I_{sample}, I_{compare})$ traverses both I_{sample} and $I_{compare}$ in a top-down fashion from structures with the strongest associations down to an extension level where all associations are included (i. e., all associations $a : w(a) \notin \{0, \varepsilon\}$). It identifies: (a) $struct(I_{compare}, 1)$ as the expansion stage which comes the closest to $struct(I_{sample}, 1)$ and (b) $struct(I_{compare}, 0)$ as the one coming the closest to $struct(I_{sample}, 0)$. The resulting divergence value is calculated as ...

$$\frac{\left(\begin{array}{r} divergence\Big(struct\big(I_{sample}, 1\big), struct\big(I_{compare}, 1\big)\Big) \\ +divergence\Big(struct\big(I_{sample}, 0\big), struct\big(I_{compare}, 0\big)\Big) \end{array}\right)}{struct\_level\_count\big(I_{sample}\big)} = \frac{\begin{pmatrix} 0 \\ +0 \end{pmatrix}}{2} = \frac{0}{2} = 0$$

$compare\,(I_{sample}, I_{compare}) = 0$ indicates that Eq. 5.4 allows to filter out all discrete structures that are also encoded in Eq. 5.2.

Our routine for divergence calculation, as we use it in Alg. 31 (in lines 4, 6), builds on the Kullback Leibler (KL) divergence [40] and works for any $I\langle\Omega\rangle$, not only for binary $I\langle\Omega, \{0,1\}\rangle$:

32 $divergence: \big(I\langle\Omega\rangle \times I\langle\Omega\rangle\big) \rightharpoonup \Big(\mathbb{R}_0^+ \cup \{\varepsilon\}\Big),\ \big((U,A,w_0),(U,A,w_1)\big) \mapsto d$

```
1:  $distribution_0, distribution_1 \leftarrow list_\varepsilon$
2:  $weightSum_0, weightSum_1 \leftarrow 0.0$
3:  for all $a \in A$ do
4:      if $w_0(a), w_1(a) \neq \varepsilon$ then
5:          $distribution_0 \leftarrow add\big(distribution_0, w_0(a)\big)$        ▷ Eq. 4.92
6:          $distribution_1 \leftarrow add\big(distribution_1, w_1(a)\big)$
7:          $weightSum_0 \leftarrow weightSum_0 + w_0(a)$
8:          $weightSum_1 \leftarrow weightSum_1 + w_1(a)$
9:      end if
10: end for
11: if $empty(distribution_0) = TRUE$ then        ▷ Eq. 4.95
12:     return $\varepsilon$
13: end if
14: if $weightSum_0 = 0.0$ then
15:     if $weightSum_1 = 0.0$ then
16:         return 0.0
17:     end if
18:     return $\varepsilon$
19: else if $weightSum_1 = 0.0$ then
20:     return $\varepsilon$
21: end if
22: $p_0, p_1 \leftarrow list_\varepsilon$
23: for $i = 0$ to $\big(size(distribution_0) - 1\big)$ do        ▷ Eq. 4.93
24:     $p_0 \leftarrow add\big(p_0, get(distribution_0, i)/weightSum_0\big)$        ▷ Eq. 4.92, Eq. 4.94
25:     $p_1 \leftarrow add\big(p_1, get(distribution_1, i)/weightSum_1\big)$
26: end for
27: return $\Big(KL_{symmetric}\big(smoothen(p_0), smoothen(p_1)\big)\Big)$        ▷ Eq. 4.209
```

The trick here is to treat weighting functions w_0, w_1 as *distributions* of weights $w_0(a), w_1(a)$ over associations $a \in A$. When transformed into (discrete) probability distributions p_0, p_1 they allow for calculating the divergence between (U, A, w_0) and (U, A, w_1) as ...

$$KL_{symmetric}\big(smoothen(p_0), smoothen(p_1)\big)$$

Here, $KL_{symmetric}$ represents the symmetrized Kullback Leibler divergence from Eq. 4.209. $smoothen(p)$ ensures that p includes no zeros.

The following example will give you an impression of the power of Alg. 32.

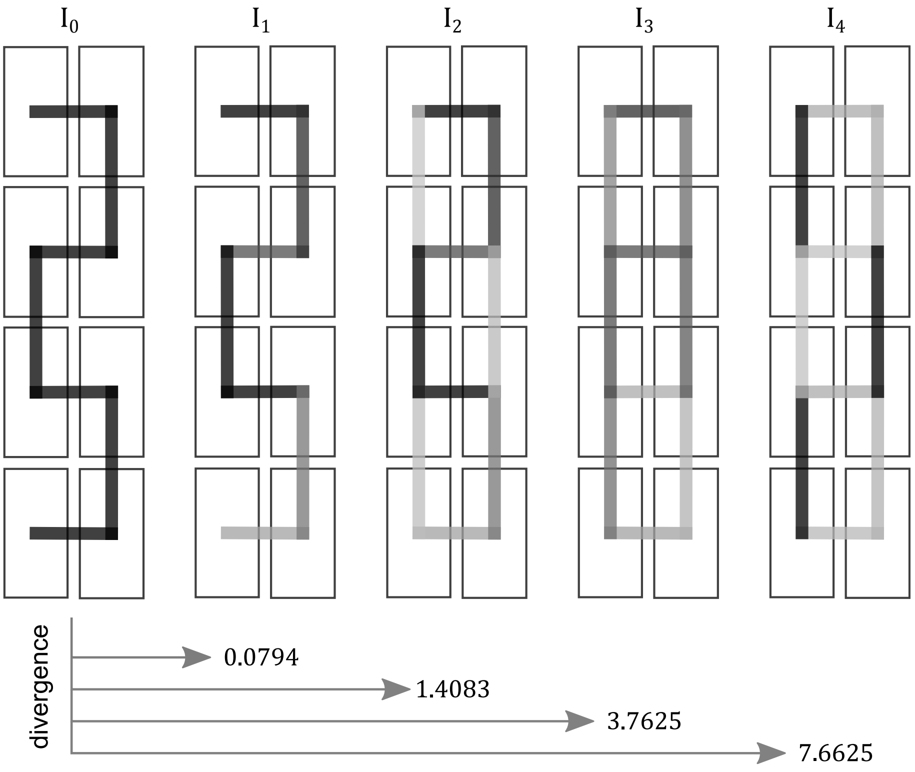


Figure 5.8: Five sample interpretation graphs $I_0, \ldots, I_4$ and their divergence values. Varying brightness of line colors symbolizes different weightings. Divergences were calculated between interpretation I_0 on the left and I_1, I_2, I_3, I_4 on the right.

Fig. 5.8 illustrates five sample interpretation graphs $I_0, \ldots, I_4$, each a network of eight white rectangles. Like in previous illustrations of interpretation graphs, also here varying weights are expressed with visual line properties. The stronger the connection between two rectangles the higher the assigned weighting and the darker the painted line. Line thickness and opacity are the same for all five illustrations. Right below these graphs, you can see the divergences Alg. 32 calculates between I_0 and I_i (for $i \in \{1, \ldots, 4\}$). According to that, ...

$$\begin{aligned}
divergence\,(I_0, I_1) &= 0.0794\\
divergence\,(I_0, I_2) &= 1.4083\\
divergence\,(I_0, I_3) &= 3.7625\\
divergence\,(I_0, I_4) &= 7.6625
\end{aligned}$$

Note, that these values were rounded to four places after the decimal point.

