FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Visualizing and interactive modelling of process hierarchies

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Masterstudium Software Engineering & Internet Computing

eingereicht von

Hubert Hirsch
Matrikelnummer 0625008

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Mag. Dr. Horst Eidenberger
Mitwirkung: Mag. Dr. Katharina Kaiser

Wien, 07.10.2013

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Visualizing and interactive modelling of process hierarchies

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Master of Software Engineering & Internet Computing

by

Hubert Hirsch
Registration Number 0625008

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Mag. Dr. Horst Eidenberger
Assistance: Mag. Dr. Katharina Kaiser

Vienna, 07.10.2013

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Hubert Hirsch

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)	(Unterschrift Verfasser)

Abstract

This study deals with the visualization and modeling of protocol-based treatment plans that are composed from hierarchically structured skeletal-plans, and defined in the Asbru language. For this it introduces PlanStrips, a tree-map based visualization, designed for ease of use and simplicity. To test the PlanStrips visualization a prototype of the same name was developed. PlanStrips main characteristics are a dense presentation of treatment plans and qualitative presentation of plan execution order. The prototype also displays other guideline features such as conditions.

After an introduction and short description of the projects background, the thesis takes a look at well-known visualization approaches for hierarchically structured data, as well as existing tools for Asbru guideline visualization and editing. In a discussion aspects/features of these tools are juxtaposed with PlanStrips.

A user study was conducted to evaluate the PlanStrips prototype and verify its claims of simplicity and easy of use. The evaluation included three physicians, two domain experts, and a software engineer. The participants' knowledge in the fields of medical guidelines, guideline definition languages and general computer knowledge varied from novice to expert level. The results of the evaluation are presented and discussed. Both a qualitative and, to a lesser degree, quantitative evaluation of the findings are given.

Kurzfassung

Diese Arbeit beschaeftigt sich mit der Visualisierung und Modellierung von medizinischen Leitlinien und Protokollen, welche auf hierarchisch verschachtelten Plaenen basieren, und in der Asbru ontologie definiert sind. Zu diesem Zweck wird PlanStrips eingefuehrt, eine *tree-map* basierte Visualisierung, deren Schwerpunkte auf Bedienbarkeit und Einfachheit liegen. Fuer die Arbeit wurde die PlanStrips Visualisierung in einem gleichnamigen Prototyp umgesetzt. Ihre Hauptcharakteristiken sind eine dichte Praesentation von Behandlungsplaenen und die qualitative Praesentation derer Ausfuehrungsreihenfolge. Der Prototyp zeigt zusaetzlich andere Merkmale, wie z.B. mit Plaenen verbundene Konditionen an.

Nach einer Einfuehrung und kurzer Beschreibung des Hintergrunds des Projekts, beschaeftigt sich die Arbeit mit wohlbekannten Visualisierungsmethoden fuer hierarchisch strukturierte Daten, als auch bereits existierenden Programmen zum Veranschaulichen und Bearbeiten von Asbru Richtlinien. In einer Diskussion werden einzelne Aspekte/Features dieser Programme mit PlanStrips gegenuebergestellt und verglichen.

Eine Benutzerstudie wurde durchgefuehrt um den PlanStrips Prototyp, und die Behauptungen bezueglich einfacher und intuitiver Bedienbarkeit zu evaluieren. Die Testgruppe setzte sich zusammen aus drei Aerzten, zwei Domain-experten, und einem Softwareentwickler. Das Vorwissen der Teilnehmer im medizinischen Umfeld, computerbasierten Sprachen zur Definition von Behandlungsprotokollen, als auch generelles Computerwissen reichte von Anfaenger- bis zu Expertenlevel. Die Ergebnisse der Studie werden praesentiert und diskutiert, und sowohl in einer qualitativen als auch, zu einem geringeren Grad, quantitativen Art vorgestellt.

Contents

CHAPTER 1

Introduction

1.1 Motivation

By the *Institute of Medicine*'s definition "Clinical practice guidelines are systematically developed statements to assist practitioner and patient decisions about appropriate health care for specific clinical circumstances" [16].

In general the information contained in a guideline are procedural aspects, descriptions of actions taken during care, declarative aspects with details about the medical concepts, definitions, and patterns of the patient's state [18]. A brief summary on the origin of guidelines and their importance in today's world is given in section 2.1.

Guidelines are mostly text based, and to some degree include tables and charts. Unless a user is already familiar with the structure of a given guideline it is no trivial task to pinpoint specific information or guideline aspect. Formulating a guideline in a standardized computer-readable format helps to solve these problems. It increases guideline usability by allowing for better sharing, (collaborative) editing, version management, tracking of changes, re-use, etc. Using editing and viewing software, specific aspects of a guideline (e.g., declarative or procedural knowledge) can be emphasized by using specialized views which help the users to better work with a guideline by looking at it from different facets. In contrast to the monolithic nature of a free-text guideline, a computerized guideline, if realized correctly, can reduce complexity using the proven *divide & conquer* design paradigm.

However the most important feature of computerized guidelines is the ability to integrate them into modern *Clinical decision support system* (CDSS). The main purpose of which is to assist clinicians at the point of care [11] (e.g. provide treatment recommendations for their current patients).

1.2 Problem description

This work applies to the end-user interface domain of computerized guidelines. The goal is to create a software (named PlanStrips) which enables easy and intuitive editing and viewing of block-based ontologies which describe control flow. Specifically PlanStrips functions with guidelines specified in the Asbru language. However the basic concept could also be applied to other block-based process specification languages.

Asbru defines a hierarchy of plans which are nested in blocks. A plan constitutes the top of the process hierarchy. A plan itself is an action, e.g. "Dietary management in diabetes", and can be composed of nested sub-plans. If a plan contains children it also specifies details about their execution (execution order, conditions, etc.). Section 2.2 describes the history, relevance and characteristics of the Asbru ontology in more detail.

The resulting plan-hierarchies can have a high number of plans and nesting depth. When dealing with a complex hierarchy it becomes difficult to mentally grasp its structure. PlanStrips tackles this problem by visualizing the hierarchy in a well arranged view that is intuitive to understand. Going one step further than pure visualization (understanding) PlanStrips provides editing functionality that aid knowledge engineers in the modelling process. PlanStrips should better enable knowledge engineers to design and model CPGs, and care providers and various stakeholders to understand and work with them.

There are other visualizations beside PlanStrips in the Asbru domain which cater to different (but sometimes overlapping) requirements, these are described in chapter 4.

As established in [42], PlanStrips caters the following specific requirements:

Dense presentation Being based on tree-maps (see section 3.2) PlanStrips can display plans with a high density. Other representations, e.g. graph-based, require rendering of edges between nodes, and have to deal with the non-trivial task of laying out the nodes in a well-arranged manner – which tends to "waste" screen space (see section 3.1).

Intuitively arrange parallel plans and sequences PlanStrips declares the x-axis as sequential time axis, and y-axis as parallel axis. Plans are arranged accordingly, which (if the user knows the convention) allows users to gain information about temporal execution order at a glance.

Qualitative presentation of the temporal dimension PlanStrips does not display temporal detail (i.e., specific or relative points in time, defining when execution of a plan begins/ends, plan-duration, etc.). Here qualitative temporal presentation means displaying execution order of plans relative to each other. This should be sufficient for understanding processes modeled in a guideline. Additionally plans often have temporal uncertainty associated with them, which complicates visualizing temporal detail. Other visualizations and software for dealing with temporal uncertainty already exist (see section 4.3, or [7]).

Easy to explain to a non-IT person Quoting [42]: "Domain experts such as physicians have limited time and little motivation for dealing with IT concepts. For a presentation to be well-received, it is crucial to demonstrate from the start that it is simple. At the same time, we do not see physicians as those modeling CGPs themselves. Therefore, it is not

required that they understand our visualisation without an aide, it is only important that it does not appear overly complex or technical.”

CHAPTER 2

Background

2.1 Clinical Practice Guidelines

General

Medical guidelines are part of an effort to better standardize medical practices in order to improve quality of care. Examples of official guidelines go back at least until World War II. Federal spending in the health care sector in the United States rose from 22% to 40% between 1960 and 1980. This led to the establishment of the federal *Agency for Healthcare Research and Quality* (*AHRQ*) in 1989. Their mandate was to produce *clinical practice guidelines* (*CPGs*). CPGs are now used throughout western medicine and number in the thousands. [53]

Relevance

The introduction of standardized international CPGs provides reference material for health care practitioners to refer to. It may put them in a better position to diagnose, classify, prevent and treat a medical condition. In addition guidelines can be used to better explain treatment to a patient, introducing more transparency in the treatment process.

Guidelines are rooted in evidence based medicine, that is, using current best evidence in making decisions about the care of individual patients [39]. They are written in natural language. CPGs also need to be kept up to date with recent developments and insights gained in the medical field. The accelerating rate at which new technology is developed and incorporated into active use makes guidelines subject to frequent revision.

2.2 Asbru

Asbru is a domain ontology that is expressive enough to formalize guidelines into a computer processable format. It is based around the concept of *skeletal plans* [17], i.e. schemata with variable level of detail, which capture the essence of a procedure while still being flexible [43].

A plan itself is an action such as "Dietary management in diabetes". Asbru structures plans in a hierarchical composition (similar to a *work breakdown structure*). A top-level plan (root) contains subplans (nodes), which in turn can contain other subplans. A plan which can no longer be decomposed (a leaf) is an *action* or *user-performed* plan.

History

Asbru is a domain specific language which was first introduced in 1998 [43]. It emerged as part of the Asgaard project led by the Vienna University of Technology and Stanford Medical Informatics. The goal of the Asgaard project is to provide computer aided solutions for the design and execution of CPGs. Asbru is used to represent clinical practice guidelines in a computer processable format.

Relevance

As mentioned in section 1.1 the most important feature of formalized computer-readable guidelines is the ability to integrate them into modern *Clinical decision support system* (CDSS). The main purpose of these systems is to assist clinicians at the point of care [11], for example by providing treatment recommendations for their current patients.

In addition translating guidelines into a computer processable format enables dynamic visualization, editing, sharing, on-line collaboration, version and change tracking, linking, sharing of knowledge concepts across guidelines, integration with electronic patient records etc. Asbru is not the only such approach, an evaluation of other ontologies can be found in [34]. As of now the Asbru ecosystem (language, libraries, tools, etc.) is still under development and subject of research. There are a few prototypical Asbru translations of guidelines (e.g. Jaundice in newborns, Diabetes) used for experimentation.

Characteristics

The main features of Asbru according to [5]:

1. prescribed actions and states can be continuous
2. intentions, conditions, and world states are temporal patterns
3. uncertainty in both temporal scopes and parameters can be flexibly expressed by bounding intervals
4. plans might be executed in sequence, all or some plans in parallel, all or some plans in a particular order or unordered, or periodically
5. particular conditions are defined to monitor the plan's execution

There are two basic operation modes that Asbru is used in: the *design* and *execution* phase.

Design phase During the design phase the plan can be thought of as a blueprint. The procedural and declarative knowledge contained in a guideline is modeled by expert physicians and knowledge engineers to create a formalized Asbru version.

Execution phase The execution phase is when actual treatment of a patient takes place. Concrete patient data is evaluated against the process modeled in the guideline to direct the treatment. Conditions are evaluated, plans executed, etc.

Information contained in a plan can be intentions, preferences, conditions, child execution order (plan synchronization) and temporal details (starting-time, duration, etc.).

Intentions

Intentions show high-level goals of a plan by describing temporal properties. An intention can be to reach, maintain or avoid a state [8]. They are important for choosing the right plan and critiquing treatment plans during the revision process [27].

Preferences

Preferences express given conditions which should be fulfilled prior to plan execution. They constrain applicability of the plan (e.g. select-criteria: exact-fit, roughly-fit). They also provide information about the behavior/nature in which a plan is executed (e.g. aggressive, normal).

Conditions

Conditions are logic expressions which guide a plan instance through its life cycle (active, suspended, completed and aborted). If multiple instances of the same plan are active simultaneously conditions are evaluated on a per-instance basis. The different types of conditions are:

Filter & Setup These two conditions must hold before execution of a plan can start. The result of *filter-precondition* is static and not expected to change during the course of treatment (e.g. "patient is male"). *Setup-precondition* can be fulfilled (e.g. anaesthesia).

Suspend & Reactivate If the plan has already been started and the *suspend-condition* is met the plan is suspended. In order to be activated again the *reactivate-condition* must hold. When a plan is re-activated, execution continues at the point where it was last suspended.

Complete & Abort For a plan to complete successfully the logic expression in the *complete-condition* must evaluate to true. If the plan contains subplans/actions these must also satisfy their complete conditions. The *abort-condition* is evaluated throughout execution, if it becomes true the plan has failed to reach its intended goals and is aborted (terminal state).

Synchronization

Plan synchronization specifies the execution order of a plan's children. Possible types of synchronization are:

sequential Children are executed one at a time, in the sequential order in which they were defined.

any-order Children are executed one at a time, however nothing is known about the actual execution other.

parallel Execution of all children starts at the same time.

unordered No constraints on execution order are given.

cyclical The single child of a cyclical plan is executed repeatedly based on a loop condition. Cyclical plans are restricted to only have one child.

Time annotations

Time annotations specify temporal aspects of a plan. These are the (a) earliest starting shift, (b) latest starting shift, (c) earliest finishing shift, (d) and latest finishing shift, (e) minimum duration and (f) maximum duration (all of which are optional). Shifts are specified relative to a point in time (specific or tied to another abstract point or event).

CHAPTER 3

State of the art – Visualization Methods

Asbru's block based structuring approach leads to potentially big hierarchies. For orientation, the two Asbru XML reference guideline implementations have the following properties:

1. Jaundice [32] depth: 20, breadth: 25, nodes: 100+
2. Diabetes [38] depth: 23, breadth: 37, nodes: 200+

Where *depth* is the depth of the hierarchy tree, *breadth* is the number of nodes at the widest depth and *nodes* describes the total number of nodes in the tree.

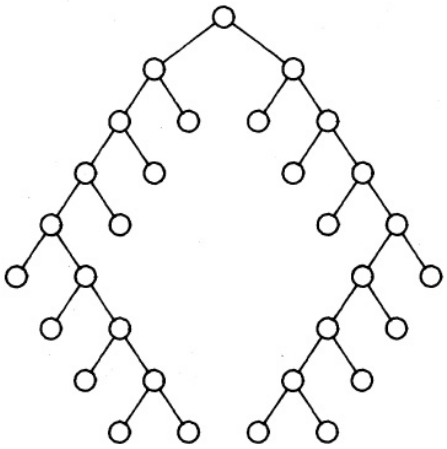
Visualizations which relate to guidelines fall into two task categories [4]. *Design time* visualizations are involved in plan generation, editing, verification, communication of concepts, etc. *Execution time* visualization take into account data which becomes available at execution time, such as patient data, treatment outcomes, etc. The area of interest here is *design time visualization*.

Four basic styles are commonly used to visualize hierarchies (i.e. trees graphs): (1) node link – figure 5.1, (2) enclosure – figure 3.2, (3) adjacency and alignment – figure 3.5a, and (4) indented outline – figure 3.6a [20, 55]. These paradigms can be combined to create novel visualizations. They are described in the following sections, with special focus on enclosures – which are used by PlanStrips.

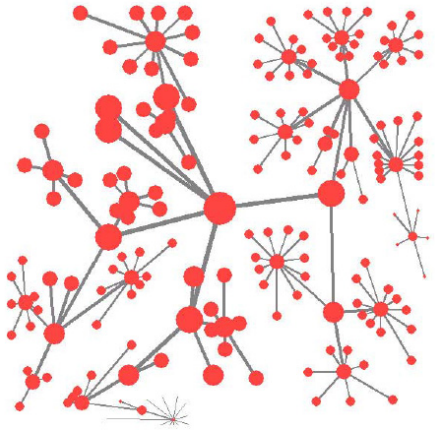
3.1 Node link

Node link diagrams, show the relation between nodes in a graph by drawing an edge (line) between parent and child. As a result they are very good at explicitly showing the relational structure of a graph. Node link diagrams use up much screen space, effectively utilizing only a small portion of the available space. Node link diagrams of trees can be drawn more efficiently while still maintaining aesthetic characteristics [37]. There are even more efficient methods, using radial layouts [30]. Still more effective layout algorithms use both a radial approach and clever positioning of nodes [31] (see figure 3.1b). The more efficient a node link diagram becomes in

terms of space utilization, the more the structure is obscured (figure 5.1). In figure 3.1b not all children have the same distance from their parents, lines start to overlap, etc. This happens even as nodes are displayed through small dots, without displaying any content information.



(a) Aesthetic tree (excerpt from [23])



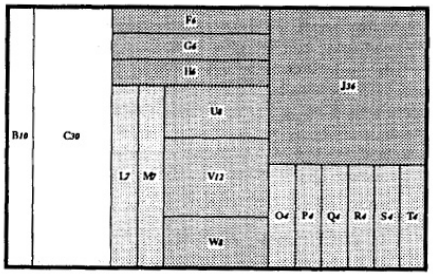
(b) Space optimized tree (excerpt from [31])

Figure 3.1: Node link diagrams of trees

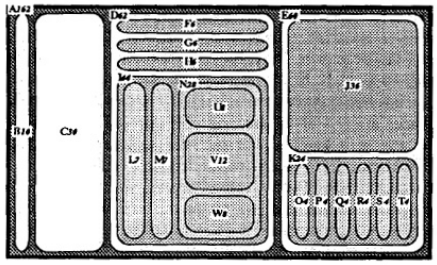
3.2 Enclosure

Tree maps are a representative of the enclosure (and space-filling) approach. They use containment to show structure information, and typically do so implicitly (figure 3.2a).

Classic tree maps use 100% of the available screen space, and display structure and content. The main objectives in their design were (1) Efficient Space Utilization, (2) Comprehension, (3) Interactivity, and (4) Aesthetics [23].



(a) Tree map



(b) Nested tree map

Figure 3.2: Rectangular 2d tree maps (excerpt from [23])

Tree maps can use the following vectors to display information: (1) alignment (e.g. vertical or horizontal), (2) color and texture, (3) node shape and size, (4) border, (5) nesting offset,

(6) animations (e.g. blinking, shaking, or pulsing), (7) labels and pictograms, and (8) multiple views (*small multiples*) [23, 31, 51]. Naturally some of these conflict with each other. Choosing to fixate or emphasize certain aspects of these vectors results in visualizations that are starkly different from the classic tree map proposed by Shneiderman in the early 1990s [48]. Those characteristics are not unique to tree maps, and some are shared across different types of visualizations.

Nesting offsets

"Classic" rectangular 2d tree maps have a *nesting offset* of 0. They show structure implicitly by only showing the leaf nodes of the hierarchy while the parent nodes are implied (figure 3.2a). This type of tree map "hides" parent nodes by using their entire screen area to render their children on top.

Nested rectangular 2d tree maps feature nesting offsets between parent and child nodes (figure 3.2b). They emphasize structural information, making it explicit, and allow displaying of content information on parent nodes – but do so at the cost of reduced space efficiency.

[51] shows that users who are familiarizing themselves with a new hierarchy often need offsets. Better understanding of the hierarchy leads to users preferring smaller offsets ($< 4\,pixels$).

Another possibility is to mix up nesting offsets. A map featuring nesting offsets could remove offsets between leaf siblings, forming leaf node clusters, to create a visual differentiation between leaf nodes and internal nodes [51]. Another option is to increase offsets of (and around) a focus node, while decreasing out-of-focus node offsets, creating a *fish eye* view.

Node shape

Since their introduction many different variations of tree maps have been created. One parameter of which is frequently changed to create a fundamentally new visualization is node shape. Briefly described here are some different tree map variations, that become possible when changing node shape. A comprehensive survey on tree map variations is performed in [41].

Voroni tree maps allow individual nodes to take any polygon shape. This makes them highly space efficient and also allows the maps to take on any shape (see figure 3.3) [9]. Differences in shape, and dense packing leave less options to enrich the visualization with content information. Voroni tree maps also use a nesting depth of 0, thus show structure implicitly.

Three dimensional icicle plots depict nodes as 3d boxes. They are a mix between enclosure diagrams and adjacency diagrams (section 3.3). Containment is visualized by placing children on top and within the area of their parents. The top-down view of this visualization is a non-nested rectangular tree map. Changes in perspective, opacity, etc. can put special emphasis on specific details. Putting the visualization into a 3d space adds potential complexity to navigation.

Radial tree maps (figure 3.5b, section 3.3), are generally also considered to be tree maps, but like 3d icicle plots they also contain adjacency diagram elements.

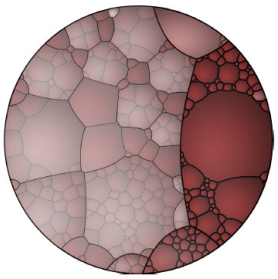


Figure 3.3: Voroni tree map (excerpt from [9])

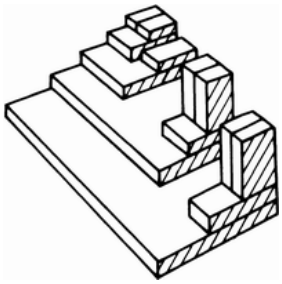


Figure 3.4: Icicle plot / tree map in 3d (excerpt from [28])

Node size

Node size in tree maps is often used to show quantitative content information. Tree maps (without nesting offset) are used to show e.g. shareholder distribution of stock markets, distribution of wealth, etc. In rectangular tree maps this can lead to ungainly aspect ratios, but other node shapes such as arcs or 3d boxes suffer the same problem. Difference in distortion (in aspect ratio or otherwise) makes it difficult to compare nodes in size. [49] writes that, "[Users] often drew arcs with their fingers back to the center to help make size judgements. Comparing two different aspect ratio rectangles in order to evaluate size with [tree maps] frustrated some participants just as much though.". Different layout algorithms exit for rectangular tree maps, trying to keep the aspect ratio close to 1, or otherwise improve specific aspects of the visualization.

Node size can also be used to put emphasis on data (both qualitative and quantitative). One could interactively tell the visualization to emphasize (by size) all nodes in a tree map that match criteria, specified by the user.

Alignment

The options available for alignment of nodes are influenced by node shape and size. Typically alignment is the result of a layout algorithm for the map, which tries to optimize for e.g. space usage or square aspect ratios.

Alignment can also be used explicitly to show structural or content information. Nodes can be aligned to mirror the order in the model, by criteria (e.g. lexicographically), etc.

Multiple views

The use of *small multiples* to enrich a visualization, is a general concept, applicable not only to tree maps. It's the use of multiple views which can be used to show aspects of the same data set. If for example the user wants to compare two nodes (that are spatially far apart), both nodes could be focused by creating two additional views that show only these nodes (each as the root of its own view) side by side.
Multiple tree maps containing data from the same domain, but taken at different points in time, could be displayed in multiple views – again side by side (or overlayed, etc.) to help the user discern trends in the data. There are many different ways in which multiple views could help the

user to better understand data.

Multiple views are a useful way to introduce new functionality into a visualization, however for every introduced view the user also needs to be able to navigate it. The number of views thus increases the overhead needed for navigation. Arranging, blending in, switching between these views etc. is a problem onto itself. The way the user interacts with small multiple views should be understandable and intuitive so as not to confuse (or mislead) the user.

Color, texture, border, pictograms, ...

These properties can be used to highlight interaction, e.g. in graphical user interfaces borders, colors, pictograms, etc. can indicate selection, highlighting or other interaction-based feedback like expand/contract icons. They can also be used to inform the user about the data itself: qualitative data (e.g. transparency for active/inactive or red flashing for value in critical range) or quantitative data (e.g. shades of color or thickness of borders).

3.3 Adjacency, alignment, and outline

In this paradigm the relative position between nodes and their alignment is used to express node to node relationships.

Adjacency and alignment

Adjacency diagrams, like enclosures, are space-filling. Nodes are drawn as solid areas. Their position within the hierarchy is shown via their placement in relation to adjacent nodes. They share many characteristics with enclosure-type diagrams such as tree maps, and are often referred to in the same context as tree maps, or as tree map variants.

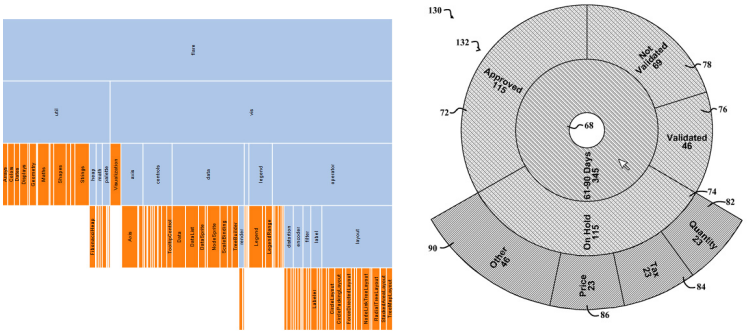
Figure 3.5a shows an *icicle*-type diagram. The top node is the root of the hierarchy. Children are placed inside the horizontal space spanned by their parents. A downside of this visualization is that nodes deep inside the hierarchy can become very small.

Figure 3.5b shows an example of a sunburst visualization – a radial version of the icicle diagram seen in figure 3.5a. The sunburst visualization addresses the issue that rectangular tree maps (section 3.2) have with displaying structural information – without nesting offsets only implicit, with nesting offsets less space efficient [50]. Structure is shown by placing children within the "arc"-space of their parents. Depth is depicted clearly by the distance from the center to a given node. Just like the rectangular *icicle* version, small outer nodes constitute a problem.

Indented outline

Indented outlines are specifically mentioned in [55]. They use node alignment and degrees of indentation to outline the structure of a hierarchy. This type of visualization is also referred to as tree listing. It is often found in file browsers.

There are similarities to node link diagrams, especially when edges are drawn for visual support (see figure 3.6a). Structure in indented outlines emerges as a result of the specific way



(a) Icicle tree (excerpt from [20])

(b) Sunburst visualization (excerpt from [36])

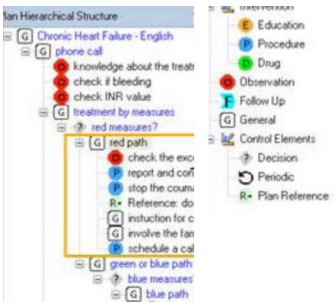
Figure 3.5: Adjacency and alignment based

nodes are arranged. Moving the nodes around would be indicative of changes in the represented hierarchical structure. This is contrary to a node link diagram where nodes can be moved around without changing the represented structure.

An interaction feature typically found in this kind of visualization is the ability to collapse and expand nodes. Nodes can be decorated with custom icons to better reflect the semantics of the hierarchy, as can be seen in figure 3.6b.

Tree listings are very simple and intuitive to use, however when displaying big hierarchies there is considerable navigational overhead when the user has to perform lots of expand/contract actions to reach the data he is interested in.

(a) Tree listing (excerpt from [55])



(b) Tree listings from Gesher (excerpt from [18])

Figure 3.6: Indented outline / tree listings

3.4 Discussion

A data-visualization problem has different requirements, some of which can be conflicting (e.g., dense and detail-rich representation). Each visualization paradigm presented in this chapter and variants thereof has its strengths and weaknesses. These paradigms can be combined and modified in various ways to create a solution (good or bad). Whereas different solutions can emphasize (or neglect) certain requirements.

As a general rule a good visualization should i) support orientation by striking a balance between showing local detail and keeping the global context, ii) minimize the cognitive overhead needed for navigation, and iii) increase expressiveness, by showing semantic details of the visualized model [12].

While normally it makes little difference if a pie chart or a bar chart is used, our understanding of the underlying information can be affected by how the data is visualized [29]. It is important to keep this in mind, and try to present data in an objective manner.

CHAPTER 4

State of the art – Software

This chapter takes a look at popular tools and visualizations that are currently used in the Asbru field. The tools are described with respect to their intended use and provided functionality, then compared to PlanStrips.

4.1 AsbruView

"AsbruView is a graphical user interface for viewing, creating and modifying Asbru plans. It is based on different 'views' which visualize different aspects of the plans. [The] second edition of AsbruView, [...] was created in 2004/2005 and allows the import of Asbru plans and their modification." [22]

AsbruView aims to show i) hierarchical decomposition, ii) mandatory vs. optional plans, iii) temporal order (and cyclicity), and iv) temporal uncertainty [25]. It features two main views to do so, the *topological view* and *temporal view*.

Topological view

TopoView is based on an *icicle plot* (see figure 3.4) – an example can be seen in figure 4.1. Plans are displayed as boxes in a 3d environment. The perspective is fixed and isometric (sometimes referred to as 2.5d). TopoView makes heavy use of a *racetrack metaphor*: The platforms (boxes) depicting plans are meant to be perceived as race tracks. Glyphs like finishing lines and traffic lights are used.

Hierarchical decomposition Decomposition is shown via stacking of boxes. When a box is lying on top of another box it is a direct descendant. The "stack level" corresponds to the depth of the hierarchy.

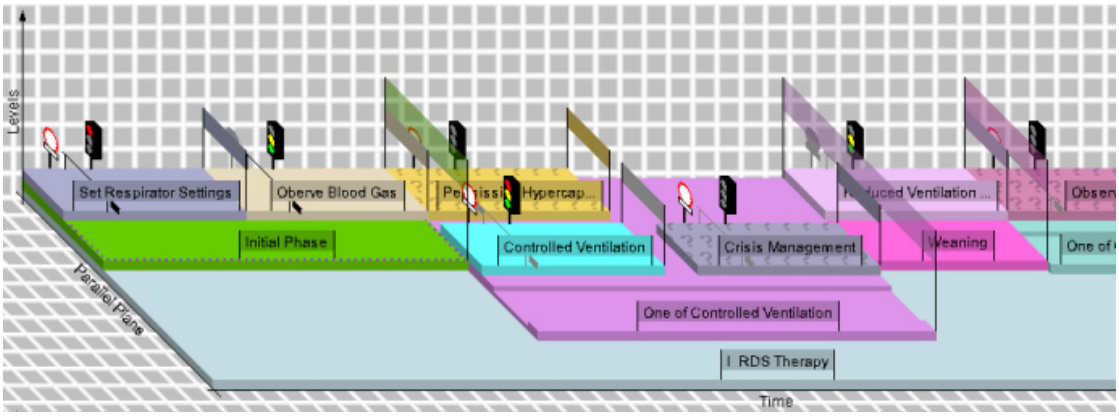


Figure 4.1: AsbruView's TopoView (excerpt from [25]

Filtering The view supports exploration using *filtering & drilling* (see section 5.5). Nodes with children feature a small triangle (at their right bottom). Clicking the triangle collapses and expands nodes. Figure 4.2 illustrates the expansion/collapse of the 2nd node from the left.

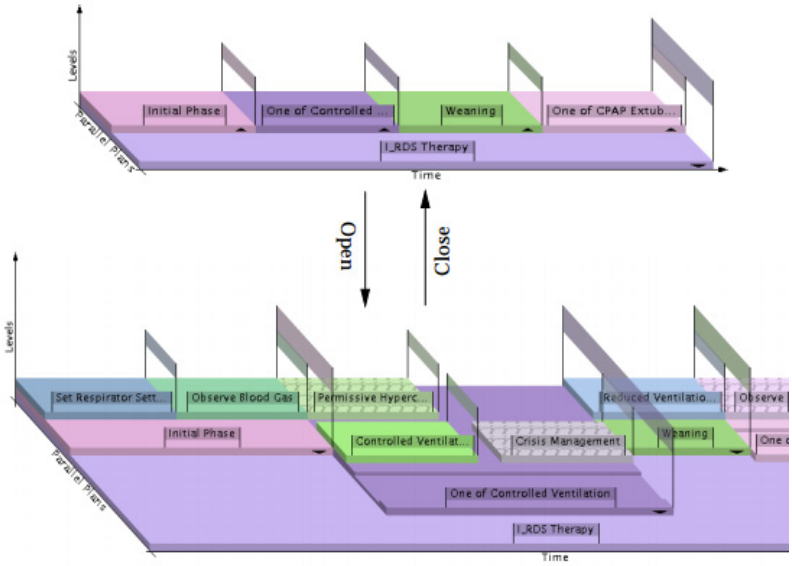


Figure 4.2: TopoView filter & drill (excerpt from [26]

Synchronization Plan synchronization is shown through the alignment of siblings. The left column in figure 4.4 shows some examples.

Sequential plans are positioned next to each other along the x axis. Seeing them as racetracks implies that a runner would go from left to right – one race track after the other.

Parallel plans etc. are positioned in a line along the y axis. The metaphor is that runners would line up and start running simultaneously.

Necessity Necessary plans are rendered using a solid color. If a plan is optional the solid color is covered with a transparent "?"-texture (see figure 4.1, 4.2).

Conditions Every condition is assigned a glyph.

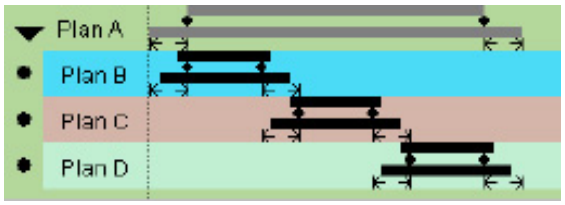
Setup preconditions are shown as start-barriers at the left side of the track. If the barrier is closed the setup condition is not fulfilled. Complete conditions are indicated using a finishing line. Abort, suspend and reactivate conditions are respectively shown using the red, yellow and green color in a traffic light.

These glyphs change depending on the semantic meaning of the used metaphor, and they are greyed-out if the condition is unspecified.

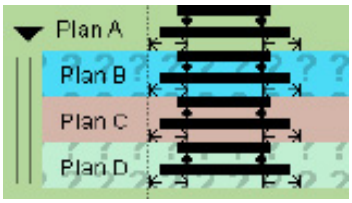
Temporal view

The temporal view is based on a *tree listing* (see section 3.3). Types of synchronization are shown using glyphs (e.g. dots for sequential plans – figure 4.3a, parallel lines for parallel plans – figure 4.3b, etc.). As is done in the *TopoView*, optional plans are textured using question marks. Like in most tree listings the nodes can also be expanded and collapsed.

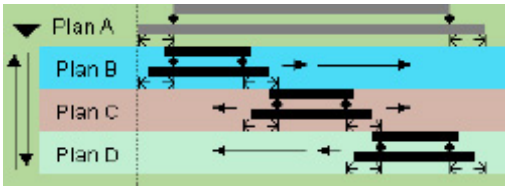
In the illustrations in figure 4.3 the black bars on the right sides show temporal detail – when and for how long plans will be executed in relation to each other. Minimum duration, maximum duration, earliest starting shift, etc. are shown. The technique used is inspired by *LifeLines* [35] and improved upon to deal with temporal uncertainty.



(a) Sequential



(b) Parallel



(c) Any-order

(d) Cyclical

Figure 4.3: Temporal view plan representation (excerpt from [26])

Discussion

TopoView is similar to *PlanStrips*. Figure 4.4 illustrates how both feature a very similar look.

While PlanStrips uses colors to express synchronization TopoView colors plans to (presumably) help the user distinguish them. AsbruView's authors rightly argue that color should be used explicitly to reflect parameters of the model, and accidental use of colors with cultural meaning (e.g. red, yellow) can be misleading [35].

The visualization of condition status is of limited use during the modeling process and can be disabled. However if the user wants to see unspecified ("missing") conditions, seeing the greyed-out glyphs could be useful. I would argue that, when enabled (during the execution phase), the forest of glyphs looks confusing and bloated when first seen. Everywhere traffic lights, flags, signposts, etc. are sticking out and have a weird look to them – see figure 4.1. The race-track metaphor is ok when looking at a cluster of sequential or parallel plans (figures 4.4a, 4.4c). The runner would go from left to right, running through sequential plans. Multiple runners would run simultaneously in parallel race tracks. But if big nested systems of runners start executing at the same time it would be an Olympic arena. Almost every visual in TopoView is based on this metaphor. So does it support understanding, or just introduce complexity? I think that the same goals can be achieved without locking the entire system into a one-trick metaphor.

The *temporal view* of AsbruView is more concerned with detailed presentation of temporal order. Something that (arguably) can't be done efficiently in PlanStrips (or TopoView). This feature is not one of PlanStrips goals. The temporal view also shows hierarchical decomposition (tree listing) and synchronization using glyphs (see figure 4.3).