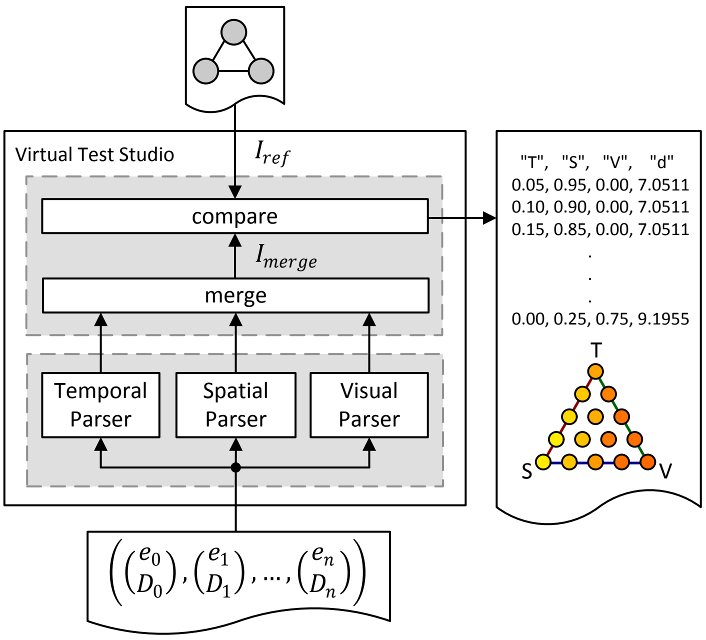


Figure 5.9: Overview of the virtual test run

In conclusion, the overall test run can be summarized as follows:

Firstly, both logged edit process (i. e., what the participant did) and reference interpretation (i. e., what the participant intended to express) are loaded into memory. The recorded sequence of edit steps is then passed through temporal, spatial and visual parser. Resulting interpretations are merged together in varying ratios and with an increment of five percent. Comparing these mixtures with the given reference interpretation results in numerical divergence values. These values indicate how close merged parse results come to user intention and therefore can be used for evaluation purposes. A graphical overview of the virtual test run can be found in Fig. 5.9.

5.3 Analysis

Our virtual test run resulted in 50 CSV-lists (one per test person). Each of those lists included 231 tuples of weighting factors for temporal, spatial and visual parser together with the respective divergence value (the number 231 comes from our chosen weighting factor increment of five percent).

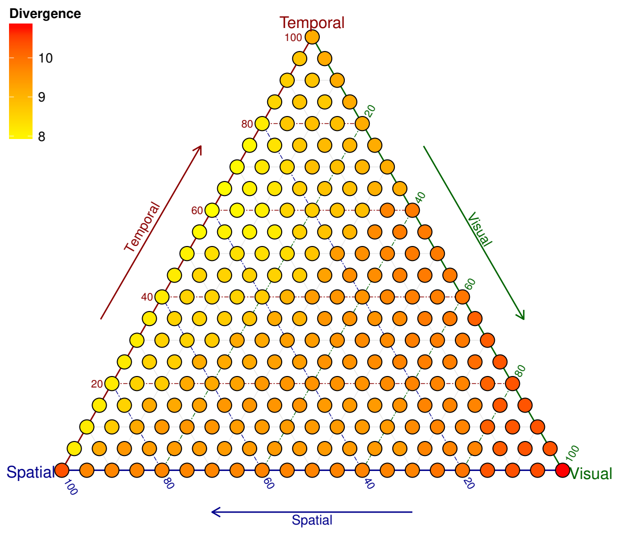


Figure 5.10: sample ternary plot

Demonstrated with an example, each of those 50 tabular listings was of the following form:

```
#0   "T" , "S" , "V" , "d"
#1   0.05, 0.95, 0.00, 7.05113242479466400000
#2   0.10, 0.90, 0.00, 7.20835108739322600000
#3   0.15, 0.85, 0.00, 7.32986400847126800000
. . .
#231 0.00, 0.25, 0.75, 9.19552448851569300000
```

“T”, “S”, and “V” label the weighting factors for temporal, spatial and visual parser by mixed ratios. The divergence of each ratio is labeled as “d”.

As an initial step, we illustrated our results as colored ternary diagrams. In ternary plots the ratios of three variables correspond to positions within a triangle. As such, they are well suited for visualizing the given triples of weighting factors. Figure 5.10 shows an example of such a colored ternary diagram. The colored circles in the triangle represent different parser combinations or triples $(T, S, V) \in \Delta^2$ and therefore are evenly distributed over the simplex. The triangle's vertices ($(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$) are special cases where only spatial, temporal, or visual parser are included. The divergence values d assigned to the ratios throughout the triangle are expressed as colors with a color scale ranging from yellow to red. Pure yellow indicates the smallest divergence value whereas pure red the biggest one in a sample; brighter colors indicate better results.

Drawing ternary plots for each of the 50 sample cases has not shown a uniform picture. We could not spot a single connected area of optimal ratios of parser configurations throughout all 50 diagrams. In contrary, we have indicated various patterns in analyzing the ternary plots. Nevertheless this analysis has provided some important insights regarding the relationship of spatial, visual, and temporal parsers:

The more explicit a spatial structure was (e. g., in form of lists, trees, mind maps etc.) the more the optimum has moved towards $(T, S, V) = (0.0, 1.0, 0.0)$ and thus put the focus on the spatial parser. The more dominant visual attributes have been used in a meaningful manner (e. g., for highlighting different object categories etc.) the more the optimum has been shifted towards $(T, S, V) = (0.0, 0.0, 1.0)$ and hence towards the visual parser. In other words, the easier it was for the spatial and visual parsers to detect clear structures, the more weight they have got in the optimal solution. The better visual or spatial heuristics fitted a test person's individual way of visual expression the lower the required support by the temporal parser. This is the expected behaviour and hence not surprising. It is interesting to note though, that even in such extreme cases pure spatial and visual parser could not outperform the spatio-temporal parser; that is, there always was at least one temporal mixture which performed at least as good as the best without temporal component.

Furthermore, we made the important observation that the impact of the temporal parser was higher for more ambiguous and "uncommon" structures. We assume the reason to be in the spatial and visual parser's limitations and lack of heuristic definitions. Thus, the temporal parser became important in cases when spatial and visual parser failed to recognize accurate structure. One could argue that the temporal parser complemented spatial and visual parser beyond their limits. To some extent this could eliminate the need for manually tuning parsers to specific users. Instead, limited precision caused by inadequate parser configurations or heuristics could be partly compensated by adding our temporal parser. Even though this could not reach the same high level of accuracy as it might be possible with highly customized parsers, the addition of our temporal parser still provided better results than pure spatial or visual parser; while being completely independent of user preferences and context of application.

Although this is the most promising finding related to our temporal parser it still does not *proof* our hypothesis from page 179. Ternary plots are good for initial assessment but do not allow for reliable conclusions about significance. A statistical approach is required instead.

5.3.1 Statistical Method

To compare spatio-visual and spatio-temporal parser statistically we had to determine optimal merging ratios, with and without temporal component.

We regard combinations of temporal, spatial and visual parser as being *optimal* if they generate results at consistently high level of accuracy. It is neither optimal to constantly produce output of low quality nor to generate perfect results for a single person only. Divergence values rather should be as low as possible while not varying between different users. Ideally they are *always zero*.

Since our ternary plots did not show a uniform picture it was not possible to identify single connected areas of optimal ratios. Thus, we had to determine temporal and non-temporal optima numerically rather than visually.

Our 50 CSV-lists contained 231 data sets of the following form:

$$\big((T, S, V), (d_0, d_1, \ldots, d_{49})\big) \in \left(\Delta^2 \times \left(\mathbb{R}_0^+\right)^{50}\right)$$

Here, (T, S, V) is a merging ratio and $(d_0, d_1, \ldots, d_{49})$ is a sequence of 50 divergence values. Let us denote such binary tuples as configuration $\underline{\text{C}}$andidates.

21 of these 231 candidates include *no* temporal component (i. e., $T = 0$):

$$\begin{aligned} C_{NT00} &= \big((0.00, 1.00, 0.00), (d_{0000}, d_{0001}, \ldots, d_{0049})\big) \\ C_{NT01} &= \big((0.00, 0.95, 0.05), (d_{0100}, d_{0101}, \ldots, d_{0149})\big) \\ &\vdots \\ C_{NT20} &= \big((0.00, 0.00, 1.00), (d_{2000}, d_{2001}, \ldots, d_{2049})\big) \end{aligned}$$

For 210 candidates, on the other hand, T is greater than zero:

$$\begin{aligned} C_{T000} &= \big((1.00, 0.00, 0.00), (d_{00000}, d_{00001}, \ldots, d_{00049})\big) \\ C_{T001} &= \big((0.95, 0.05, 0.00), (d_{00100}, d_{00101}, \ldots, d_{00149})\big) \\ &\vdots \\ C_{T209} &= \big((0.05, 0.00, 0.95), (d_{20900}, d_{20901}, \ldots, d_{20949})\big) \end{aligned}$$

In order to identify temporal and non-temporal “winner”-configurations we had to make these candidates comparable. For this we used the following trick:

We calculated for each $C_{NT} \in \{C_{NT00}, ..., C_{NT20}\}$ and $C_T \in \{C_{T000}, ..., C_{T209}\}$ *Median* and *Standard Deviation* of their assigned divergence values and mapped them to value range $[0, 1]$. For this we used the following equations:

$$\begin{aligned} median_{mapped} &= (median_{candidate} - median_{min}) \,/\, (median_{max} - median_{min}) \\ stddev_{mapped} &= (stddev_{candidate} - stddev_{min}) \,/\, (stddev_{max} - stddev_{min}) \end{aligned}$$

This allowed us to represent both $\{C_{NT00}, ..., C_{NT20}\}$ and $\{C_{T000}, ..., C_{T209}\}$ as points in $\{0.0, \ldots, 1.0\}^2$. Sorting these two-dimensional vectors in ascending order according to their Euclidean norm resulted in two ranking lists: one for temporal and one for non-temporal configuration candidates. The first entries in these lists represented optimal merging ratios. These were ...

$$\begin{gathered}\text{non-temporal: } (0.00, 0.45, 0.55) \text{ with median: } 9.552966 \text{ and stddev: } 0.964822\\ \text{and}\\ \text{temporal: } (0.75, 0.25, 0.00) \text{ with median: } 8.821289 \text{ and stddev: } 0.998641\end{gathered}$$

Note, that in the latter ratio the temporal parser has made the visual parser obsolete (i. e., $V = 0.00$). We assume the reason to be visual over-interpretation of our visual parser. Such over-interpretation can lead to redundancies regarding correct results but also decreased accuracy due to unintended visual relations. This connection between temporal and visual parser, however, is target of future work; so we shall not go into any more detail.

Let us denote the two “winner”-configurations as ...

$$\begin{gathered}C_{NT} = \big((0.00, 0.45, 0.55), D_{NT}\big) \in \{C_{NT00}, ..., C_{NT20}\}\\ \text{and}\\ C_T = \big((0.75, 0.25, 0.00), D_T\big) \in \{C_{T000}, ..., C_{T209}\}\end{gathered}$$

For both divergence samples D_{NT} and D_T we can assume the following:

Assumption 1: D_{NT} and D_T are at least interval-scaled. Divergences between interpretation graphs (see Alg. 31, 32) are measured in computer bits on a continuous scale. Thus, they are metric data. Note, that our divergence builds on Eq. 4.209 which has some of the properties of a metric on the space of probability distributions: $KL_{symmetric}(P, Q)$ is non-negative, it becomes zero if and only if the two distributions P, Q are equal and it is symmetric; that is, $KL_{symmetric}(P, Q) = KL_{symmetric}(Q, P)$. However, it does not satisfy the triangle inequality. Hence it is rather a semi-metric than a distance-metric on the space of probability distributions. Such divergences are typically (though not exclusively) measured in computer bits.

Assumption 2: D_{NT} and D_T are dependent. For each of our 50 test persons we collected (or rather we measured) samples under two experimental conditions: $(0.00, 0.45, 0.55)$ and $(0.75, 0.25, 0.00)$. Thus, one could argue that each participant contributed a pair of related scores. Therefore D_{NT} and D_T can be regarded as being dependent.

Assumption 3: homogeneity of variance. Having applied Levene’s test (for Homogeneity of Variance) on D_{NT} and D_T resulted in a p-value $= 0.7218$, which is non-significant for a significance level $\alpha = 0.05$. From this we can conclude that the variances in both experimental conditions are roughly equal.

Assumption 4: The distribution of the differences between D_{NT} and D_T is approximately normally distributed. The Shapiro-Wilk test was used to prove this assumption. Considering a significance level of $\alpha = 0.05$, the test was non-significant for both D_{NT} (with a p-value = 0.0804695) and D_T (where the p-value was 0.8747074). This tells us that the distributions of both samples D_{NT} and D_T are not significantly different from a normal distribution. Thus they are probably normally distributed and so are their differences.

Provided that D_{NT} and D_T are normally distributed, dependent, metric samples with equal variances, we can analyse them with the paired Student's t-test. This can be performed either one- or two-tailed.

Before collecting our reference data we could not state with certainty that the direction of a (possible) difference in parser performance would only go one way. We could not be sure that the spatio-temporal parser would always perform *at least* as good as the spatio-visual parser. For this reason, and because we were looking for a more conservative test of significance, we decided to perform a two-tailed rather than a one-sided test. Thus, our method of choice was:

the *two-tailed paired Student's t-test.*

With the two-tailed paired (Student's) t-test one can evaluate whether the mean difference between paired observations is significantly different from zero. That is, you can determine whether there is a significant difference between the arithmetic means of two variables, such as in our case D_{NT} and D_T.

5.3.2 Result

Our null hypothesis H_0 stated that there was no effective difference between the non-temporal sample mean and the temporal sample mean; that is, any measured difference in divergence was only due to chance and thus including the temporal parser had no effect:

$$H_0 : \mu_{NT} = \mu_T \quad \text{or} \quad H_0 : \mu_{NT} - \mu_T = 0$$

Here, μ_{NT} stands for the mean of D_{NT} and μ_T represents the mean of D_T.

The opposite of our null hypothesis H_0 was our experimental (or alternative) hypothesis H_1, which assumed that the means of D_{NT} and D_T were not equal; that is, that including the temporal parser changed the mean.

$$H_1 : \mu_{NT} \neq \mu_T \quad \text{or} \quad H_1 : \mu_{NT} - \mu_T \neq 0$$

As a level of significance we selected the widely adopted $\alpha = 0.05$.

Applying the two-tailed paired Student's t-test on D_{NT} as first and D_T as second condition resulted in:

$$t = 3.0635,\ df = 49,\ \text{p-value} = 0.00355$$

with a 95 percent confidence interval ranging
from 0.1325554 to 0.6380730

From $p < \alpha$ or rather from $0.00355 < 0.05$ we can conclude, that there is a significant difference between the means of D_{NT} and D_T (i. e., we reject H_0).

This gets also reflected by the confidence interval. Both, upper bound 0.6380730 and lower bound 0.1325554 have the same sign, which means that the null finding of zero difference lies outside of the confidence interval. From this we can conclude that the difference is statistically significant.

The fact that the t-value ($t = 3.0635$) is positive tells us that D_{NT} had a bigger mean than D_T and so divergences were lower when using the temporal parser.

Even though the result is statistically significant, it tells us nothing of whether the effect is substantive; that is, if it is important in practical terms.