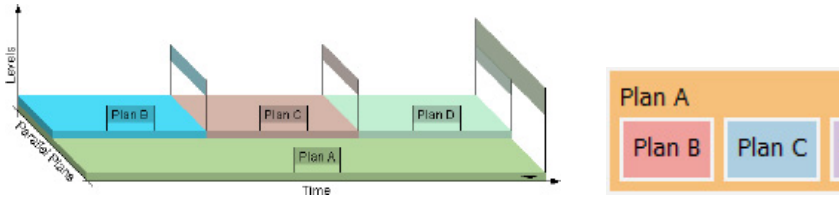
4.2 GMT/DELTA

GMT was designed "to support (i) the linking of informal and formal representation of guidelines to increase the structuring and understanding of guidelines in both representation and to trace back flaws and errors and (ii) the facility of design patterns to ease the authoring of guidelines in a formal representation " [52].

After more than two years of development the name GMT was changed to *DELTA* (*Document Exploration Linking Tool with Add-ons*).

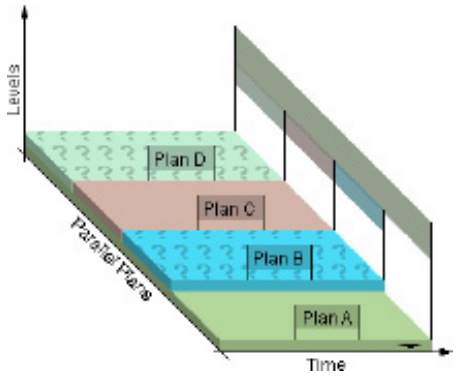
It allows the creation of links which connect HTML content (i.e. electronic HTML versions of guideline documents) with the Asbru file. These links help the user to reference values from the Asbru file back to their source. Once the computerized version of the guideline is created, it is not always clear where certain parts of the computerized version originated from. Additionally, it can be helpful to a user's understanding if he can view the original guideline from which a section of the Asbru file originated from.

Another feature of DELTA are macros which can be used to perform automated changes and modelling actions on Asbru files. This includes adding new nodes etc. For example, if a user wants to add both a new node and a source-link, he would select the reference in the displayed *HTML-view*, the place to add the node in the *XML-View* and finally execute an "add-node-with-link" macro from the *Structure View*.

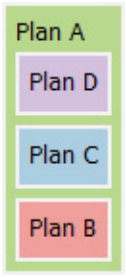


(a) Sequential

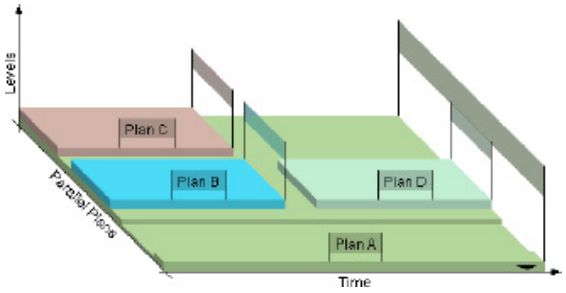
(b) Sequential



(c) Parallel



(d) Parallel



(e) Any-order

(f) Any-order

Figure 4.4: *TopoView* plan representation (*left*, excerpt from [26])
PlanStrips plan representation (*right*)

In terms of plan-visualization the *tree-listing* (2a) in figure 4.5 shows the Asbru XML's file structure.

The tool uses *brushing* to highlight nodes in the XML-View when a link in the HTML-view was activated or vice versa – figure 4.6.

Discussion

The *XML-view* looks tedious to work with because it involves much vertical scrolling, and XML nodes have to be constantly expanded and contracted. Additionally working on the native Asbru XML file structure requires detailed knowledge of the Asbru syntax.

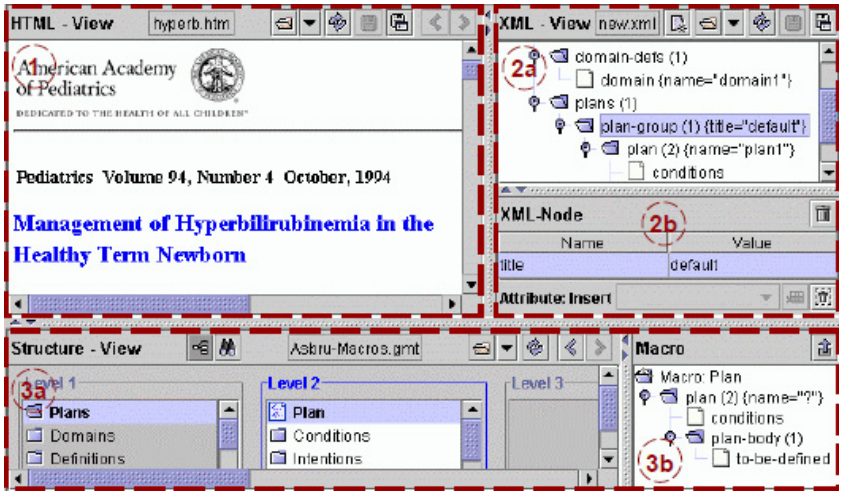


Figure 4.5: Three GMT Views (excerpt from [52]). HTML-View provides a native view of the original guideline. XML-View provides a tree-listing of the XML file structure. Structure-View breaks the guideline down semantically for the user.

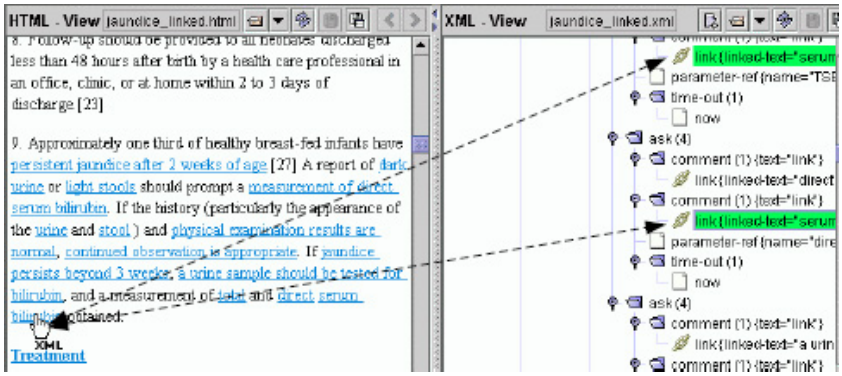


Figure 4.6: GMT - Link activation. When a link endpoint is clicked in the HTML view or the XML view, its counterpart is highlighted (excerpt from [52])

The ability to create and execute scripts is useful from a programming point of view. But the question arises when new scripts would be needed during the guideline translation process. The scripting is emphasized in the tool, but I would think that commonly used scripts could be integrated natively and hidden from a general user. Scripts can still provide a powerful tool when new functionality has to be added experimentally.

The ability to link sections of the Asbru file to their source material can quickly provide users with additional reference material. Original guideline passages can be useful for any user as a second source of information. A paragraph containing written-language can be easier to understand than the corresponding visualization part, or tie the content of *small multiples* semantically together. *Gesher* (section 4.4) takes this concept one step further.

4.3 CareVis/AsbruFlow

These two projects are tied in together closely. Strictly speaking AsbruFlow (like PlanStrips) is both the name of a concrete visualization and editor prototype. In this section I won't try to differentiate when talking about the two prototypes, and instead refer to the project as *CareVis*.

CareVis tries to communicate the logic of an Asbru plan during execution or analysis of a plan or for educational reasons [6]. The visualization is made up from two main views (logical, temporal) and two helper views (quick, overview). The views are connected via *linking & brushing*.

QuickView

The QuickView displays parameters and variables that arise during the course of treatment (e.g. patient name, blood values, etc.). The view provides these values as a list of facts in a small panel visible in the top of the application. It is used to show data available at execution-time.

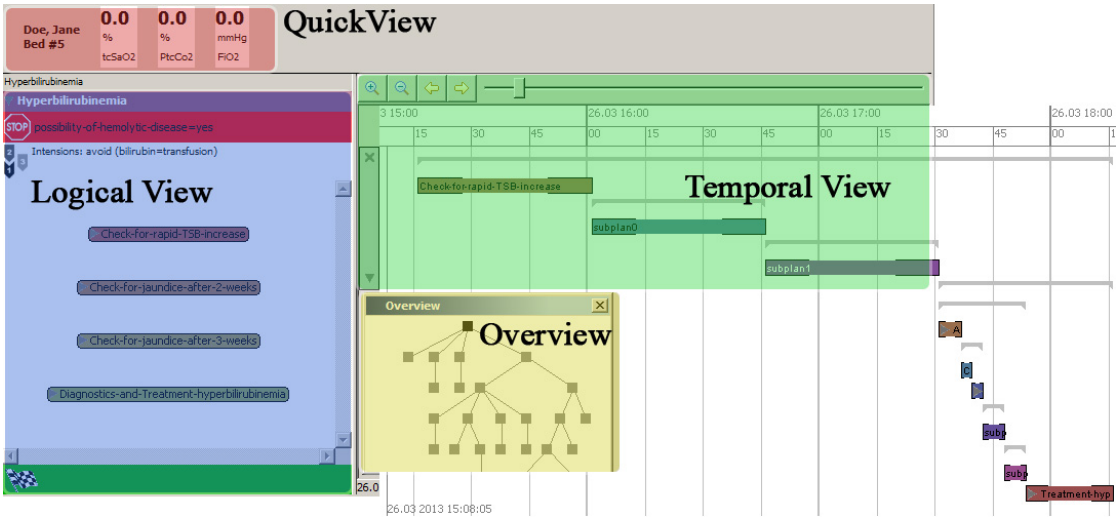


Figure 4.7: CareVis Views

Temporal View

This view uses a visualization glyph called *PlanningLines* (*LifeLines+*) [7]. Figure 4.8 illustrates how a PlanningLine conceptually represents an entity. They can also be seen in figure 4.9. PlanningLines are an extension of the concept of LifeLines. They are meant to visualize task/process duration despite temporal uncertainty (e.g. start-time within a time-interval and not fixed time-point), and to enable the display of hierarchical decomposition.

The grey triangle seen at the left end of the Life+/PlanningLines is rendered when the plan contains sub-plans and can be expanded. Children then can be further expanded (figure 4.7). In effect this shows *hierarchical decomposition* (exploration by *drilling*). The view spreads plans

on a timeline. Only plans having a time-annotation (or already executed plans) can be accurately displayed on the timeline.

Below the treatment plans the temporal view can display patient data (and its behavior over time) that becomes available during execution. The timeline can be distorted using fish-eye. Vertical portions can be partially collapsed to take up less space (by showing less detail).

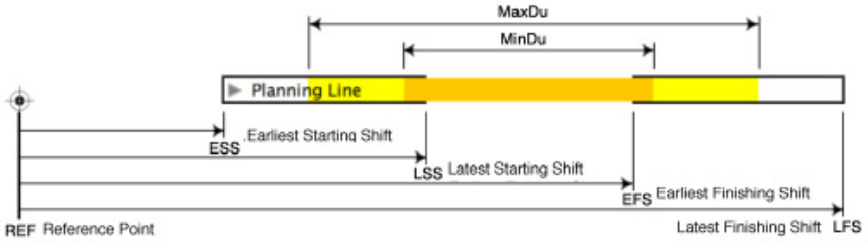


Figure 4.8: PlanningLine glyph [5]

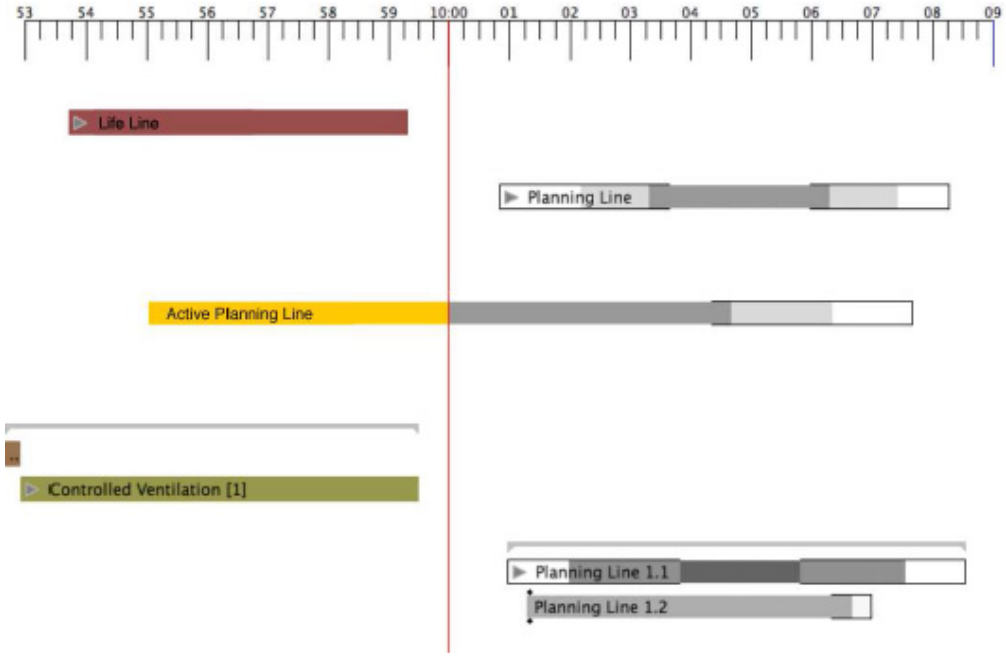


Figure 4.9: Temporal view elements (LifeLines+, PlanningLines) [5]

Logical View

This view mixes *containment* with *node-link* representation. According to [5] medical experts are already familiar with flow charts, because of an existing representation standard (called *flow-chart algorithms* or *clinical algorithm maps*).

Plans are represented in boxes as depicted in figure 4.10. Shown are the plan name, abort and complete conditions, synchronization and intentions (in the prototype but not the graphic).

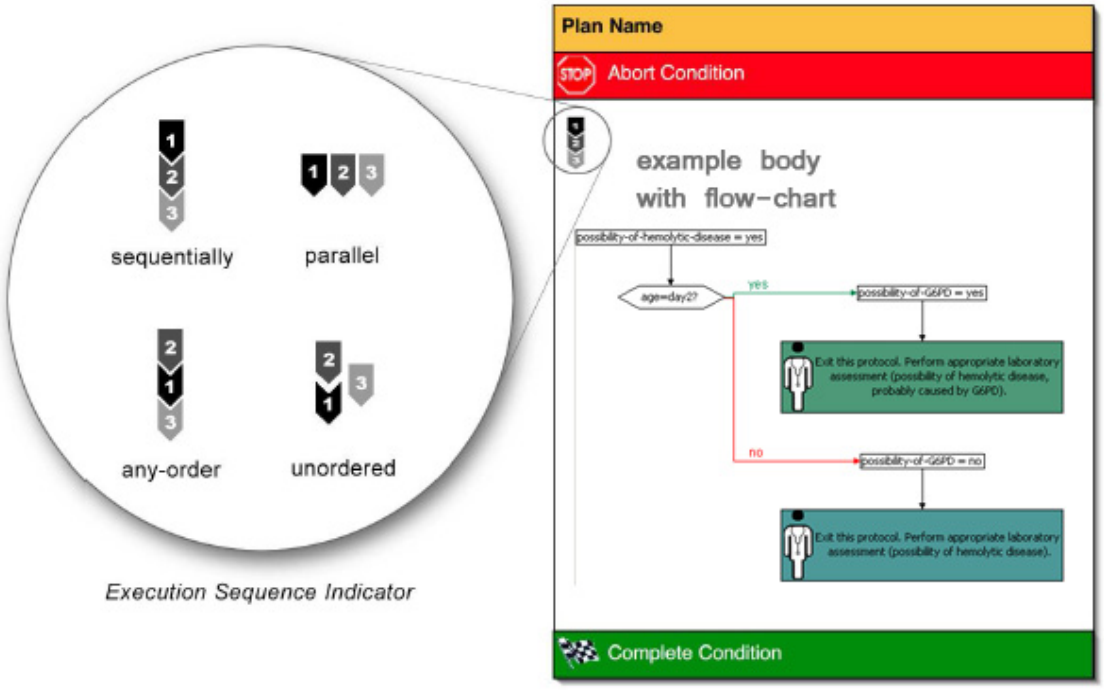


Figure 4.10: Plan body in logic view [5]

The main area of the plan shows the contents using flow-charts. In the example figure a flow chart depicting a variable assignment and if-then-else can be seen. Plans are shown using small colored nodes containing the plan name. Nodes with children can be expanded by clicking on a small triangle. The view only shows the currently focused plan and provides two modes of navigation.

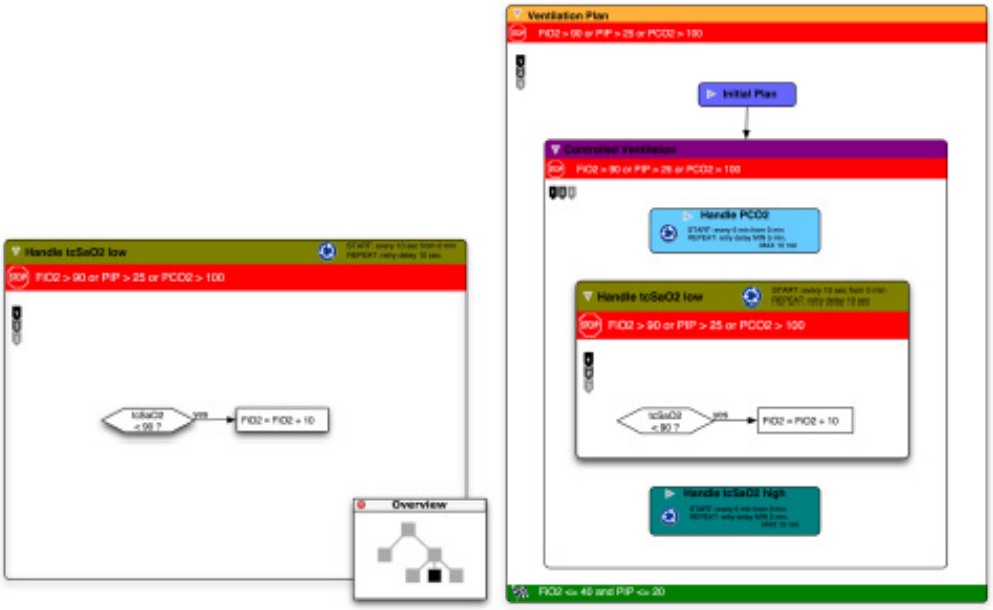
Translation of plan content into a flow charts is done by transforming constructs like if-then-else into according flow-chart elements. Variable assignments, *ask actions*, *user performed plans* and other Asbru specific details are visualized using special elements (see figure 4.10).

Overview+Detail Expanding a plan in this mode will trigger a transition effect that zooms into the node, showing its contents. In order to preserve the context an extra window containing an overview (node-link tree graph) is shown (figure 4.7, yellow area) to keep track of the place of the currently focused node in the hierarchical structure. See figure 4.11a.

Fisheye Basically a *filtering & drilling* approach. When nodes are expanded the view does not transition but instead expands the node into a full plan-box (figure 4.11b).

Overview

The overview can be seen in figures 4.7 (yellow area) and at the bottom right of 4.11a. It provides a simple aesthetic tree graph where each node is rendered as a small black rectangle with no



(a) Overview+Detail (b) Fisheye view

Figure 4.11: Logical view showing parts of the Asbru plan for artificial ventilation of newborn infants – Overview+Detail mode (left) vs. Fisheye view mode (right) ([5])

additional information. “Field of depth” is applied through color by rendering the focused plan in black and unfocused ones in a neutral grey.

Discussion

The Temporal View

Temporal View seems to satisfy its purpose of displaying temporal detail of plans. Hierarchical decomposition is technically shown but practically it seems only useful to expand one/two plans for some additional information; not for browsing or understanding the full structure. Plans lacking temporal annotations can not be shown accurately. Also it is conceivable that plans which semantically belong together could be separated by a large distance on the time axis. Conditions, synchronization, etc. are not shown (synchronization could be shown using colors like PlanStrips does).

I think that Temporal View (as its name suggests) displays temporal aspects very well, but it is ineffective in showing hierarchical decomposition (especially on a larger scale). The layout this view has by necessity (PlanningLines, LifeLines+ being horizontal bars) should best be kept clean and simple to be understandable – displaying e.g. conditions would result in too much clutter. PlanStrips completely lacks the ability to show temporal information (other than simple synchronization), but is good at displaying other Asbru facets. Taking this into consideration I think that PlanStrips and the Temporal View of CareVis would complement each other well.

Logical View

Logical View shows intentions, conditions, ask-actions, if-then-else, etc. it is very verbose in showing these details. A user-performed plan can for example contain the full string "Perform appropriate laboratory assessment, including possibility of cholestatic jaundice". Local detail is presented in a nice way. Based on the argument that medical practitioners are already familiar with flow charts it is plausible to incorporate them to illustrate the processes of variable assignments, if-then-else, etc. This is well done.

The two viewing modes (figure 4.11) that the user can choose from are:

"Fisheye view" (figure 4.11b) that uses exploration using *filtering & drilling*. In this approach the view makes strong use of the containment metaphor (as does PlanStrips). Local detail of a plan is presented well in this view. Hierarchical decomposition is displayed technically but if many nodes are expanded it becomes messy and hard to understand. PlanStrips does something similar but structures and visually presents plan-boxes in a more ordered way that seems more intuitive.

"Overview+Detail" is the second viewing mode (figure 4.11a) which semantically splits the view in two. One of PlanStrips' goals in section 5.1 was "support orientation by striking a balance between showing local detail and keeping the global context". The detail part of the view shows only local detail but no global context at all. The overview part shows only structure with no detail at all. The problem here is that the user has to remember which node (black rectangle) in the overview corresponds to which detail-containing plan box. Having two views also means the user has to coordinate between them which adds extra overhead.

The conditions displayed at the top and bottom of the plan boxes are very verbose. Like PlanStrips' *ConditionView*, the LogicalView translates the conditions into strings using a simple custom syntax – it seems that the string is a conjugation of the current node's and its ancestors' conditions. These strings in theory can grow to be pretty long. Truncating them would render displaying the condition somewhat useless and using line-breaks could result in many lines (especially with narrow boxes) negatively impacting readability. These strings shouldn't be put into the visualization directly. The detail should be shown on demand or externalized into an extra view. The view itself (like AsbruView's race tracks, section 4.1) should only indicate if a condition is i) fulfilled or ii) unfulfilled or iii) unevaluated or iv) unspecified .

The user should always have the option to toggle on the small "**overview**" window. If he has enough screen space (resolution, dual monitor) a compact aesthetic tree-graph could serve nicely as a *minimap*. I would extend it to use synchronization coloring and special icons (instead of just black boxes) for ask-actions, user-performed plans, etc. This would add more information without changing the structure and small footprint of this view.

In summary I believe PlanStrips does a better job at showing hierarchical decomposition than the Logical View. The most convincing concept used in the Logical View appear to be the flow-chart translations of logical operations (e.g. if-then-else) which are rendered as part of the local detail. This is something that is missing from PlanStrips and could be included there.

4.4 Gesher

Gesher is part of the DeGeL framework which, besides Gesher, features a knowledge base server and runtime application engine for clinical guidelines. The server architecture is made up of the following modules: (1) a guideline database that contains the overall schema to support the hybrid multiple ontology representation (2) a module which is responsible for guideline-knowledge creation, reading, updating, and deletion (3) a guideline search engine (4) an authorization module, supporting the group-based authorization model (5) a web-service API that enables the guideline knowledge-base server to accept client requests and to orchestrate multiple steps when performing transactions [19]. Here we focus on Gesher, a graphical framework that concerns itself with guideline specification.

The authors of Gesher distinguish between two types of knowledge contained in a guideline. Both types are formally expressed in an underlying ontology (here Gesher currently supports Hyrbid-Asbru – a customized Asbru format).

Declarative knowledge are concepts that exist in a specific context (e.g. Hypothyroidism = TSH <= 0.4 IU/ml and FT3>4.2pg/ml). For simple expressions Gesher provides the *Expression Builder* module. For standard terms the tool provides access to different controlled medical vocabularies. These can be searched using the MEIDA system [44], which includes a vocabulary server and a search engine. For complex expressions Gesher interfaces with the temporal abstracted knowledge acquisition tool (TAKAT), which is used in the Idan architecture [14].

Procedural knowledge describes the guideline's control flow.

Ontology independence

Gesher (DeGeL) is not tied to any specific ontology (e.g. Asbru). The framework itself is generic and supports different ontology implementations (Asbru being a reference implementation) by internally working on a meta ontology. A target ontology is realized using *knowledge roles* (in Asbru a knowledge role would be e.g. a *filter-precondition*).

The yellow and green areas in figure 4.13 show an example of a guideline being translated from the free-text to the semi-structured level respectively using the *GEM* and *Asbru* ontology. The authors of the tool have modified Asbru to support their intermediate guideline translation levels, referring to the intermediate-ontologies as hybrids (e.g. hybrid Asbru). The semi-structured version of Asbru effectively is a simplified version with less expressive syntax. It contains most of the knowledge roles present in Asbru (plan-body, conditions, etc.) and is capable of describing time-annotations and simple temporal constraints. Hybrid (and formal) ontology representations are managed by DeGeL in a single structure.

Representation levels

One unique feature of Gesher is the support of abstraction levels to represent a guideline. The reasoning behind this, is that the expert physician is an expert regarding information contained in the guideline, but can't be expected to deal with the low-level ontology. At the same time the

knowledge engineer is proficient in the use of the ontology but has very limited understanding of the medicinal knowledge contained in the guideline.

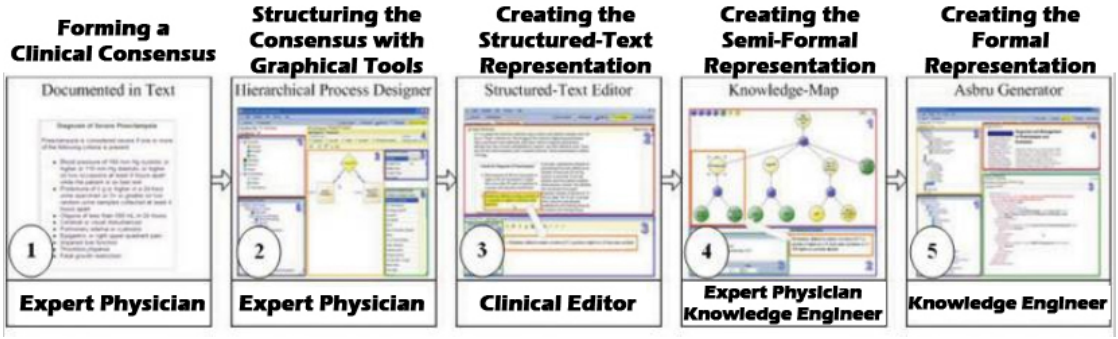


Figure 4.12: Gesher guideline specification process (excerpt from [18])

Figure 4.12 shows the incremental translation process. The abstraction levels from left to right are (1) informal free-text guideline (2) semi-structured (hierarchy and raw procedural aspects) (3) semi-structured (declarative knowledge markup) (4) semi-formal (5) formal . An expert physician (or clinical editor) with minimal training in the used ontology can extract declarative knowledge from the free-text guideline easily by identifying knowledge roles (of the used ontology) in the free-text guideline and marking them up. Figure 4.13 shows the mark-up tool used to extract declarative knowledge while translating the guideline from an informal to semi-structured state. The red area shows part of the free-text guideline which the expert physician directly translates to a hybrid Asbru filter-precondition by extracting and formatting the text (blue area). The second version of Gesher also allows the expert physician to extract simple procedural knowledge using graphical tools (step 2 in the graphic).

After the expert physician (or clinical editor) extracts the declarative information from the guideline and creates a first draft of the hierarchical structure and procedural information, the knowledge engineer can use the semi-structured version to create the semi-formal guideline. The knowledge engineer then creates the (final) formal guideline from the semi-formal one.

During this process the expert physician and knowledge engineer can collaborate with each other. If the expert physician is unsure as to the (hybrid) ontology details he can ask the knowledge engineer during creation of the semi-structured guideline, while the knowledge engineer can ask the expert physician about medical information while creating the semi-formal guideline. The general idea however seems for the expert physician to create the semi-structured guideline mostly autonomously, and for the knowledge engineer to do the same with the semi-formal version.

Structured Text Editor

This tool as seen in figure 4.13 allows the user to mark up text and drag it into the corresponding knowledge-role widgets, in the figure this can be seen for an Asbru filter-precondition. Other user interface tabs containing different knowledge roles are also visible. These areas can then

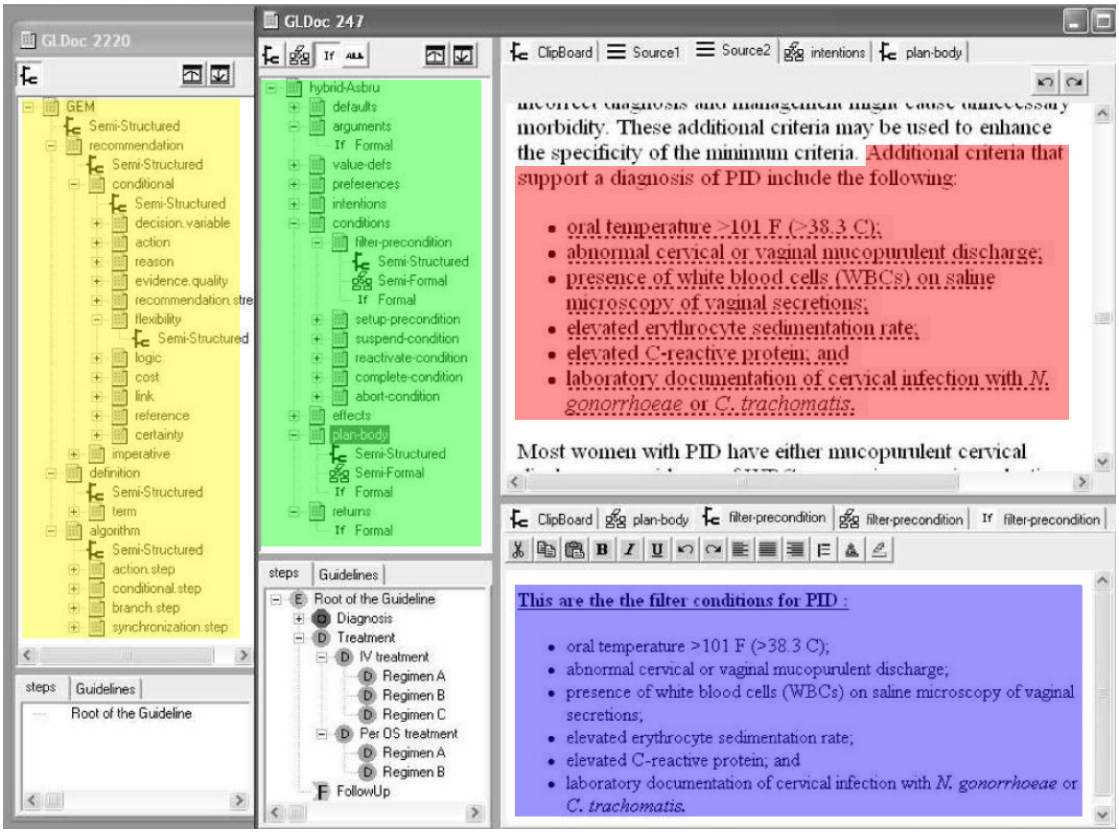


Figure 4.13: Gesher semi-structured markup tool (old version) (excerpt from [45]). Red area: Part of original guideline containing a condition. Blue area: Condition identified as filter-precondition and isolated in semi-structured guideline. Yellow/Green area: Tree-listing of the guideline structure in GEM (yellow) and Hyrbrid-Asbru (green).

be edited by the user using a WYSIWYG editor. The knowledge roles of the semi-structured guidelines are linked to the free-text guideline for referencing.

At this point the user can also link raw data and declarative knowledge concepts to the aforementioned medical vocabularies (LOINC, etc.).

As both the free-text and semi-structured guideline versions are text based this view does not use visualizations. The view focuses on declarative knowledge and is aimed at clinical editors or expert physicians, it corresponds to the third translation step in figure 4.12.

Hierarchical Plan Builder

The hierarchy plan builder seen in figure 4.14 is used to create a draft of the plan hierarchy (plans and sub-plans) and thus model the procedural flow (i.e. procedural knowledge) of the guideline. By design this tool can be used with ontologies created around the block-based hierarchical plan and sub-plan approach [45].

Panel *1* contains a palette of semantic plans that the user can add to the structure. Panel *2* shows the procedural flow of the currently focused plan. Semantic plans can be (1) procedures, (2) drug administrations/prescriptions, (3) observations, (4) educational steps, (5) follow-ups, (6) decisions (if-then-else in Asbru), (7) general (generic) plans, and (8) reference-plans (reference to existing plans) . Panel *5* shows the hierarchical decomposition of the guideline in a tree-listing while panels *3, 4, 6* are used to define plan properties etc.

The plan hierarchy can be navigated either through the flow diagram (panel 2) or using the tree listing (panel 5). The tool is mainly aimed at expert physicians and corresponds to the second translation step in figure 4.12.

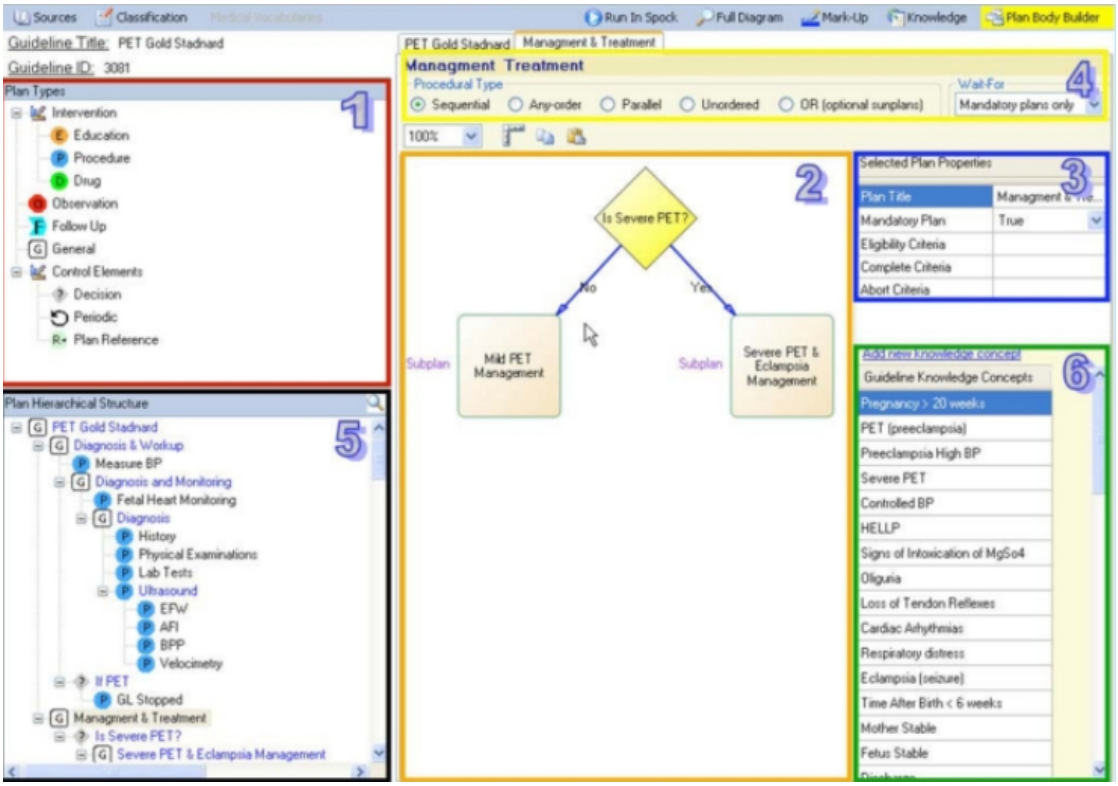


Figure 4.14: Gesher's Hierarchical Plan Builder (excerpt from [18]

Knowledge Map

The *Knowledge Map* (figure 4.15) is used during creation of the semi-formal guideline. It is used to translate the marked-up and formatted (free-text) knowledge roles into formal expressions. These knowledge roles were defined in the *Structured Text Editor* during creation of the semi-structured guideline. The Knowledge Map thus deals with declarative knowledge.