For this reason we converted our t-statistics into a standard effect size. Some widely used effect size for the paired-samples t-test is Cohen's d [41] or rather d_z [42], which is calculated from the difference scores from matched pairs. In order to avoid overestimation of effect sizes we used a more conservative adaption of d_z which is known as d_{rm} [42, 43]. For our t-statistics we got:

$$|d_{rm}| = 0.39$$

Preferably, effect sizes should be interpreted by comparing them to related effects in literature. Since, however, in our special case there are no such references we decided to use the benchmarks defined in [41] instead. In [41] effect sizes are classified as *small* ($|d| = 0.2$), *medium* ($|d| = 0.5$), and *large* ($|d| = 0.8$). Thus, a $|d|$-value between 0.0 to 0.3 could be interpreted as *small*, if it is between 0.3 and 0.6 it would be *moderate*, and an effect size bigger than 0.6 could be regarded as a *large* effect size. According to [44] *medium* $|d|$ represents an effect visible to the naked eye, *small* $|d|$ is noticeably smaller than *medium* but not trivial, and a *large* $|d|$ has the same distance above *medium* as *small* has below.

Our (conservative) effect size $|d_{rm}| = 0.39$ lies between 0.2 and 0.5 and therefore between the thresholds for *small* and *medium* effects (with a slight tendency towards *moderate*). Therefore, as well as being statistically significant, our detected effect is non-trivial and recognizable by humans.

Finally, we can summarise our findings as follows:

On average, divergence from user intention when adding our temporal parser ($M = 8.9335$, $SE = 0.1412$) was significantly lower than when using only spatial and visual parser ($M = 9.318871$, $SE = 0.136446436$); $t(49) = 3.0635$ and $p < 0.05$. The 95 percent confidence interval for the mean difference between the two conditions was 0.1325 to 0.6380. The effect size estimate $|d_{rm}| = 0.39$ indicates a non-trivial effect that is recognizable by humans.

From this we conclude, that the addition of a temporal parser (as defined in Sect. 4.4.5) shifts machine detected structures (encoded in interpretations ; Sect. 4.2.3) significantly closer to what target users (knowledge workers with technical background) intend to express (see our initial hypothesis on page 179).

Thus, spatial parser performance (i. e., accuracy) can be significantly increased when taking into account not only spatial and visual but also temporal object relations.

Chapter 6

Summary, Conclusion and Future Work

6.1 Summary and Conclusion

We started our discussion of spatio-temporal parsing with a general introduction into the field of spatial hypermedia and provided an overview of relevant literature (Chapter 1). It was shown how spatial hypertext and spatial parsing are informally defined (Sect. 1.2 and Sect. 1.3), which default-categories of spatial and visual structures have been identified in literature (Sect. 1.4) and what practical implementations of spatial parsers were developed (Sect. 1.5).

Chapter 2 provided a formal view on spatial hypertext. According to Sect. 2.1, spatial hypertext languages are sets of spatial hypertext artifacts, which are flat collections of spatial hypertext symbols. This forms our "syntactic view". Semantics are defined by interpretations (Sect. 2.2). Interpretations are encodings of how spatial hypertext can be understood and thus form the semantic complement to spatial hypertext artifacts. Spatial parsers are the linking element between spatial hypertext artifacts and interpretations.

Conventional (non-adaptive) parsers are conceptually limited by their underlying source of information (i. e., the spatial hypertext). Due to this limitation there are several types of structures that cannot be recognized properly. In Chapter 3 we identified three categories: (1) ambiguous structures (Sect. 3.1); (2) destroyed structures (Sect. 3.2) and (3) temporal structures (Sect. 3.3). Not recognizing these types of structures limits both quality of parser output and parser performance.

In order to overcome this issue we suggested in Sect. 3.4 to consider not only spatial and visual properties, but also temporal aspects in spatial parser designs. For the detection of destroyed structures we suggested in Sect. 4.4.1 to equip parsers with short-term memory (see "fading" on pages 93–94). Detection of ambiguous and temporal structures required a temporal parser (Sect. 4.4.5). We claimed that considering not only spatial and visual properties but also temporal aspects in spatial parser design can lead to significant increase in parsing accuracy, detection of richer structures and thus higher parser performance.

Both, temporal parser as well as fading-feature were implemented in a prototypical spatial hypermedia system, following a theoretical system model which is described in Chapter 4. According to that model, spatial hypermedia systems are compositions of editing systems supporting in the creation of spatial hypertext (Sect. 4.1) and interpretation systems performing structural analysis (Sect. 4.2). Editing systems are mainly determined by workspace models as described in Sect. 4.1.3, whereas interpretation systems are primarily defined by parsing algorithms (Sect. 4.4). We designed and implemented algorithms for the detection of spatial (Sect. 4.4.2), visual (Sect. 4.4.3), content-related (Sect. 4.4.4) and temporal (Sect. 4.4.5) object relations. A full blockdiagram of our prototypical spatial hypermedia system can be found in the appendix on page 216. A screenshot of the user interface is given on page 217.

Under laboratory conditions only synergies between spatial, visual and temporal parser have been examined. Content parser and fading-functionality were not included in our analysis (for details see page 178).

In order to compare spatio-temporal with spatial or visual parser performance reference data were collected by surveys in a laboratory (Sect. 5.1.2). In our studies 50 participants were asked to solve a simple term-structuring task using an adapted version of our prototypical spatial hypermedia system (Sect. 5.1.1). User activities during a session were recorded in log-files. Having finished their assignment subjects were asked to indicate explicitly what they intended to express. That qualitative feedback was then quantified in order to make it comparable by a machine. With this data at hand we could finally determine to what extent our spatio-temporal parser performed better than pure spatial or visual parser (Sect. 5.2). To this end, the logged edit process (i. e., what the participant did) was passed through temporal, spatial and visual parser and results were merged together in varying ratios. Comparing these mixtures with the given reference data (i. e., what the participant intended to express) resulted in numerical divergence values. These values indicated how close merged parse results came to user intention and therefore could be used for evaluation purposes. Fig. 6.1 summarizes that again.

It turned out, that in none of the test cases pure spatial or visual parser could outperform the spatio-temporal parser. Instead the spatio-temporal parser

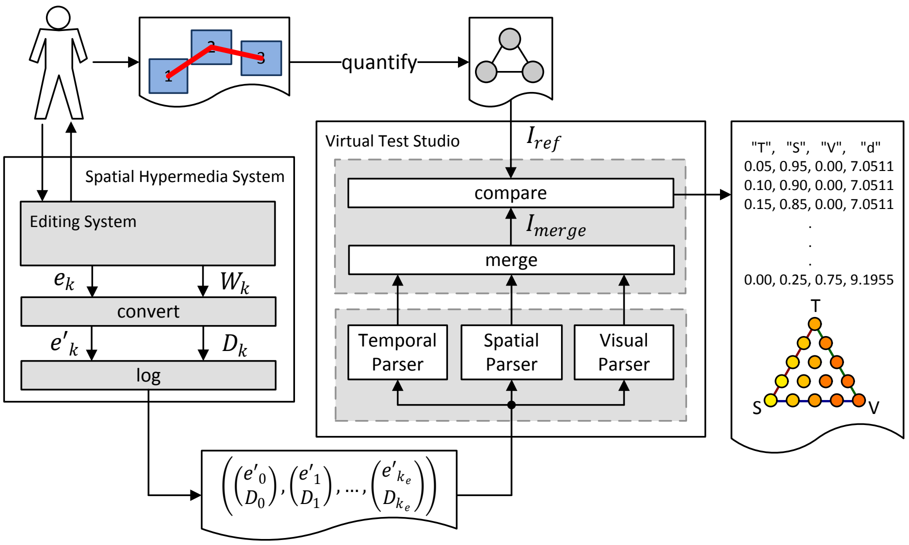


Figure 6.1: overview of survey-process (Sect. 5.1.2) + virtual test run (Sect. 5.2)

rather compensated limitations of conventional parsers and hence provided better results. In order to show that this was not only due to chance, differences between spatio-visual and spatio-temporal parsing accuracy were tested for statistical significance. To this end the two-tailed paired Student's t-test was applied on selected merging ratios, with and without temporal component (Sect. 5.3.1). The result was statistically significant (Sect. 5.3.2). Converted into a standard effect size our statistics indicated a non-trivial effect that is recognizable by humans.