The map displays the declarative concepts and their relations in a node-link diagram. Selection of a concept displays its semi-structured free-text version (panel 2). The value, time, etc. constraints of the concept are edited in panel 3.

Concepts of the knowledge map can be:

Primitives, dates and events describe quantitative values or qualitative values (e.g. low/high). These are used to express things like blood pressure. Such measurements can be associated with a time stamp indicating when the measurement was taken. Dates and events (e.g. date of birth) specify concepts that carry no additional numeric or qualitative values.

Abstractions are composite concepts that decompose into other concepts. These composites describe the relations (e.g. and, or) and logic value/temporal constraints (e.g. greater-than, equal) to their sub-concepts. The authors call this *abstracted form*. An example would be "Elevated Liver Enzymes": SGOT>60 AND SGPT>60.

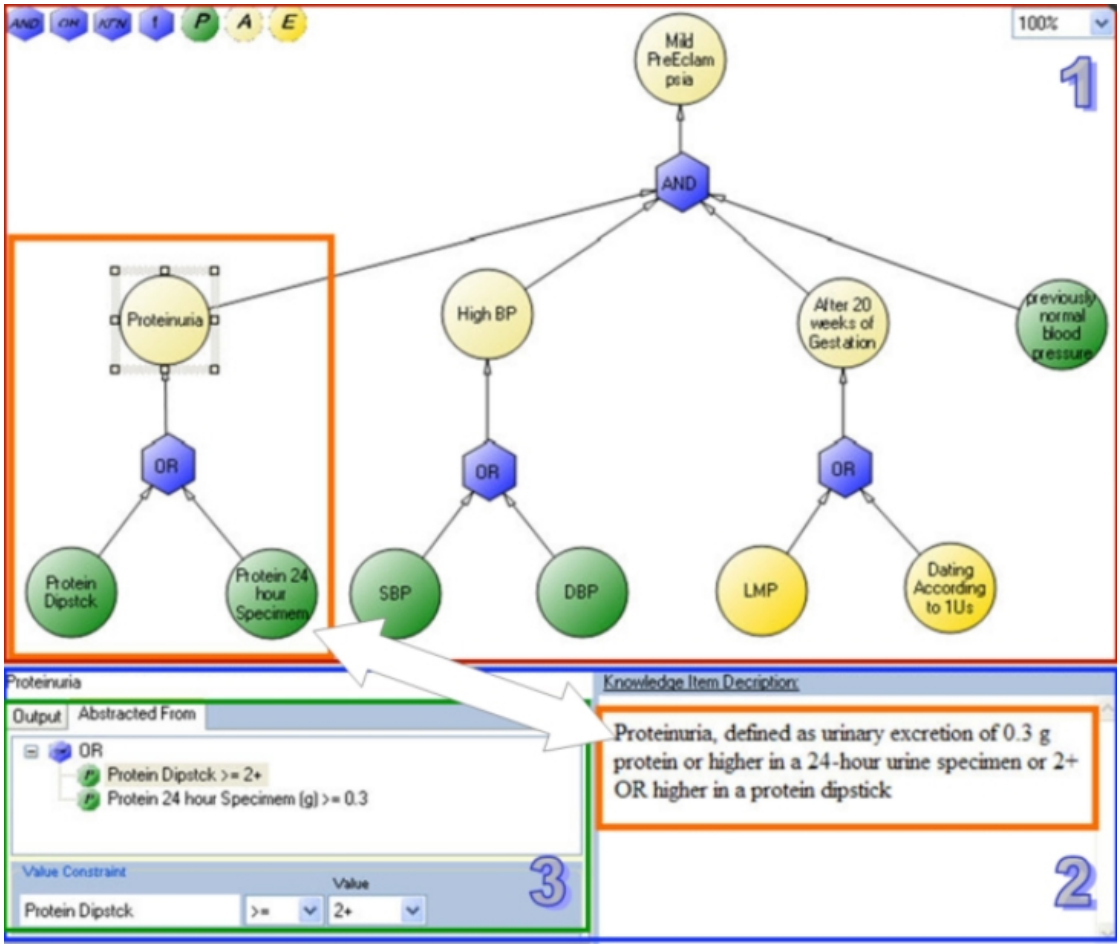


Figure 4.15: Gesher's Knowledge Map (excerpt from [18]

Formal Representation

This interface is used in the final step – the creation of the formal guideline. It is thus specific to the implemented ontology (Asbru), however the framework architecture allows for implementation of other ontologies as Gesher internally uses a meta-ontology.

Using the data created in the previous steps Gesher can generate a native Asbru XML file. Linking between the procedural and declarative aspects of the guideline has to be performed by a user (knowledge engineer). The interface seen in figure 4.16 allows the user to select a plan from the hierarchy (panel 1), and then browse through the plan's contents by selecting knowledge roles from panel 2. The native XML then appears in panel 3 while panel 4 shows the original source guideline (previously linked to the knowledge roles during mark-up). The knowledge engineer is also required to verify the correctness of the generated guideline, for which the tool offers a *debug mode* that can test runtime aspects of the guideline.

As can be seen in the graphic the hierarchy is again displayed using a tree-listing, as are the knowledge roles of the underlying ontology.

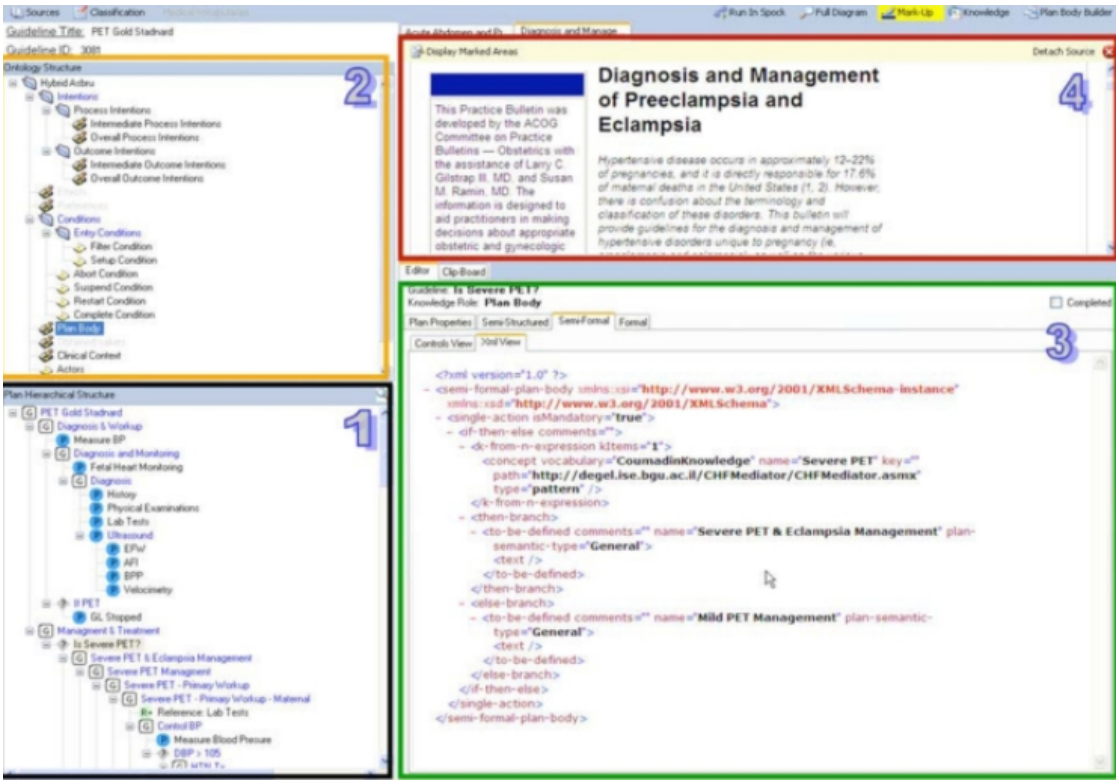
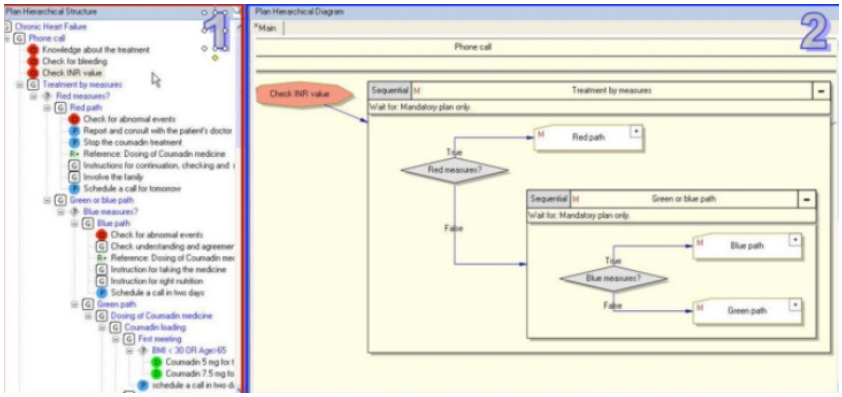


Figure 4.16: Gesher's formal guideline creation interface for Asbru (excerpt from [18])

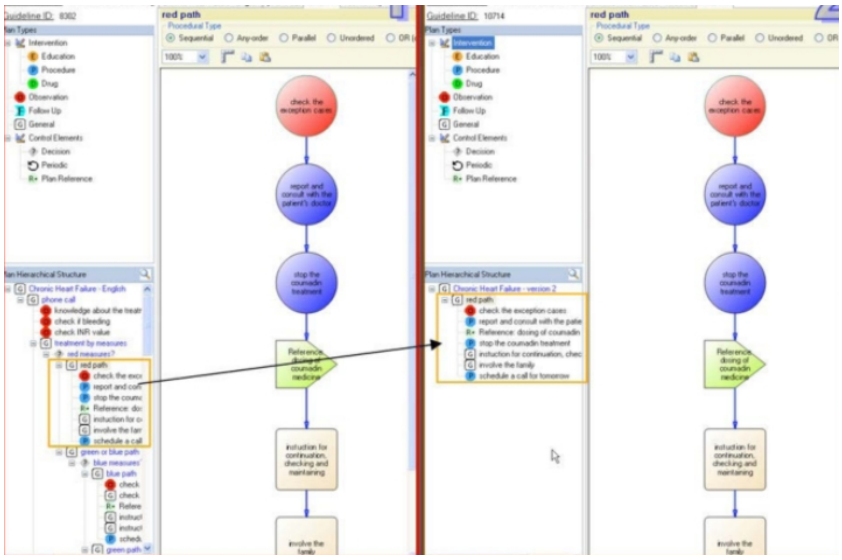
Exploration

For guideline exploration Gesher also uses the tree-listing seen in the previous steps, which displays the plan hierarchy. When selecting a plan Gesher displays its procedural aspects. This can be seen in figure 4.17a where a plan is visualized using a flow-chart.

The second interface seen in figure 4.17b displays both hierarchical decomposition and procedural information. It can be used for navigating the guideline or, if multiple guidelines are opened concurrently, to move/copy parts (e.g. subplans) between them.



(a) Exploration using flow-charts



(b) Exploration / copying between guidelines

Figure 4.17: Gesher guideline exploration (excerpt from [18])

Discussion

As far as translating and authoring of formal computerized guidelines, Gesher's functionality in its completeness exceeds the other tools presented here. The architecture of the framework is distributed via web services and allows for different front-end implementations and realization of other ontologies than Asbru. It should be mentioned, not all of Gesher's functionality was introduced in full detail.

The approach of incremental guideline specification makes much sense. It allows for different user roles to work on a guideline (even simultaneously). The free-text guideline markup and knowledge-role extraction can be performed by any user who possesses rudimentary knowledge of the ontology. However guideline translation is performed in practice – the modular approach and separation of concerns allows for flexibility.

The existence of concepts through different guideline levels effectively subsumes *DELTA*'s linking ability (section 4.2). If a change has to be performed in the guideline the user can either work his way from the formalized version back to the free-text guideline or the other way around. This covers both the use-cases of having to change the formal specification because of changes in the free-text version or having to track down an error.

Gesher seems to be the only tool that explicitly provides a way to define declarative knowledge using visualizations via its *Knowledge Map*. Other tools (including PlanStrips) mostly concern themselves with showing hierarchical decomposition and temporal aspects of plans (i.e. procedural knowledge). Visualization of these declarative knowledge concepts would not fit well into hierarchy/time visualizations, but it should not be ignored. Many papers dealing with guideline visualization and translation hardly even mention declarative knowledge visualization.

As far as process hierarchy visualization, Gesher (like many other tools) uses a simple tree listing. This tree listing isolates visualization of hierarchical decomposition from other ontology or guideline specific details. This means that the user can focus on the actual hierarchy without being distracted by additional clutter. Selection of a plan in the hierarchy (depending on the current view) presents the user with additional detail. Figure 4.16 is a good example of this – panel 1 shows the hierarchy, panel 2 displays the knowledge roles of the ontology, panel 3 shows the section of the formal guideline's XML while panel 4 shows the free-text guideline. The same approach is taken in figure 4.17a where selecting a plan in the hierarchy displays its procedural aspects in a flow chart. This separation of hierarchical decomposition and detail is reminiscent of *CareVis*' *Overview+Detail* view (figure 4.11a). The advantage here is that throughout different guideline specification levels the basic type of navigation (the hierarchy tree-listing) stays the same while detail is provided in different ways best suited to the current view. Still navigating the hierarchy using a tree-listing seems awkward because of expand/collapse events and vertical scrolling. The navigation overhead is higher than it should be, but because of the clean way the tool abstracts the guideline it should nevertheless be easy to understand.

The flow-chart display of procedural plan logic as seen in figure 4.17a from an aesthetic point of view looks better than the odd nexted-box-flow-chart hybrid of *CareVis*' *Fisheye view* (figure 4.11b).

I believe that one thing which Gesher shows is that conveying understanding of a guideline can hardly be done in a single visualization such as PlanStrips but requires multiple views which are linked and brushed together and can be navigated in a comprehensive way – using consistent

naming, pictography, colors, navigation elements etc. While Gesher's visualizations (mostly simple tree-listings and node-link diagrams) are generic and simple the multi-level abstraction and completeness of the tool add a level of sophistication the other tools lack.

CHAPTER 5

Implementation

PlanStrips is the name of a visualization based on nested tree maps (see figure 3.2b). Its main focus lies on (interaction-friendly) displaying of hierarchical structure. Each box in the nested tree-map corresponds to an Asbru plan. The visualization is meant to help users better understand the guideline on a procedural level. The tree-map format is also well suited for user friendly editing.

The visualization was created for use with block-based languages. The software prototype supports Asbru (a representative of block-based languages).

This chapter describes in detail the PlanStrips visualization and complementary visualization gadgets from the prototype software. Chapter 4 contains an overview of other visualizations and tools from the Asbru domain.

5.1 Goals, Requirements

Functional In short the functional goals of PlanStrips are to i) convey to users an understanding of the processes described in an Asbru guideline, and ii) provide editing functionality to modify the plan-hierarchy and specific plan parameters (e.g., plan-names) .

Visually PlanStrips shows *hierarchical process structure*, *plan synchronization* (sequential, parallel, etc.), and *content information* (plan names, conditions, etc.).

Editing functionality includes standard file operations (open, save, and create new guideline) and the following in-editor operations: plan creation, plan deletion, moving existing plans around the hierarchy, and editing of plan properties.

Non-Functional The (important) non-functional goal is for the tool to be as simple and intuitive to use as possible. Optimally a novice user who is familiar only with the most basic concepts behind medical guidelines, and of adept experience with computer software in general (e.g., Word, Excel) should be able to work out how to use PlanStrips quickly.

[46] states that clinical editors with mark up experience can structure guideline knowledge with a high degree of completeness, the main demand for correct structuring is to understand the

used ontology (i.e., Asbru). Reducing this main demand (understanding of the Asbru-ontology) should therefor allow users to quickly pick up and work with the tool. This is achieved by abstraction (e.g., plan-subplan relations are depicted by nested boxes, complete-conditions become check-icons, etc.) and omission (non-vital information in the current context is not displayed or can be toggled). The user should not be overwhelmed with information, but still be able to dig deeper into the ontology to find it.

Hierarchies are a natural and easily understandable way for humans to structure information. This is reflected in organizational/social structures, the way we categorize nature using taxonomic ranks, *work breakdown structures*, family trees, etc. Conveniently the skeletal-plans of Asbru guidelines are also structured in hierarchies. PlanStrips leverages this by correlating the guideline's structural hierarchy with a visual hierarchy. The assumption being, that putting emphasis on the hierarchical nature of the process structure will make it easier to understand, as juxtaposed to e.g. a node-link diagram (excluding tree graphs).

Use of the inherently space-efficient tree-map results in a relatively high number of simultaneously visible elements, allowing users to keep the context of a node, they are currently looking at, in their peripheral vision.

Finally PlanStrips also aims to give the user instant visual feedback (wherever reasonable) to help with orientation.

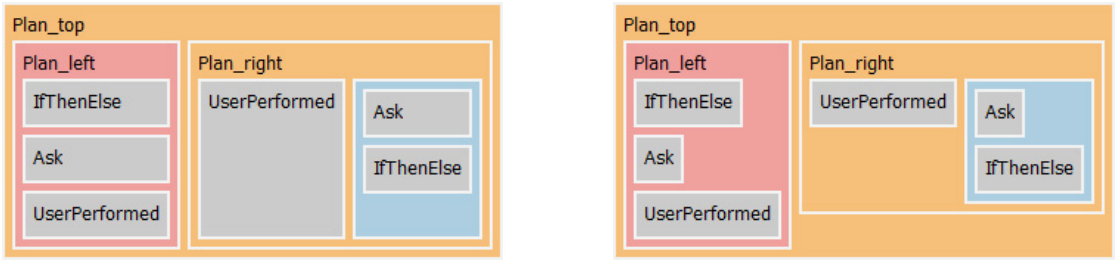
5.2 The Visualization

Nested Tree Map A nested rectangular tree map is chosen as the basis for the visualization. The mix of a high visible node count (spatial efficiency) and pronounced visualization of structure (nesting offsets) presents the user with more global context.

Boxes inside boxes The visual metaphor of nested boxes directly translates the hierarchically ordered data into a visual hierarchical. This presentation is vivid and simple to grasp, and should therefor enable the user to build a mental model of the underlying data structure. The rectangles also fill in as small display areas, and are well suited to display labels and pictograms in them.

Spatial efficiency A dense representation allows for many nodes to be visible simultaneously. This reduces the total (mechanical) navigation that is required by the user. The use of nesting offsets results reduces spatial efficiency so that fewer plans can be displayed. This number however is still high. The hit to spatial efficiency (caused by nesting) is acceptable because nesting offsets put an emphasis on the hierarchical structure of the guideline. The number of plans (nodes) and nesting depth in average guidelines (judging by the two reference implementations) should not be high enough for the cumulative nesting offsets to become a problem.

Horizontally aligned nodes take up all available vertical space, and vertically aligned nodes all horizontal space. This makes siblings easier to identify because they always have the same size along one axis. No space is wasted, because available space is determined by the biggest sibling. The difference can be seen between figure 5.1a (filling) and figure 5.1b (non-filling).



(a) Siblings filling minor axis

(b) Siblings without filling

Figure 5.1: PlanStrips with and without boxes filling available space

Plan synchronization Plan execution order (i.e. synchronization) is shown using the color scheme proposed in [42], and influenced by [15] and [10]. Each type (sequential, parallel, any-order, etc.) is assigned a distinct color. User performed plans are also assigned a special color.

Child ordering The ordering between siblings is determined by their parent's type of plan synchronization. Sequential ordering is represented by aligning boxes horizontally (left to right), and parallel ordering by aligning vertically (top to bottom).

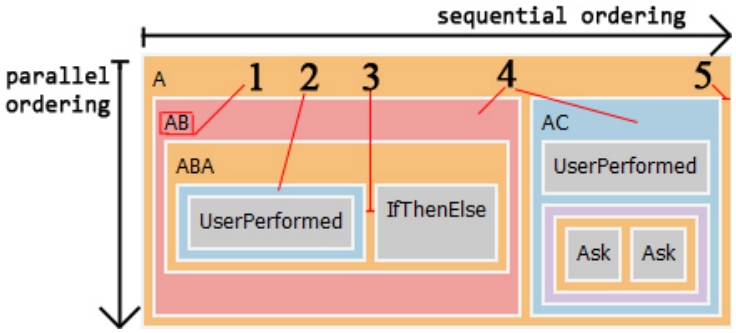


Figure 5.2: Simple hierarchy using PlanStrips

Figure 5.2 showcases a simple hierarchy visualized using PlanStrips, where:

1 Area used to display content information about the model. In this particular case the plan name is shown inside a label (the red box containing the label *AB*).

2 Anonymous *subplans* element (no label).

3 Offset (horizontal/vertical) between siblings

4 Different colors showing plan synchronization

5 Nesting offset (parent to child)

Condition view

Each plan can have a number of conditions associated with it, e.g. *setup preconditions* specify conditions which have to be met before a plan can be executed. However the conditions of the plan's ancestors also matter. Clashes in logic between the conditions of a plan and one of its ancestors can be indicative of a design error. Conditions are also important for understanding the semantics of a plan.

The *condition view* is a UI widget that displays the conditions associated with a plan and it's ancestors in a table. Using a drop down menu the user can choose which condition to display.

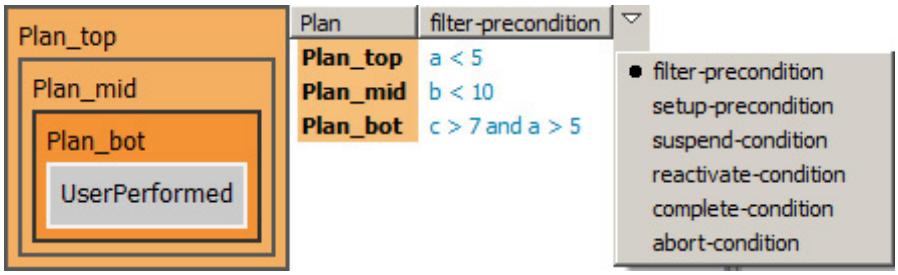


Figure 5.3: Condition view

Figure 5.3 shows the condition view in action. In the example the selection of *Plan_bot* triggers an update in the view. *Plan_bot* requires $a > 5$, but its ancestor *Plan_top* requires $a < 5$. This could constitute an error, or be indicative of a badly formalized guideline. The view helps both to better understand the semantics of the underlying process, and to uncover mistakes.

Breadcrumb

The breadcrumb trail is a is a commonly used navigation aid. It shows the ancestry of the currently selected plan. Inside the tree map itself, a selected plan could be spatially far removed from its parent's label – the user (if interested) would have to look for the parent (cognitive overhead) or even have to scroll/zoom (navigation overhead). In large tree maps the breadcrumb trail can help the user find his bearings by showing the ancestry of the focused plan.

Figure 5.4 shows a small example of what the trail looks like for a selected plan.

5.3 Interaction

PlanStrips allows basic interactions that enable users to model a new hierarchy, or to alter an existing one. The active interaction tool is selected using a *tool palette*, a common practice in graphical editors.

Selection

Plans are selected through point-and-click actions. Unselected plans appear in neutral pastel colors, their borders in a light neutral grey. When a plan is selected its color is intensified and

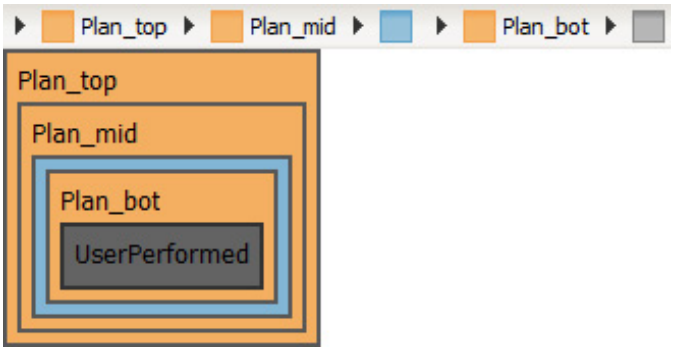


Figure 5.4: Breadcrumb of current selection

its border turns near black. A weaker version of this effect is also applied to the plans context (siblings, ancestors). This puts a fish-eye like focus on the selection (using shading).

The breadcrumb trail also provides limited interaction. A point-and-click on a node in the breadcrumb trail (see figure 5.4) results in a selection of the corresponding PlanStrips plan box. Having this option available can sometimes reduce navigation overhead.

Editing

Editing actions follow commonly used paradigms. Changes performed by the user can be undone and re-applied after undoing. Where possible (and feasible) the user is provided with instant feedback.

Deletion Nodes are deleted from the visualization by first selecting the unwanted node and then triggering the delete action. Another commonly used option is to place small icons ($\otimes$) directly into the visualization. These act as buttons, triggering a delete when clicked. This type of delete mechanism could be useful during early design stages, when deletion of nodes is performed more frequently.

Insertion New nodes are added by first selecting the desired tool from the tool palette (see blue area in figure 5.5), then clicking at the position where the selected node should be placed. Boxes that can be inserted are either containers (plan, sub-plan, cyclical plan) or leafs (e.g. user-performed plan).

Moving Existing nodes are moved using drag-and-drop operations. Similar to insertion, the drag action is first initiated on a node, then the cursor is moved to a drop position – and the node is dropped.

Feedback Visual feedback to user actions is used during placement (i.e. while inserting or dragging). Figure 5.5 shows what the feedback looks like when a node can be dropped at a given position, the cursor features a small plus icon. Figure 5.6 represents an invalid position (the cyclical plan can only contain one child).

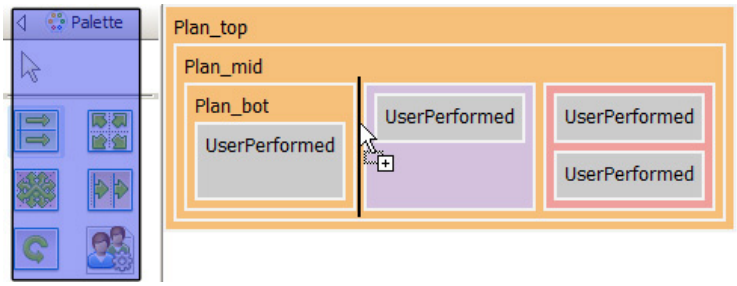


Figure 5.5: Palette and positive placement feedback

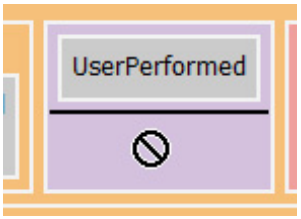


Figure 5.6: Negative placement feedback

Horizontal and vertical lines are used to preview where a node will be dropped. In figure 5.7 nodes would be added in the following way: (1) last child in *Plan_bot*, (2) middle child of *Plan_mid*, (3) first child in the red box (aligned horizontally), (4) middle child in the red box, (5) last child of *Plan_mid*, and (6) last child of *Plan_top* .

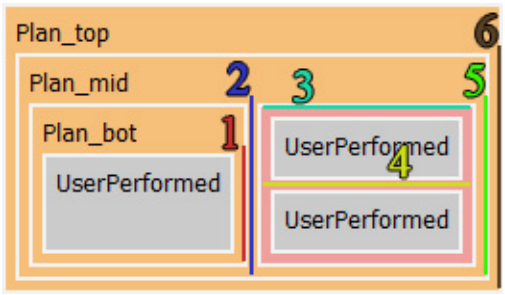


Figure 5.7: Placement preview lines

Content information Optimally content information visible in PlanStrips would be editable directly inside the visualization. Alternatively (or additionally) a properties UI element (e.g. dockable window) can be used.

5.4 Limitations and drawbacks

This section contains known issues and oddities inherent to the PlanStrips visualization.

Form

PlanStrips aligns child nodes horizontally and vertically, depending on sequential or parallel temporal ordering (see 5.2). If one type of ordering is predominant, the entire visualization can be stretched in length or height. The impact of this effect on the visualization is limited because in aspect ratio is not considered to be very important.

The stretching of nodes along their minor axis (see figure 5.1a) can cause very long or wide nodes whose empty body is still partially visible (display clutter), while the (useful) section

of the body containing content information is off-screen. However, such non-stretched nodes would be completely off-screen. Both these effects deliver unsatisfactory results.

Alignment

Aligning nodes horizontally and vertically can give the user an immediate impression about plan synchronization, though one might question the effective usefulness of this feature – which directly influences the shape of nodes. Plan synchronization is already expressed through box colors. And while horizontal ordering stands for sequential synchronization, vertical ordering is used for all other types of synchronization.
Still being able to immediately distinguish between sequential and other synchronizations is of some value. The alternative option would be to use a custom child-layout algorithm for better aesthetic appearance.

Unfolding

Strictly speaking, the hierarchy modeled by Asbru is not necessarily a tree-graph. The same plan can be used as a child multiple times. In the XML representation this is done using references. In PlanStrips such plans are simply duplicated as often as necessary. If such plans are small it is of little consequence, however big "redundant" plans can potentially take up a large amount of space by displaying the same information several times. In a node-link diagram this problem would not occur – the node (plan) would be rendered once with multiple edges.

Temporal detail

PlanStrips does not support the display of temporal information (other than plan synchronization). While temporal annotations on Asbru plans are optional, if consequently present this information can give the user further insight into the semantic process modeled in the guideline. This information would be useful to have. It could be augmented using an extra view like the temporal view of *CareVis* (see section 4.3).

5.5 Possible improvements

Smart scrolling

Whenever scrolling would cause the content-information of a node to disappear off the screen, but an empty stretched plan body is still visible, the content can be shifted to stay inside the view port. Figure 5.8 demonstrates this technique. The transparent blue area symbolizes the view port. Upon scrolling the content information (including children) of *plan AA* (blue plan on the left) is shifted to stay within view. Effectively this provides the user with global context information which would otherwise not be visible.

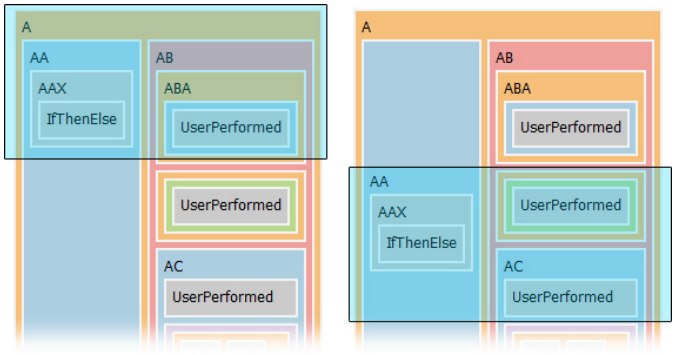


Figure 5.8: Smart scrolling

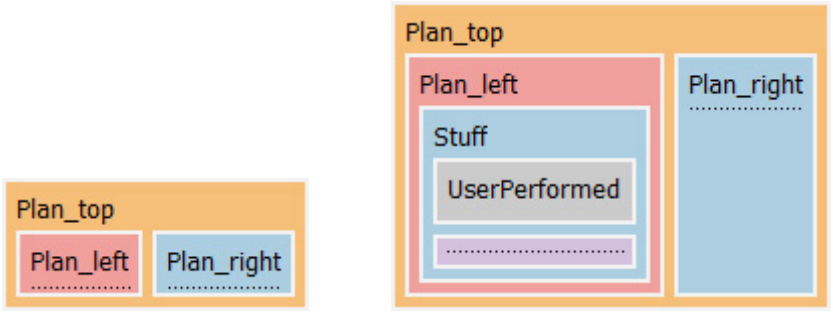
Filtering and drilling

The concept behind filtering and drilling in principle is very similar to *fisheye views* [40], also referred to as *continuous semantic zooming* [47]. Nodes in the visualization can be filtered by complete omission, or reduced size (i.e. the node is visible but very small – only showing important content information, and no children). Filtering can occur based on criteria such as depth, context, plan attributes etc.

The user can expand filtered nodes one *level* at a time, drilling down into the hierarchy, and collapse them when no longer needed [13, 51].

This allows the user to view the detail of a particular area of interest in the context of the complete hierarchy [51].

Figure 5.9 shows how this could be implemented in PlanStrips. Initially *Plan_left* and *Plan_righ* are filtered (indicated by dots). By drilling down on *Plan_left* its content is expanded by one level. Additionally the font size used in filtered nodes could be reduced.



(a) Depth filtered (b) Left side drilled

Figure 5.9: Filtering and drilling of nodes

Distortion

Distortion is a similar navigation technique to *drilling* (section 5.5). The key difference is that the technique features more degrees of level of detail than just filtered and non-filtered nodes.

Initial visibility of nodes, just like in the filter-drill approach, is determined by certain criteria. The less-visible nodes however are not simply filtered but still present in the visualization with a lower level of visibility. For example their content information and overall size could be reduced. This way a much larger portion of the overall hierarchical structure is visible, but with smaller (deeper) nodes only being rendered schematically.

Upon drilling down into a node it is enlarged (distorted in size). Enlarging the node also enlarges its children, effectively providing more *local detail* on the focused node – while keeping the context. This can either be done at the expense of the surrounding nodes (reducing their size while maintaining the total space used), or by enlarging the entire visualization.

Figure 5.10 illustrates the distortion approach to drilling. The user first initiates a focus action on the node (e.g. through clicking). The focus node is then enlarged and previously unseen (or strongly distorted) detail revealed. In the example surrounding nodes are reduced in size as the focus node is being enlarged. The distortion process should also incorporate some form of animation to present the user with a smooth transition. Instantaneous or too abrupt transition can disorient the user.

The concepts of drilling and distortion are more closely described in [47].

(a) focus (b) LOD++ (c) LOD++ (d) unfocus, LOD−− (e) LOD−−

Figure 5.10: Drilling using distortion (excerpt from [47])

Query

Queries can be used to allow the user to locate data of interest through search criteria. Queries are done in 2 phases. First the user performs the query, then the system displays the results. If a query component is implemented it should be decoupled from PlanStrips and serve as a generic search element – one that searches PlanStrips, but is not exclusively designed to do so. The search results which are possibly scattered throughout many views should be highlighted using brushing.

Effectively queries provide shortcuts reducing mechanic navigation overhead and (by highlighting results in the visualization) can reduce cognitive overhead.

Query action

Plausible search criteria used with PlanStrips could be (i) strings in plan name, description, intention, etc., (ii) variables names, qualitative or quantitative constants, (iii) conditions, and (iv) special types (subplans, if-then-else, etc.) . The interface component used for querying can be a simple text input. The search box would use a simple and verbose syntax to perform queries. A drop down component with smart suggestions is used to give the user an overview of search options, and features auto completion. This paradigm is used by the search box of *YouTrack* from *JetBrains*. Users unfamiliar with the search system use it as a keyword search. At the same time the system makes suggestions to the user to introduce him to more advanced searches – at a pace chosen by the user himself.

Feedback

There are different ways in which PlanStrips could present the user with feedback. Single matches can be highlighted, and automatically navigated to in the visualization. Multiple matches pose the question of how they should be presented to the user. A possible solution would be a result list, with selection (e.g. single click, double click) triggering an automatic navigation to the corresponding element in the visualization.

Syntax highlighting

The *condition view* (section 5.2) translates conditions, which are natively defined using nested XML elements, into more compact in-line strings (ie. $a > b \;\&\&\; completed(plan : AA)$). The presentation could be improved by using custom syntax highlighting for logic operators, variables, plan names, constant values, etc.

Highlight Asbru semantics

The first version of PlanStrips introduced here depicts (sub-)plans as boxes, synchronization through color and temporal order through alignment. However there are many language specific features and semantic information that is not visualized explicitly. *User-performed plans* are a special type of plan, indicating a procedure that has to be performed by an external agent. *Ask*-type nodes (also leafs) indicate that a parameter value has to be fetched (from an external source). *If-then-else* checks evaluate logic conditions to direct control flow. These are just a few examples.

Changes to the way PlanStrips displays nodes with special semantics can further support understanding. For example leaf nodes intrinsic to Asbru (*user-performed*, *ask*, etc.) could receive a custom node shape, be replaced by special icons, or otherwise visually differentiated. These nodes can offer interaction and presentation techniques that are unique to them.