Thus, for our target group we have shown that adaption of parsers is not absolutely necessary to increase quality of parser output. Instead, limited accuracy caused by inadequate parser configurations or heuristics can be partly compensated by adding a temporal parser. This finding is especially helpful in cases when spatial or visual parser would fail to recognize accurate structure, but tuning of parsers to individual user profiles or context of application is undesirable or even impossible.

A good application example includes interfaces for navigating through large and (possibly) unknown information spaces, such as the *World Wide Web*. A widespread practice of navigating the web is text-based search. That is, web-surfers use the output of conventional search engines as shortcuts to documents, possibly continuing their search by further traversing links. Relevant documents are then read and notes are taken on paper or via appropriate applications. Thus, searching and taking notes becomes an iterative process of the user juggling two distinct media or applications. Searching the web for information and taking notes are nowadays integral components of our daily office work. Nevertheless, there is nearly no machine support for integrating both tasks, though there are interfaces for explicit text-based search (e. g., web-interfaces of search engines) and tools that support in note-taking (such as text editors, digital post-its etc.). Consequently, the user is forced to frequently translate between formal search terms and informal note taking without computer support. This produces cognitive overhead that could be avoided by using appropriate tools: Spatial hypermedia systems including specialized parsers.

As mentioned earlier in Chapter 4 (on page 47) and in Sect. 5.1.1, spatial hypermedia systems realise visual structure creation loops where human and machine "interact" to gradually develop meaningful visual structure. Provided that websurfers take their notes in a visual information space (e. g., as post-its in a web browser) we could apply this principle to the aforementioned navigation task. The only difference to our generic system model would lie in the presentation of system feedback; that is, how returned parse results are processed on application level. In our prototypical spatial hypermedia system, interpretations were displayed as graphical overlays on the workspace (see our considerations on page 91). For demonstration purposes this is completely sufficient. However, for more advanced applications, such as the described information navigation service, internal post-processing of parse results is required.

For an intelligent browser interface this might work as follows: structures detected by the system are not simply displayed as they are (i. e., as connecting lines between objects). Instead they are used for auto-generating search queries. That is, interpretation graphs are treated as networks of search terms. These pattern-based queries are executed in the background (e. g., by some semantic data basis). Search results are then added as new elements to the information space (e. g., as geometrical objects that are labeled with key terms). The user could then select relevant objects and re-arrange, scale, color etc. them to implicitly express another (refined) "query-structure". Obsolete elements are deleted. Once the user has decided on which objects to incorporate into the spatial hypertext and which objects to ignore another interpretation–search–evaluation–cycle starts. Fig. 6.2 illustrates that conceptually.

This realises natural and intuitively to handle information navigation loops where searching and note-taking are no separate tasks anymore. Users rather gradually discover an information space by "playing around" with text snippets on a 2d-screen. This decreases cognitive load for knowledge workers and hence improves productivity. In a long run this could increase the average quality of collected knowledge while still reducing manpower costs for online research.

This apparently requires efficient spatial parsers that do not need to be tuned to users or context of application. Otherwise there is no chance to reach out to a broad audience. Our findings from this thesis suggest that this is feasible.

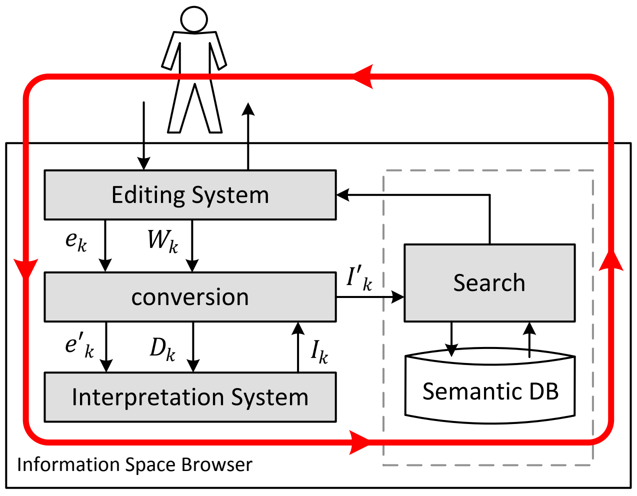


Figure 6.2: Enhanced spatial hypermedia system realising information navigation loops

Another requirement relates to the computer devices such an information navigation service should operate on. Since gesture-based interaction with computers has become the de-facto standard way of working with portable devices (e. g., smart phones, tablets etc.) users increasingly tend to organize information in an intuitive rather than a standardized way. This, however, is more or less ignored by "apps" available for such platforms. Gestures are detected, translated into system commands and forwarded to application-level without undergoing an advanced (structural) analysis. Due to this, information regarding user intention is inevitably getting lost. Thus spatial parsers might fill a gap between multi-touch interfaces and "app-layer". Limitied 2d-space and sloppy interface handling by users, however, might limit a conventional spatial parser's performance. Again, a spatio-temporal parser could solve that problem.

6.2 Future Work

There are enough reasons to further investigate in parsers, as described in this thesis. The following considerations are intended to provide information about our subsequent plan of action and will give inspiration for further research.

Long-Term Studies So far we have analyzed synergy effects between pure spatial, visual and temporal parser only in short-term tests and under controlled conditions. Content parser and fading-functionality were not included in our analysis. As a next step we are planning to integrate all four parsers (with activated fading) into a domain-specific application for document triage; that is, for computer-supported examination of documents. This is where we expect the content parser to show its real added value. Such a productive system will finally make it possible to perform long-term studies outside of the laboratory. This will provide us information about long-term effects of content parser, temporal parser and (particularly) fading under real working conditions and thus will reveal our algorithm's potential of supporting document management tools. If results are positive, we are planning further test runs with probands of different cultural backgrounds. This will give us information about our heuristic's and hence our algorithm's validity across cultural boundaries.

Redundancies Even though spatial, visual and temporal parser analyse different attributes with different heuristics, they do not necessarily generate results that are disjunct. Our findings from Sect. 5.3.1, for instance, let us assume that there are partial overlaps and hence redundancies between the output of visual and temporal parser. We suspect that, to a certain extent, this also applies to the relation between spatial and temporal or spatial and visual parser. As an example, when users create spatially separated groups of visually uniform objects to express categorical relationships, they inevitably introduce redundancies: on the one hand by putting members of the same category close together and on the other hand by providing them with clear visual identity. The first can be exploited by spatial and the latter by visual parsers. When such groupings also change as a whole (for instance because the user moves or scales them en bloc) then category membership can not only be inferred from spatial proximity and visual similarity but also from temporal dependencies. Thus, there are clear redundancies between the output of spatial, visual and temporal parser. The effects of theses overlappings on parser performance, however, still need to be investigated. Furthermore, studies should be conducted (1) whether there are significant differences in pairwise intersections of spatial, visual and temporal parse results and (2) what the determining (heuristic) parameters are.

Granularity Practical experience with our prototypical spatial hypermedia system suggests, that spatial, visual and temporal interpretations vary in their granularity. That is, we assume that there are systematic differences in structural nuances. Spatial parsing, as we defined and implemented it, can be imagined as the process of "information sculpting", where hidden information structure is made "visible" by "carving" (i. e., removing unwanted associations) and "modelling" (i. e., adding meaningful object connections). Depending on how much emphasis you put on different structural aspects (such as spatial, visual, temporal etc.) complete interpretation-graphs (i. e., the "rough material") is sculptured into structures with varying granularity. Our spatial parser, for instance, delivers rather coarsely structured output, whereas visual and (particularly) temporal parser provide comparatively finely structured results. Apart from our initial thoughts concerning primary vs. secondary structures in Sect. 1.4 (on page 10) this has not been considered yet in our work. Thus, assuming that differences in structural granularity are significant, it is still unknown how great these differences are.

Performance As we know already from Chapter 3, good spatial parsers do not only generate interpretations of high accuracy and hence produce results of high quality, but they also perform as resource-saving and as fast as possible. Thus, there are two aspects which play an important role in spatial parser design: effectiveness and efficiency. In this thesis the focus was put solely on parsing accuracy and therefore on effectiveness. Aspects, such as processing speed and resource consumption were not considered. Even though our prototypical implementation has been proven to be functional under test conditions (i. e., in all test cases parsers terminated in reasonable time), their general performance has not been analysed. Although not required for our proof of hypothesis, a theoretical analysis of parser performance would be highly beneficial though. Knowing the differences in efficiency between parsers or heuristics allows to identify performance bottlenecks. This, in turn, will reveal optimization potentials that were not considered in our original parser designs.

Extension We designed and implemented algorithms for the detection of spatial (Sect. 4.4.2), visual (Sect. 4.4.3), content-related (Sect. 4.4.4) and temporal (Sect. 4.4.5) object relations. Thus, our parser design covers the recognition of primary spatial structures and secondary visual nuances (as discussed in Sect. 1.4 on page 10), combined with topic-related similarities and temporal dependencies. Although these mixtures cover already a great range of general structures, there is still room for improvement. We suspect that spatial, visual, content-related and temporal properties of spatial hypertext are not the only determining factors for intended information structure. Instead we assume that there are additional indicators for user intention, which may not be as apparent as proximity, similarity etc., but still contribute significantly to a spatial hypertext's structural meaning. We expect that exploiting these (currently unknown) properties via additional heuristics will not only result in richer structures but also further support disambiguation and therefore enhance structure detection. This, however, requires that potential inconsistencies between individual parse results are compensable. Note, that adding further heuristics automatically increases the risk of logical inconsistencies. What is still unknown is whether there are mutually contradictory heuristics where compensation of inconsistencies is not possible. Therefore, both needs to be investigated, (1) properties of spatial hypertexts that are still unexploited by parsers and (2) the effects of combining them with spatial, visual, content and temporal object relations.

Bibliography

[1] C. C. Marshall and F. M. Shipman, III, "Spatial hypertext: designing for change," *Commun. ACM*, vol. 38, no. 8, pp. 88–97, Aug. 1995. [Online]. Available: http://doi.acm.org/10.1145/208344.208350

[2] F. M. Shipman, III and C. C. Marshall, "Formality considered harmful: Experiences, emerging themes, and directions on the use of formal representations in interactive systems," *Comput. Supported Coop. Work*, vol. 8, no. 4, pp. 333–352, Oct. 1999. [Online]. Available: http://dx.doi.org/10.1023/A:1008716330212

[3] C. C. Marshall and F. M. Shipman, III, "Searching for the missing link: discovering implicit structure in spatial hypertext," in *Proceedings of the fifth ACM conference on Hypertext*, ser. HYPERTEXT '93. New York, NY, USA: ACM, 1993, pp. 217–230. [Online]. Available: http://doi.acm.org/10.1145/168750.168826

[4] F. Shipman and C. Marshall, "Formality considered harmful: Experiences, emerging themes, and directions." Department of Computer Science, Univ. of Colorado, Boulder, CO, Technical Report CU-CS-648-93, 1993.

[5] F. M. Shipman, III and R. McCall, "Supporting knowledge-base evolution with incremental formalization," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '94. New York, NY, USA: ACM, 1994, pp. 285–291. [Online]. Available: http://doi.acm.org/10.1145/191666.191768

[6] C. Atzenbeck and D. L. Hicks, "Integrating time into spatially represented knowledge structures," in *Proceedings of the 2009 International Conference on Information, Process, and Knowledge Management*. IEEE Computer Society, 2009, pp. 34–42. [Online]. Available: http://dl.acm.org/citation.cfm?id=1510519.1510546

[7] F. M. Shipman, III, H. Hsieh, P. Maloor, and J. M. Moore, “The visual knowledge builder: a second generation spatial hypertext,” in *Proceedings of the 12th ACM conference on Hypertext and Hypermedia*, ser. HYPERTEXT ’01. New York, NY, USA: ACM, 2001, pp. 113–122. [Online]. Available: http://doi.acm.org/10.1145/504216.504245

[8] F. Shipman, R. Airhart, H. Hsieh, P. Maloor, J. M. Moore, and D. Shah, “Visual and spatial communication and task organization using the visual knowledge builder,” in *Proceedings of the 2001 International ACM SIGGROUP Conference on Supporting Group Work*, ser. GROUP ’01. New York, NY, USA: ACM, 2001, pp. 260–269. [Online]. Available: http://doi.acm.org/10.1145/500286.500325

[9] F. M. Shipman, III and C. C. Marshall, “Spatial hypertext: an alternative to navigational and semantic links,” *ACM Comput. Surv.*, vol. 31, no. 4es, Dec. 1999. [Online]. Available: http://doi.acm.org/10.1145/345966.346001

[10] D. Bucka-Lassen, C. Pedersen, and O. Reinert, *Cooperative Authoring Using Open Spatial Hypermedia.* Aarhus University, Department of Computer Science, 1998. [Online]. Available: http://books.google.de/books?id=CZayZwEACAAJ

[11] L. Francisco-Revilla and F. Shipman, “Parsing and interpreting ambiguous structures in spatial hypermedia,” in *Proceedings of the sixteenth ACM conference on Hypertext and hypermedia*, ser. HYPERTEXT ’05. New York, NY, USA: ACM, 2005, pp. 107–116. [Online]. Available: http://doi.acm.org/10.1145/1083356.1083376

[12] C. C. Marshall, F. M. Shipman, III, and J. H. Coombs, “VIKI: spatial hypertext supporting emergent structure,” in *Proceedings of the 1994 ACM European conference on Hypermedia technology*, ser. ECHT ’94. New York, NY, USA: ACM, 1994, pp. 13–23. [Online]. Available: http://doi.acm.org/10.1145/192757.192759

[13] H. Hsieh and F. Shipman, “Activity links: supporting communication and reflection about action,” in *Proceedings of the sixteenth ACM conference on Hypertext and hypermedia*, ser. HYPERTEXT ’05. New York, NY, USA: ACM, 2005, pp. 161–170. [Online]. Available: http://doi.acm.org/10.1145/1083356.1083388

[14] F. M. Shipman, III, C. C. Marshall, and T. P. Moran, “Finding and using implicit structure in human-organized spatial layouts of information,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI ’95. New York, NY, USA:

ACM Press/Addison-Wesley Publishing Co., 1995, pp. 346–353. [Online]. Available: http://dx.doi.org/10.1145/223904.223949

[15] E. J. Golin, "Parsing visual languages with picture layout grammars," *J. Vis. Lang. Comput.*, vol. 2, pp. 371–393, December 1991. [Online]. Available: http://dx.doi.org/10.1016/S1045-926X(05)80005-9

[16] *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations.* World Scientific, 1997.

[17] T. Igarashi, S. Matsuoka, and T. Masui, "Adaptive recognition of implicit structures in human-organized layouts," in *Visual Languages, Proceedings., 11th IEEE International Symposium on*, Sep 1995, pp. 258–266.

[18] C. C. Marshall and R. A. Rogers, "Two years before the mist: experiences with Aquanet," in *Proceedings of the ACM conference on Hypertext*, ser. ECHT '92. New York, NY, USA: ACM, 1992, pp. 53–62. [Online]. Available: http://doi.acm.org/10.1145/168466.168490

[19] F. M. Shipman, III, "Supporting knowledge-base evolution with incremental formalization," Ph.D. dissertation, Boulder, CO, USA, 1993, uMI Order No. GAX94-06815.