Colorblind features

PlanStrips heavily depends on colors to show plan synchronization. These can be customized so that in theory a person suffering from red-green color blindness can switch out the “defective”

color. However it could be useful to offer an additional mean to show synchronization (e.g. via small icons in the corners of nodes). This could both supplement or be an alternative to the color approach.

Condition presence

Conditions are an important part of Asbru. They are explicitly shown in tools like AsbruView (section 4.1) and CareVis (section 4.3). While the details of conditions are visible in the *condition view* on-demand (when a plan is selected), it is not possible to see which conditions are present just from the PlanStrips visualization. Currently even when a plan is explicitly selected the user has to cycle through the drop-down list (figure 5.3).

Small condition indicator icons could be shown in the head section of the boxes (which also contains plan names). A mockup of this can be seen in figure 5.11. Unspecified conditions are displayed with an alpha value rendering them transparent, while specified conditions are displayed in full opacity and color. The metaphors/reasoning behind the icons proposed here (in order from left to right) are:

filter-precondition user-icon for identity, as filters apply to traits of the user which are not expected to change

setup-precondition screwdriver and wrench – tools are used to set up the right conditions for the plan

suspend-condition attention sign – a known but likely irregular condition has occurred and the plan has to be suspended, "watch out"

reactivate-condition re-cycle/cycle icon – execution of the plan resumes

complete-condition check-sign – everything is ok and the plan completes successfully

abort-condition x-sign – signifies an error and the plan has to be aborted

These icons, when clicked, would also be used as triggers to automatically jump into the condition view with the according condition being selected.

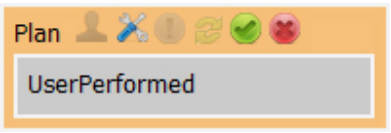
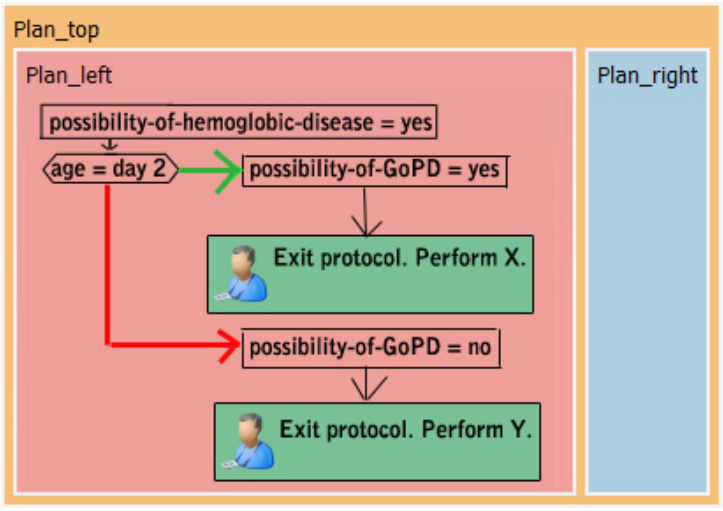


Figure 5.11: Condition display

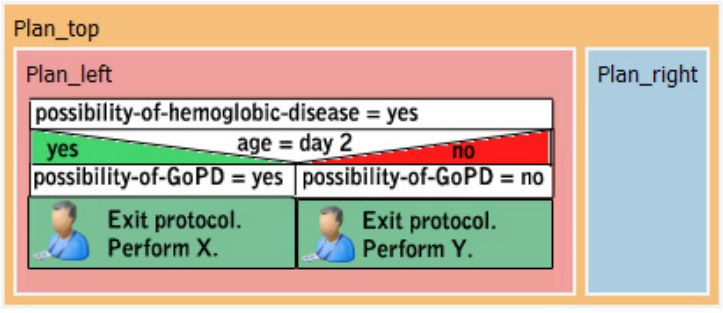
Semantic zooming, Flow chart representation

Some Asbru semantics such as if-then-else constructs are hard to express using PlanStrips' box based visualization. However the boxes conveniently provide a square display area which can be used to embed other visualizations. Figure 5.12 demonstrates how an *if-then-else* element with both branches leading to *user-performed* plans would look like as a flow-chart or structogram. The flow-chart has the advantage of resembling the, among physicians, well-known *clinical algorithm maps* [5], while the structogram is still easy to understand but offers a more dense representation.

Such constructs however can quickly use up a substantial amount of screen-space. It would therefor make sense to provide only an indicator in the initial view, where such embedded visualizations are available, and expanded them into full view upon clicking the indicator.



(a) Flow-chart



(b) Structogram

Figure 5.12: Embedded Asbru semantic visualization (user icon by [1])

5.6 Discussion

By using a tree-map PlanStrips achieves a high presentation density, this reduces navigational overhead (by reducing the amount of scrolling a user has to do in large guidelines). To some degree it also reduces cognitive overhead, by often allowing the user to keep plans that he is interested in, in his peripheral vision.

The location of a plans ancestors, children, and siblings always follows the same pattern. It is not necessary to track them down by following edges, as would be the case with a node-link diagram.

The metaphor of nested boxes is simple and very easy to understand. Dragging boxes around and creating new boxes to change the hierarchical structure should also be something that users are immediately familiar with, as this approach is largely analogous to moving around files and folders using any conventional graphical OS interface.

Showing temporal execution order by aligning plans along the x and y axes, allows users to distinguish between sequential and parallel-type plans. While this holds true for *sequential*, *parallel*, and *unordered* types of synchronization, a caveat arises with *any-order* type plans. Any-order plans are really executed sequentially, but in arbitrary order. The problem here is, that displaying any-order plans aligned along the x-axis would likely mislead the user into thinking, that they are executed sequentially from left to right. At the same time aligning them along the y-axis (as is currently the case) is arguably "wrong" – as they are not really parallel.

PlanStrips shows hierarchical structure. It shows task decomposition into smaller tasks, which allows users to understand the overall process modeled by a guideline (similar to the way a *work breakdown structure* does). It qualitatively displays when these sub-tasks are executed in relation to each other. This to some degree is sufficient for a user to understand many aspects of the process modeled in a guideline. Conveying understanding in this way is a valid approach, but it feels somewhat implicit – still on the level of abstraction that PlanStrips uses it seems sufficient. However more detailed processes, such as those described in if-then-else constructs etc. should probably be modeled in a flow-chart like fashion, as explained in section 5.5 (even though they could also be displayed using PlanStrips boxes).

CHAPTER 6

Evaluation

6.1 Method

The prototype evaluation was performed in form of a user study. Each session lasted up to an hour. The study included three physicians, two domain experts, and a software engineer. Because of the small sample size, putting the focus on a quantitative evaluation of user performance would not be meaningful enough, instead the focus laid on user feedback and observing user behavior. The tests were performed in the form of semi-structured interviews. The interview process was dynamic with specific questions/tasks being omitted/added based on the testers willingness to complete a given task, or time constraints.

Within reason users were given as little information as possible about the prototype. The main goal of the evaluation was not to test the functionality of the software or detect bugs, but to determine how quickly, and with how little schooling someone who had never worked with PlanStrips before could figure out how to use the prototype. Accordingly the tasks users had to perform were for the most part simple cognitive, and small hierarchy/process-editing tasks, which were meant to verify if the basic use cases of the software are easy enough to perform.

The questionnaire, and the questions referenced by the *Task#* column in section 6.3, can be found in appendix A.

Introduction

Since the evaluation was mainly based on observation and user feedback it was important to first establish the testers prior knowledge and experience. For this testers were presented with introductory questions to assess their proficiency in using a PC, prior knowledge of information visualizations, planning tools, clinical practice guidelines and possible familiarity with tools from the Asbru domain or other guideline authoring software.

Prototype Testing

In the second part of the test users were presented with a small example process modeling the baking of a pizza (in PlanStrips). The process was briefly explained, specifically the meaning of nested plans, and the different types of plan synchronization with the corresponding color coding. For the duration of the tasks the users could use a cheat-sheet which explained which colors represent which type of plan-synchronization. The introduction was very brief and lasted no more than 2 minutes. Beyond that nothing about the software was explained regardless of the users prior knowledge. The introduction was immediately followed by a series of simple tasks the users had to perform – first using PlanStrips, then with AsbruView. PlanStrips was the first program every time. The reason for doing this was to avoid potential learning effects, which might have arisen if AsbruView were to be used prior to PlanStrips. In total users had to perform 12 tasks per tool (24 tasks per user).

Tasks required testers to locate, describe, or edit features of an example process. Two equivalent processes were used. Testers either used *process A* with PlanStrips and *process B* with AsbruView or *process B* with PlanStrips and *process A* with AsbruView. Testers were free to skip tasks (whatever the reason). If a user had problems completing a task he was given hints, or upon failure eventually given the solution.

Feedback

In the final part testers were asked a series of questions which fit into the following basic categories i) how comfortably/easy was it for them to use the prototype, ii) how does the prototype (or aspects of it) compare to AsbruView or other software, iii) which features could be improved (and how), are missing, and which are notably well executed

6.2 Interviews, Participants

Interview 1

Frieder Ulm BSc. as of 2013 is enrolled in *Software Engineering & Internet Computing MSc.* at the *Vienna University of Technology*. Due to his studies he has experience in software development and is an expert computer user. He has a fair amount of expertise in UML modeling and has worked with other modeling software such as *Microsoft Project*, *Protégé*, *BPEL*, etc. He has adept experience of information visualizations which covers fundamental visualizations such as bar/pie-charts, trees, graphs, flow-charts, GANTT-charts, work breakdown structures etc., however has no prior knowledge regarding medical guidelines or guideline definition ontologies such as Asbru.

Interview 2

Dr. Popow

Univ. Prof. Dr. Christian Popow is a child psychiatrist of the *Medical University of Vienna* (Universitätsklinik für Kinder- und Jugendpsychiatrie, Ambulanz für Zwangskrankheiten) who specializes in obsessive-compulsive disorders. He is proficient in working with computers (having worked with tools like e.g. SAS for statistical data analysis). He previously participated in Asbru prototype evaluations and is thus familiar with the ontology. Further he has some knowledge of PROforma – a different ontology which, like Asbru, deals with knowledge representation, and computer-interpretable guideline development and execution.

He is familiar with several types of visualizations: generic ones like bar charts, pie charts, flow charts and several specialized visualizations such as clinical algorithm maps, star glyphs, and LifeLines. He was also involved in GenEdit [33] (a generic editor for questionnaires) and is listed as co-author in *Gravi++: Interactive Information Visualization of Highly Structured Temporal Data* [21]. Dr. Popow regards guidelines as absolutely essential in all possible areas of application (self-education, control in regard to own line of action, teaching students, etc.).

Dr. Ohmann

Mag. Dr. Susanne Ohmann works closely with Dr. Popow. She is a psychotherapist who specializes in eating disorders. She joined in on the interview with Dr. Popow and assisted during the practical-tasks portion of the study, as well as giving feedback during the post-interview questions.

Interview 3

Dr. Gabriele Grohs is the head of the intensive care department at the hospital *Krankenhaus Göttlicher Heiland*. Among other things she handles patient therapy, visits and management tasks. Being a novice PC user she works with standard programs for internet browsing, word processing etc. Her hospital's IT is based on SAP, which is set up to perform tasks like querying for lab work, X-ray reports, conciliar reports from other medical experts, patient information etc. She has no previous experience with planning tools or electronic guideline formats (such as Asbru). Her visualization knowledge covers the basics: bar charts, pie charts, clinical algorithm maps (flow-chart). She too regards guidelines as a very important.

Interview 4

Mag. Dr. Katharina Kaiser is a postdoctoral research fellow at the *Institute of Software Technology & Interactive Systems, Interactive Media Group, Vienna University of Technology*. From October 2008 to October 2010 she was an assistant professor at the *Center for Medical Statistics, Informatics, and Intelligent systems, Medical University of Vienna*. She received her M.Sc. and Ph.D. in information systems from the *Vienna University of Technology* in 2002 and 2005, respectively. Since November 2002 she's been involved in several national and international projects. Since November 2010 she has worked as a principal investigator in the BRIGID project, funded by the *Austrian Science Fund* (FWF) (Nov. 2010 - Oct. 2013) [24]. The BRIGID project deals with the development of a comprehensive solution for authoring CGPs in formal and informal (freetext) form by a distributed team of experts with complementary expertise [54]. Within the scope of the BRIGID project she is involved in natural language processing – the automatic translation of freetext guidelines into ontology-based formats.

Being involved the development of guideline authoring software, she is both an expert in computer science and guideline ontologies, primarily working with Asbru. As such she is also well versed in information visualization and process modeling software.

Interview 5

Dipl.-Ing. Dr. Wolfgang Aigner received his PhD in computer science from Vienna University of Technology in March 2006. From 2003-2006 he was scientific researcher at the project *Gaining New Medical Insights through Temporal Data Abstraction and Clinical Protocols* funded by the *Austrian Science Fund* (FWF). He has a MSc from Vienna University of Technology where he wrote his master thesis on "Interactive Visualization of Time-Oriented Treatment Plans and Patient Data" [3]. His research interests are in the fields of information visualization, visual analytics, human computer interaction, user centered design, and interaction design. As of 2013 he is working as an assistant professor at the *Institute of Software Technology & Interactive Systems* of the *Vienna University of Technology* [2]. He is an expert in computer science, information visualization, and as the co-author of *CareVis* (see section 4.3) also intimately familiar with the Asbru ontology.

6.3 Results, Discussion

General

The results across all interviews were mostly consistent. Regardless of prior knowledge and skill level users were making the same kinds of mistakes when working with the PlanStrips prototype, and generally also giving the same feedback.

Layout All users preferred PlanStrips to AsbruView on the basis of PlanStrips' tree-map layout being clearly arranged, while AsbruView's 3d icicle plot was seen as confusing and hard to navigate. As noted by Dr. Aigner however it is not immediately obvious which axis is the sequential (x) and which is the parallel (y) one.

Process representation Users in general liked the way PlanStrips represents process hierarchies. All but one user correctly answered the question about the execution order of two "cousin"-plans (same grand-parent), and the question which required them to describe the execution order of a plan and its 8 children with a nesting depth of up to 3. The one user who failed to answer these questions correctly was Dr. Grohs, who had no prior knowledge of Asbru (or other ontologies), no expertise with modeling software and described herself as a novice PC user.

Usability, Features Users had little problems with dragging, dropping and creating plans using the placement-feedback lines (figure 5.7). Hasty drop actions sometimes resulted in a plan being moved to the wrong position, this did not happen frequently. In the feedback section however everyone criticized the i) lack of keyboard shortcuts, ii) lack of a right-click context menu, and iii) (sometimes) lack of direct in-visualization edit functionality . Dr. Aigner suggested that upon creation of a new plan, in addition to being able to set the name and title, the user should also be allowed to specify the plan type (from a drop-down box) – in the prototype version used for testing plans were created without a type, which had to be set in a separate user action. Dr. Popow pointed out that the ability to link PlanStrips data values to their respective HTML-guideline source material would be useful.

Colors The color coding for plan-synchronization was not a success during the evaluation. While an experienced user may instantly recognize the colors, they lack "intuitive recognizability" (i.e., why is green parallel, orange sequential, etc.). Mostly everyone noted that the colors should at least be augmented with an option to display the plan-synchronization in some other way. This could be done by showing a legend. A better option would be to display optional recognizable icons in the left-upper corner of plan-boxes (which could be toggled off when the user becomes familiar enough with the color coding). Another problem with the colors was, that users would confuse the pre-set default colors with each other. This even happened to Dr. Kaiser who is intimately familiar with the color coding.

Conditions, Properties Users had a hard time finding the *condition view* and *properties view* (where plan-names, plan-types, etc. can be changed). By clicking on a condition icon inside the visualization the condition-view displays the corresponding condition in a custom one-line syntax. Most users had to be made aware of that fact, and even those who figured it out themselves took a relatively long time to do so. The same is true for the properties-view. This is probably due to the fact that these two elements (the views and their respective in-visualization elements) are spatially very far apart. Again an experienced user may not have a problem with this, however this program behavior is not beginner friendly. A resolution for this problem would be to provide a right-click context menu (which is what users typically looked for first).

The custom condition syntax was somewhat readable for both experienced an un-experienced users – but took some time and guesswork to puzzle out, thus not readable enough. A better way to display the content of conditions would be useful. Either the current syntax can be improved upon with tooltip-explanations, syntax highlighting etc. or maybe an entirely different approach could be used. Further the conditions are currently not editable (however this was outside the scope of the project).

The in-visualization condition icons were intuitively guessed correctly by users with computer experience, and using the tool-tips which appear when hovering the mouse over the icons correctly identified by the rest.

Availability on tablets To some degree users found that having software like PlanStrips available on a tablet would be useful. The tool is still somewhat specialized, and therefor limited in its functionality. Dr. Grohs said that, once powerful enough such software would be nice to have on a tablet for patient visits. As far as modeling guidelines on tablets, the testers generally agreed that it would be a nice-to-have, but many pointed that a touch-screen could cause potential problems with dropping and picking up plans at precise locations.

Task completion time

The following two tables contain a breakdown of how long users took to complete the individual tasks. In both tables PS stands for PlanStrips, and AV for AsbruView. These measurements should be taken with a grain of salt – as mentioned in section 6.1 users who had difficulty completing a task or asked for help were given hints and explained details about how each program works. The tasks were not performed with a stopwatch in hand.

Tasks (referred to by Task#) from the tables correspond to question numbers from the questionnaire (appendix A).

Table 6.1 shows how long users need to complete tasks on average. For the majority of tasks users where more proficient in using PlanStrips. One thing to consider here is that tasks were always performed in PlanStrips first. As a result of which new users established some degree of familiarity with Asbru guidelines before they continued to the AsbruView task section.

Table 6.2 shows how long each individual interview task took to complete. n/a means that the task was skipped or (less frequently) answered incorrectly and not counted as completed.