[20] C. C. Marshall and F. M. Shipman, III, "Spatial hypertext and the practice of information triage," in *Proceedings of the eighth ACM conference on Hypertext*, ser. HYPERTEXT '97. New York, NY, USA: ACM, 1997, pp. 124–133. [Online]. Available: http://doi.acm.org/10.1145/267437.267451

[21] C. C. Marshall, F. G. Halasz, R. A. Rogers, and W. C. Janssen, Jr., "Aquanet: a hypertext tool to hold your knowledge in place," in *Proceedings of the third annual ACM conference on Hypertext*, ser. HYPERTEXT '91. New York, NY, USA: ACM, 1991, pp. 261–275. [Online]. Available: http://doi.acm.org/10.1145/122974.123000

[22] F. G. Halasz, T. P. Moran, and R. H. Trigg, "Notecards in a nutshell," in *Proceedings of the SIGCHI/GI Conference on Human Factors in Computing Systems and Graphics Interface*, ser. CHI '87. New York, NY, USA: ACM, 1987, pp. 45–52. [Online]. Available: http://doi.acm.org/10.1145/29933.30859

[23] F. M. Shipman, R. J. Chaney, and G. A. Gorry, "Distributed hypertext for collaborative research: The virtual notebook system," in *Proceedings of the Second Annual ACM Conference on Hypertext*, ser. HYPERTEXT '89. New York, NY, USA: ACM, 1989, pp. 129–135. [Online]. Available: http://doi.acm.org/10.1145/74224.74235

[24] T. W. Malone, “How do people organize their desks?: Implications for the design of office information systems,” *ACM Trans. Inf. Syst.*, vol. 1, no. 1, pp. 99–112, Jan. 1983. [Online]. Available: http://doi.acm.org/10.1145/357423.357430

[25] R. Mander, G. Salomon, and Y. Y. Wong, “A pile metaphor for supporting casual organization of information,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI ’92. New York, NY, USA: ACM, 1992, pp. 627–634. [Online]. Available: http://doi.acm.org.zorac.aub.aau.dk/10.1145/142750.143055

[26] A. Kidd, “The marks are on the knowledge worker,” in *Conference Companion on Human Factors in Computing Systems*, ser. CHI ’94. New York, NY, USA: ACM, 1994, pp. 212–. [Online]. Available: http://doi.acm.org/10.1145/259963.260346

[27] O. Reinert, D. Bucka-Lassen, C. A. Pedersen, and P. J. Nürnberg, “CAOS: a collaborative and open spatial structure service component with incremental spatial parsing,” in *Proceedings of the tenth ACM Conference on Hypertext and hypermedia : returning to our diverse roots*, ser. HYPERTEXT ’99. New York, NY, USA: ACM, 1999, pp. 49–50. [Online]. Available: http://doi.acm.org/10.1145/294469.294484

[28] M. B. Nielsen and P. Ørbæk, “Spatial parsing within the Topos 3d environment,” in *IN THE FIRST WORKSHOP ON SPATIAL HYPERTEXT*, 2001, p. 1.

[29] ——, “Finding hyper-structure in space: Spatial parsing in 3d,” in *THE NEW REVIEW OF HYPERMEDIA AND MULTIMEDIA*. Taylor & Francis, Inc., 2001, pp. 153–183.

[30] K. Nakakoji, Y. Yamamoto, S. Takada, and B. N. Reeves, “Two-dimensional spatial positioning as a means for reflection in design,” in *Proceedings of the 3rd conference on Designing interactive systems: processes, practices, methods, and techniques*, ser. DIS ’00. New York, NY, USA: ACM, 2000, pp. 145–154. [Online]. Available: http://doi.acm.org/10.1145/347642.347697

[31] Y. Yamamoto, K. Nakakoji, and A. Aoki, “Spatial hypertext for linear-information authoring: Interaction design and system development based on the ART design principle,” in *Proceedings of the thirteenth ACM conference on Hypertext and hypermedia*, ser. HYPERTEXT ’02. New York, NY, USA: ACM, 2002, pp. 35–44. [Online]. Available: http://doi.acm.org/10.1145/513338.513351

[32] Y. Yamamoto, K. Nakakoji, Y. Nishinaka, M. Asada, and R. Matsuda, "What is the space for?: the role of space in authoring hypertext representations," in *Proceedings of the sixteenth ACM conference on Hypertext and hypermedia*, ser. HYPERTEXT '05. New York, NY, USA: ACM, 2005, pp. 117–125. [Online]. Available: http://doi.acm.org/10.1145/1083356.1083378

[33] M. Bernstein, "Design note: Neighborhoods in spatial hypertext," *SIGWEB Newsl.*, vol. 6, no. 1, pp. 15–19, Feb. 1997. [Online]. Available: http://doi.acm.org/10.1145/278530.278532

[34] F. Shipman, J. M. Moore, P. Maloor, H. Hsieh, and R. Akkapeddi, "Semantics happen: knowledge building in spatial hypertext," in *Proceedings of the thirteenth ACM conference on Hypertext and hypermedia*, ser. HYPERTEXT '02. New York, NY, USA: ACM, 2002, pp. 25–34. [Online]. Available: http://doi.acm.org/10.1145/513338.513350

[35] D. Kim and F. M. Shipman, "Interpretation and visualization of user history in a spatial hypertext system," in *Proceedings of the 21st ACM conference on Hypertext and hypermedia*, ser. HT '10. New York, NY, USA: ACM, 2010, pp. 255–264. [Online]. Available: http://doi.acm.org/10.1145/1810617.1810663

[36] F. M. S. III and H. wei Hsieh, "Navigable history: a reader's view of writer's time." *The New Review of Hypermedia and Multimedia*, vol. 6, pp. 147–167, 2000. [Online]. Available: http://dblp.uni-trier.de/db/journals/nrhm/nrhm6.html#ShipmanH00

[37] F. M. Shipman, H. Hsieh, J. M. Moore, and A. Zacchi, "Supporting personal collections across digital libraries in spatial hypertext," in *Proceedings of the 4th ACM/IEEE-CS joint conference on Digital libraries*, ser. JCDL '04. New York, NY, USA: ACM, 2004, pp. 358–367. [Online]. Available: http://doi.acm.org/10.1145/996350.996433

[38] J. Schanda, *Colorimetry: Understanding the CIE System.* Wiley, 2007. [Online]. Available: https://books.google.de/books?id=uZadszSGe9MC

[39] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Mar. 2003. [Online]. Available: http://dl.acm.org/citation.cfm?id=944919.944937

[40] S. Kullback and R. A. Leibler, "On information and sufficiency," *Ann. Math. Statist.*, vol. 22, no. 1, pp. 79–86, 1951.

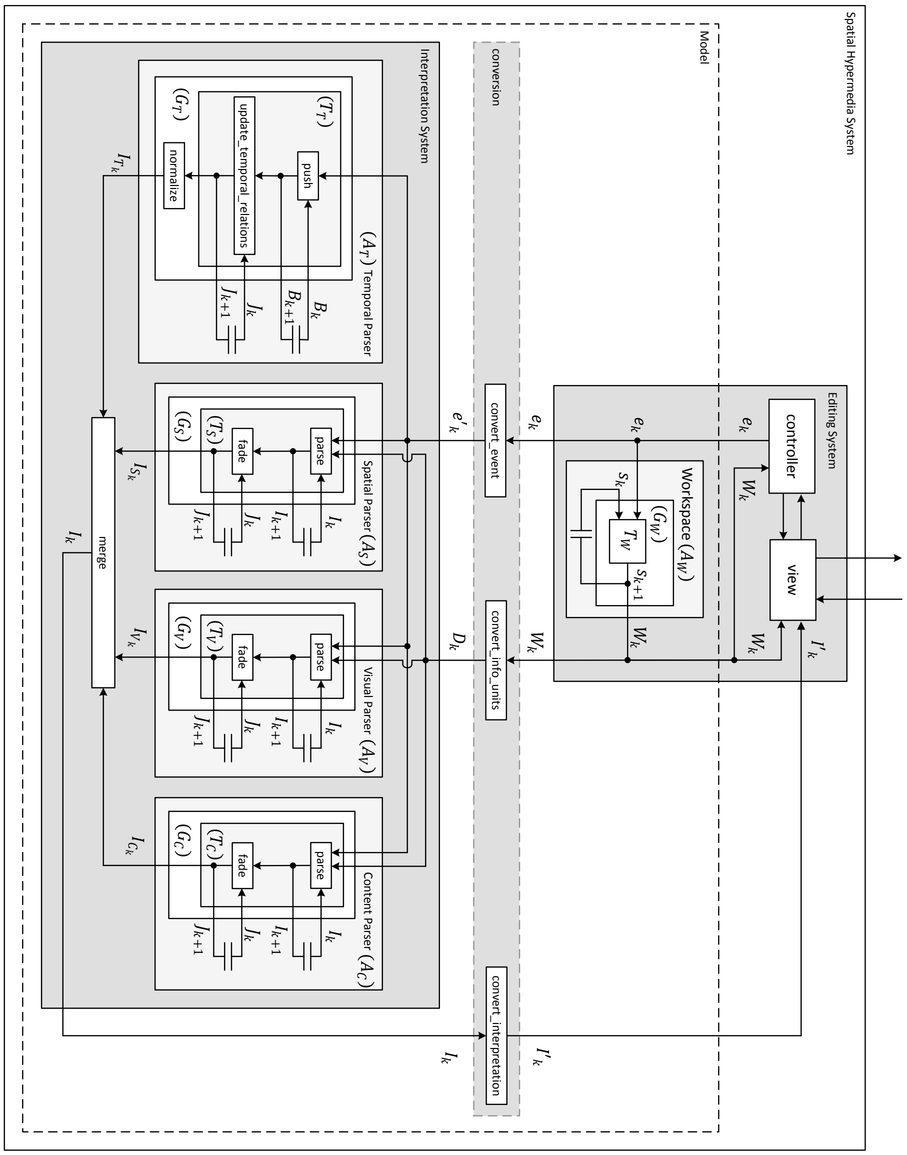
[41] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences.* L. Erlbaum Associates, 1988. [Online]. Available: https://books.google.de/books?id=Tl0N2lRAO9oC

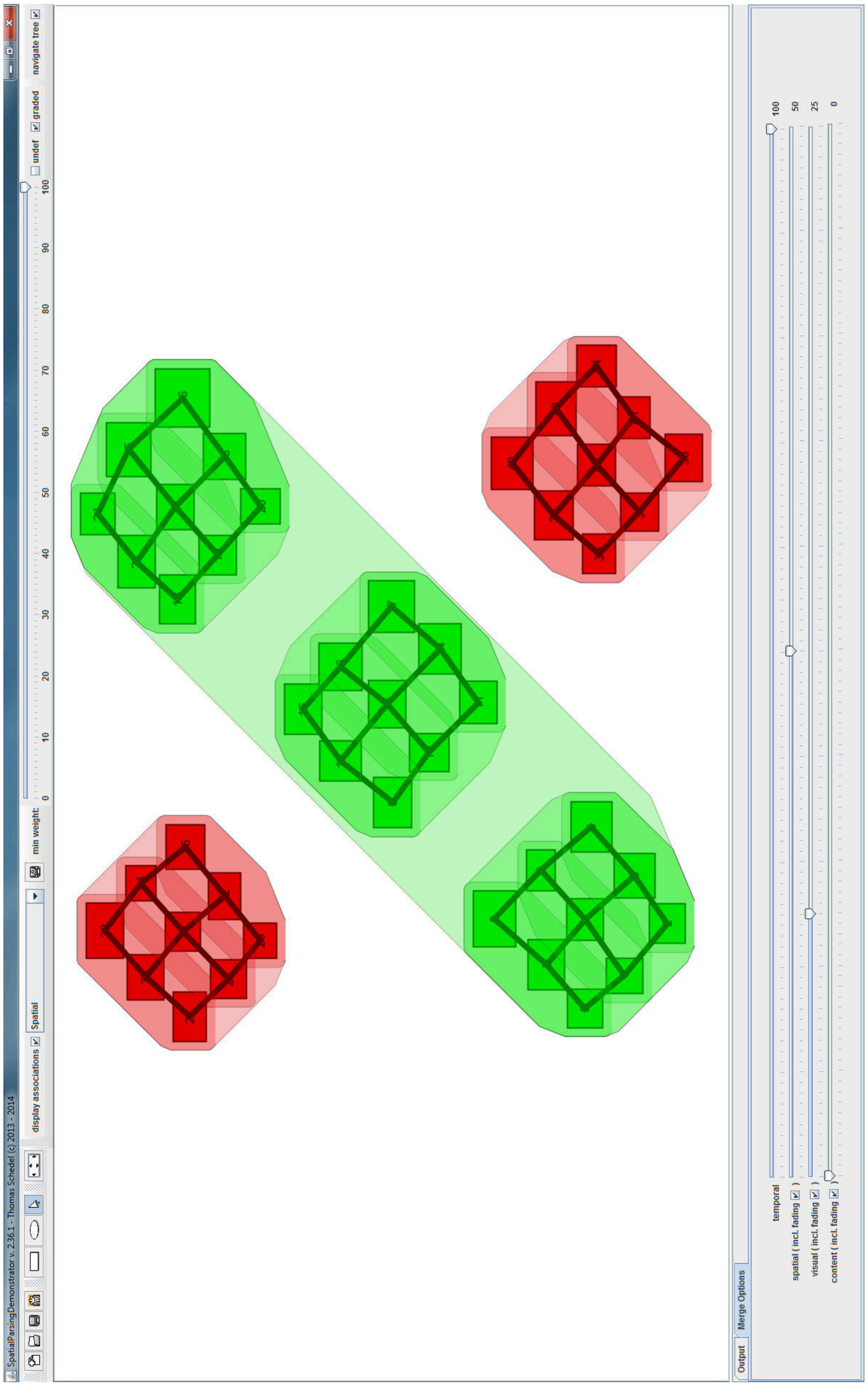
[42] D. Lakens, "Calculating and reporting effect sizes to facilitate cumulative science: A practical primer for t-tests and ANOVAs," *Frontiers in Psychology*, vol. 4, no. 863, 2013. [Online]. Available: http://www.frontiersin.org/cognition/10.3389/fpsyg.2013.00863/abstract

[43] W. P. Dunlap, J. M. Cortina, J. B. Vaslow, and M. J. Burke, "Meta-analysis of experiments with matched groups or repeated measure designs." *Psychological Methods*, vol. 1, no. 2, pp. 170–177, 1996.

[44] J. Cohen, "Quantitative methods in psychology: A power primer," *Psychological Bulletin*, vol. 112, no. 1, pp. 155–159, 1992.

Appendix


Spatial Hypermedia System
Model
Interpretation System
conversion
Editing System
Temporal Parser
(A_T)
(G_T)
(T_T)
push
update_temporal_relations
normalize
B_k
B_{k+1}
J_k
J_{k+1}
I_{T_k}
Spatial Parser (A_S)
(G_S)
(T_S)
parse
fade
I_k
I_{k+1}
J_k
J_{k+1}
I_{S_k}
Visual Parser (A_V)
(G_V)
(T_V)
parse
fade
I_{V_k}
Content Parser (A_C)
(G_C)
(T_C)
parse
fade
I_{C_k}
merge
I_k
convert_event
convert_info_units
convert_interpretation
e'_k
e_k
D_k
W_k
I'_k
controller
view
Workspace (A_W)
(G_W)
T_W
s_k
s_{k+1}


SpatialParsingDemonstrator v. 2.36.1 - Thomas Schedel (c) 2013 - 2014
display associations
Spatial
min weight:
0
10
20
30
40
50
60
70
80
90
100
undef
graded
navigate tree
Output
Merge Options
temporal
spatial (incl. fading)
visual (incl. fading)
content (incl. fading)
100
50
25
0