Table 6.1: Average (avg) time (mm:ss) over all interviews required for task completion in PlanStrips (PS) and AsbruView (AV), and performance difference (i.e. AV +00:30 means that on average the task took 30s longer to complete in AsbruView)

Task#	avg-PS	avg-AV	difference
20	00:12	00:41	AV +00:29
21	00:33	01:37	AV +01:04
22	00:13	00:19	AV +00:06
23	02:14	01:24	AV -00:50
24	00:35	01:46	AV +01:10
25	01:02	02:20	AV +01:18
26	00:11	01:36	AV +01:25
27	00:12	00:29	AV +00:16
28	00:52	00:35	PS +00:16
29	00:29	00:57	AV +00:28
30	00:20	00:21	AV +00:00
31	00:13	00:11	PS +00:02

Table 6.2: Break down of task completion times (mm:ss) per interview (section 6.2) per tool (i.e. interview#-PlanStrips and interview#-AsbruView)

Task#	1-PS	1-AV	2-PS	2-AV	3-PS	3-AV	4-PS	4-AV	5-PS	5-AV
20	00:17	n/a	00:16	01:50	00:05	00:08	00:14	00:27	00:08	00:19
21	00:28	n/a	00:37	02:20	01:00	n/a	00:28	00:53	00:13	01:40
22	00:08	n/a	00:19	00:38	00:16	00:15	00:15	00:10	00:07	00:14
23	01:47	n/a	02:40	n/a	04:37	n/a	01:05	01:29	01:05	01:20
24	00:25	n/a	01:20	01:30	00:43	00:20	00:16	04:47	00:14	00:27
25	00:33	n/a	00:36	02:43	n/a	n/a	02:10	01:13	00:49	03:05
26	00:10	n/a	00:03	03:10	00:11	n/a	00:10	00:40	00:24	01:00
27	00:05	n/a	00:08	00:20	00:32	n/a	00:11	00:15	00:06	00:52
28	01:11	n/a	01:18	00:50	00:55	00:20	00:20	00:30	00:36	00:41
29	00:27	n/a	01:00	01:15	00:25	n/a	00:06	00:40	n/a	n/a
30	00:09	n/a	00:27	00:40	00:18	00:20	00:26	00:12	00:24	00:14
31	00:08	n/a	00:12	00:20	00:33	00:08	00:10	00:13	00:04	00:03

CHAPTER 7

Conclusion

Chapter 1 gives a brief introduction into the domain of medical guidelines, the importance of translating them into a computer processable format, a succinct overview of the Asbru ontology, and also contains tidbits of background and history associated with this work.

Different types of fundamental visualization approaches, which can be used to achieve the functional requirements of PlanStrips (section 5.1), are presented in chapter 3 and their respective pros and cons are discussed. The work gives a relatively detailed overview of existing programs and tools for working with Asbru guidelines in chapter 4. Some of these have goals which strongly differ from those of PlanStrips, others like AsbruView more strongly coincide with them. For this reason AsbruView was chosen as a control program to compare with the PlanStrips prototype during the evaluation phase. The thesis discusses the functionality that these tools offer and presents thoughts on which of these tools' features would be useful to incorporate into PlanStrips, if the prototype were to be extended. Some of the visualizations and editors introduced by these tools, while unfit for integration with the PlanStrips visualization, would make great complementary visualizations to be used in conjunction with it (e.g., CareVis' *Temporal View*). This would require proper *brushing and linking* between views.

Following from the requirements and goals that were set for the PlanStrips prototype/visualization (section 5.1) the underlying visualization method chosen for the tool is a nesteed tree-map. Its characteristics, the customizations that the tool introduces to this basic visualization, and the resulting effects are described in section 5.2.

Usability, intuitiveness, ease of use, being the main goals of the prototype are evaluated in a user study, the process and reasoning behind the study are described in section 6.1. In short the findings of the study confirmed that PlanStrips is very intuitive to operate and can be used after very little introduction. A few technical shortcomings however, such as the lack of keyboard-shortcuts, were criticized by testers. This is discussed in more detail in section 6.3.

7.1 Future Work

PlanStrips works well for visualizing and modeling of hierarchical plan structure. There are still many improvements which could greatly benefit the existing use-cases the software covers, these are described in section 5.5. Further section 6.3 contains feedback given by users during the evaluation, much of which contains sensible ideas on how to further improve the existing Prototype.

The tool itself is based on the Eclipse RCP platform. New editors, that would complement the functionality that PlanStrips offers, could be added to give users the ability to view and edit other facets of a guideline, without having to switch between tools. Cues on which additional visualizations would complement PlanStrips are contained in the discussion portions of chapter 4.

It should be noted, that PlanStrips uses Eclipse RCP version 3 – which at the time of its creation was in the process of being transitioned to version 4. Version 4 of the Eclipse RCP platform contains major changes in the application model, and while it offers a backward compatibility layer to version 3, it is not complete and contains bugs (as does version 4 itself at the time of creating the prototype). The main reason why version 3 was used in the first place, is that many libraries available for version 3 have not yet been ported to version 4 (among them the *Graphical Editing Framework*, which PlanStrips makes use of). When extending the Prototype this should be taken into consideration.

APPENDIX A

Questionnaire

Fragebogen PlanStrips User-Study

Teilnehmer

1. **Beschreiben sie kurz ihre beruflichen Aufgaben.**
2. **Beschreiben Sie kurz ihre Erfahrung in der Medizin.**

Informatik

3. **Computerwissen** niedrig □—□—□—□—□ hoch
4. **Beschreiben Sie kurz ihre Erfahrung mit Computern ausserhalb der Arbeit.**
5. **Wie verwenden sie Computer in der Arbeit (Taetigkeiten, Software, Haeufigkeit, Betriebssysteme).**
6. **Wie verwenden sie Computer in der Arbeit (Taetigkeiten, Software, Haeufigkeit, Betriebssysteme).**
7. **Sind sie mit Planungs- / Modelierungssoftware vertraut (BPEL, UML, Microsoft Project, etc.)?**
8. **Kennen/verwenden sie Software die CPGs unterstuetzt (Asbru, PROforma, EON, GEM, etc.)? Wenn ja, welche Erfahrungen haben sie mit diesen Tools gemacht?**

Visualisierung

9. **Flow Charts** □ Unbekannt. □ Gehoert. □ Grundlegendes Verstaendnis.

10. GANTT Charts	□ Unbekannt.	□ Gehoert.	□ Grundlegendes Verstaendnis.
11. LifeLines	□ Unbekannt.	□ Gehoert.	□ Grundlegendes Verstaendnis.
12. Block Diagram	□ Unbekannt.	□ Gehoert.	□ Grundlegendes Verstaendnis.
13. Pie Chart	□ Unbekannt.	□ Gehoert.	□ Grundlegendes Verstaendnis.
14. Clinical Algorithm Map	□ Unbekannt.	□ Gehoert.	□ Grundlegendes Verstaendnis.

15. Andere: _____

Protokolle

16. Welche Relevanz haben Behandlungsprotokolle / Clinical Guidelines fuer sie?

17. Als wie wichtig/praktisch/nuetzlich erachten sie CPGs generell (Peer-Diskussion, Unterricht, Informiert bleiben, etc.)?

18. Welche Daten typischerweise benoetigt?

19. Sind diese Daten ausreichend gut zugaengig?

Tasks

20a. Wie oft kommt Plan *Salz* vor? -> 4

20b. Wie oft kommt Plan *Kohle* vor? -> 2

21a. Koennen *Kirche* und *Salz* gleichzeitig ausgefuehrt werden? -> J

21b. Koennen *Schluessel* und *Otter* gleichzeitig ausgefuehrt werden? -> J

22a. Wo sind die Plaene *Stuhl*, *Kepler*, *Mikrowelle*?

22b. Wo sind die Plaene *Schluessel*, *Karton*, *Ofen*?

23a. Beschreibe die Conditions in *Apfel*?

23b. Beschreibe die Conditions in *Birne*?

24a. Welche Plaene haben *suspend* aber fehlende *re-activate* conditions?

24b. Welche Plaene haben *suspend* aber fehlende *re-activate* conditions?

25a. Beschreibe was in Plan Mikrowelle passiert.

25b. Beschreibe was in Plan Ofen passiert.

26a. Verschiebe *Feder* zw. *Bruecke* u. *Pfad*.

26b. Verschiebe *Haken* zw. *Wolke* u. *Sonne*.

27a. Vertausche *Auto* mit *Widerstand*.

27b. Vertausche *Kapazitor* mit *Kutsche*.

28a. Aendere *Sueden* von *unordered* auf *parallel*.

28b. Aendere *Flugzeug* von *unordered* auf *parallel*.

29a. Kopiere *Sand* nach *Sueden* an letzte Stelle.

29b. Kopiere *Treibstoff* nach *Flugzeug* an letzte Stelle.

30a. Aendere den Namen von *Tisch* auf *Vase*.

30b. Aendere den Namen von *Tuer* auf *Fenster*.

31a. Loesche den *sequential* sub-plan der *Bison* und *Ratte* beinhaltet.

31b. Loesche den *sequential* sub-plan der *Plastik* und *Wolle* beinhaltet.

Endfragen

32. Wurden die Beispielprozesse verstaendlich genug dargestellt?

33. Wie sinnvoll / verstaendlich finden sie die vorgestellte Art prozesse darzustellen?

34. War PlanStrips leicht / intuitiv zu verwenden?

35. Gibt es positive oder negative Eindruecke die bei der verwendung von PlanStrips herausstehen?

36. Welche Funktionen fehlen der Software, bzw. haben besondere Wichtigkeit?

37. Gibt es etwas, dass sie anders machen wuerden?

38. Beschreiben sie ihre Eindruecke bezueglich der Verwendung von PlanStrips vs. AsbruView?

39. Gibt es konkrete Programme, Benutzer-Interfaces, Workflow-Paradigmen, etc. mit denen sie gut vertraut sind, deren Charakteristiken sie sich in PlanStrips vorstellen koennten?

40. Haben sie persoenliche Praeferenzen was Benutzung von PCs vs. Tablets angeht?

41. Legen sie Wert auf, oder sehen Potential darin, solche Software auf Tablets verwenden zu koennen?

42. Was sind ihre Gedanken bzgl. Steuerung (Maus, Keyboard, Touchscreen, Gesten, Kamera, etc.)?

43. Welche Erwartungen haben sie an derartige Software, welche Aspekte stehen im Mittelpunkt?

Bibliography

[1] User-performed icon for visualization mock-up. `http://www.gettyicons.com/free-icon/112/office-space-icon-set/free-user-icon-png/`.

[2] Wolfgang Aigner. Wolfgang aigner, information engineering group about page.

[3] Wolfgang Aigner. Wolfgang aigner, personal bio page.

[4] Wolfgang Aigner, Katharina Kaiser, and Silvia Miksch. *Visualization Techniques to Support Authoring, Execution, and Maintenance of Clinical Guidelines*, page 140–159. IOS Press, Health Technology and Informatics, 2008.

[5] Wolfgang Aigner and Silvia Miksch. Carevis: Integrated visualization of computerized protocols and temporal patient data. *Artificial intelligence in medicine*, 37(3):203–218, 2006.

[6] Wolfgang Aigner, Silvia Miksch, et al. Communicating the logic of a treatment plan formulated in asbru to domain experts. *Studies in health technology and informatics*, pages 1–15, 2004.

[7] Wolfgang Aigner, Silvia Miksch, Bettina Thurnher, and Stefan Biffl. Planninglines: Novel glyphs for representing temporal uncertainties and their evaluation. In *Information Visualisation, 2005. Proceedings. Ninth International Conference on*, pages 457–463. IEEE, 2005.

[8] Michael Balser, Christoph Duelli, and Wolfgang Reif. Formal semantics of asbru-an overview. In *Proceedings of the International Conference on Integrated Design and Process Technology*, 2002.

[9] Michael Balzer, Oliver Deussen, and Claus Lewerentz. Voronoi treemaps for the visualization of software metrics. In *Proceedings of the 2005 ACM symposium on Software visualization*, pages 165–172. ACM, 2005.

[10] Lawrence D Bergman, Bernice E Rogowitz, and Lloyd A Treinish. A rule-based tool for assisting colormap selection. In *Proceedings of the 6th conference on Visualization'95*, page 118. IEEE Computer Society, 1995.

[11] Eta S Berner. *Clinical Decision Support Systems*. Springer Science+ Business Media, LLC, 2007.

[12] Stefan Berner, Stefan Joos, Martin Glinz, and Martin Arnold. A visualization concept for hierarchical object models. In *Automated Software Engineering, 1998. Proceedings. 13th IEEE International Conference on*, pages 225–228. IEEE, 1998.

[13] Renaud Blanch and Eric Lecolinet. Browsing zoomable treemaps: Structure-aware multi-scale navigation techniques. *Visualization and Computer Graphics, IEEE Transactions on*, 13(6):1248–1253, 2007.

[14] David Boaz and Yuval Shahar. Idan: A distributed temporal-abstraction mediator for medical databases. In *Artificial Intelligence in Medicine*, pages 21–30. Springer, 2003.

[15] Cynthia A Brewer. Color use guidelines for mapping and visualization. *Visualization in modern cartography*, 2:123–148, 1994.

[16] Institute of Medicine Committee on Quality of Health Care in America. *Crossing the Quality Chasm: A New Health System for the 21st Century*, page 151. The National Academies Press, 2001.

[17] Peter E Friedland and Yumi Iwasaki. The concept and implementation of skeletal plans. *Journal of automated reasoning*, 1(2):161–208, 1985.

[18] Avner Hatsek, Yuval Shahar, Meirav Taieb-Maimon, Erez Shalom, Denis Klimov, and Eitan Lunenfeld. A scalable architecture for incremental specification and maintenance of procedural and declarative clinical decision-support knowledge. *The Open Medical Informatics Journal*, 4:255, 2010.

[19] Avner Hatsek, Ohad Young, Erez Shalom, and Yuval Shahar. Demonstration of degel: A clinical-guidelines library and automated guideline-support tools. *AI Techniques in Healthcare: Evidence-based Guidelines and Protocols*, page 22, 2006.

[20] Jeffrey Heer, Michael Bostock, and Vadim Ogievetsky. A tour through the visualization zoo. *Communications of the ACM*, 53(6):59–67, 2010.

[21] Klaus Hinum, Silvia Miksch, Wolfgang Aigner, Susanne Ohmann, Christian Popow, Margit Pohl, and Markus Rester. Gravi++: Interactive information visualization to explore highly structured temporal data. *Journal of Universal Computer Science*, 11(11):1792–1805, 2005.

[22] Benedikt Huber. *Asbruview 2.0 User Guide*, 2005.

[23] Brian Johnson and Ben Shneiderman. Tree-maps: A space-filling approach to the visualization of hierarchical information structures. In *Visualization, 1991. Visualization'91, Proceedings., IEEE Conference on*, pages 284–291. IEEE, 1991.

[24] Katharina Kaiser. Katharina kaiser, personal bio page, 2013.

[25] Robert Kosara and Silvia Miksch. Visualization techniques for time-oriented, skeletal plans in medical therapy planning. *Artificial Intelligence in Medicine*, pages 291–300, 1999.

[26] Robert Kosara and Silvia Miksch. Metaphors of movement: a visualization and user interface for time-oriented, skeletal plans. *Artificial Intelligence in Medicine*, 22(2):111–131, 2001.

[27] Robert Kosara, Silvia Miksch, Andreas Seyfang, and Peter Votruba. Tools for acquiring clinical guidelines in asbru. In *Proceedings of the Sixth World Conference on Integrate Design and Process Technology (IDPT'02)*, 2002.

[28] Joseph B Kruskal and James M Landwehr. Icicle plots: Better displays for hierarchical clustering. *The American Statistician*, 37(2):162–168, 1983.

[29] Albert A Larkin. Misleading graphics: Can decision makers be affected by their use. Technical report, DTIC Document, 1990.

[30] Su-Youl Mun and Seok-Wun Ha. An efficient visualization of hierarchical structures. In *Enterprise networking and Computing in Healthcare Industry, 2005. HEALTHCOM 2005. Proceedings of 7th International Workshop on*, pages 419–425. IEEE, 2005.

[31] Quang Vinh Nguyen and Mao Lin Huang. A space-optimized tree visualization. In *Information Visualization, 2002. INFOVIS 2002. IEEE Symposium on*, pages 85–92. IEEE, 2002.

[32] American Academy of Pediatrics. Practice parameter: Management of hyperbilirubinemia in the healthy term newborn. *Pediatrics*, 94(4):558–565, 1994.

[33] Martina Osztovits. Genedit–a generic editor and tools for questionnaires.

[34] Mor Peleg, Samson Tu, Jonathan Bury, Paolo Ciccarese, John Fox, Robert A Greenes, Richard Hall, Peter D Johnson, Neill Jones, Anand Kumar, et al. Comparing computer-interpretable guideline models: a case-study approach. *Journal of the American Medical Informatics Association*, 10(1):52–68, 2003.

[35] Catherine Plaisant, Brett Milash, Anne Rose, Seth Widoff, and Ben Shneiderman. Lifelines: visualizing personal histories. In *Proceedings of the SIGCHI conference on Human factors in computing systems: common ground*, pages 221–227. ACM, 1996.

[36] Neil Ramsay and Terrance Wampler. Visualization and interaction with financial data using sunburst visualization, September 20 2012. US Patent 20,120,240,064.

[37] Edward M. Reingold and John S. Tilford. Tidier drawings of trees. *Software Engineering, IEEE Transactions on*, (2):223–228, 1981.

[38] G.E.H.M. Rutten, S. Verhoeven, R.J. Heine, W.J.C. de Grauw, P.V.M. Cromme, K. Reenders, E. van Ballegooie, and T. Wiersma. Nhg-standaard diabetes mellitus type 2. *Huisarts en Wetenschap*, 42:67–84, 1999.

[39] David L Sackett, William Rosenberg, JA Gray, R Brian Haynes, and W Scott Richardson. Evidence based medicine: what it is and what it isn't. *Bmj*, 312(7023):71–72, 1996.

[40] Doug Schaffer, Zhengping Zuo, Saul Greenberg, Lyn Bartram, John Dill, Shelli Dubs, and Mark Roseman. Navigating hierarchically clustered networks through fisheye and full-zoom methods. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 3(2):162–188, 1996.

[41] H-J Schulz, Steffen Hadlak, and Heidrun Schumann. The design space of implicit hierarchy visualization: A survey. *Visualization and Computer Graphics, IEEE Transactions on*, 17(4):393–411, 2011.

[42] Andreas Seyfang, Katharina Kaiser, Theresia Gschwandtner, and Silvia Miksch. Visualizing complex process hierarchies during the modeling process. In Ross Brown, Simone Kriglstein, and Stefanie Rinderle-Ma, editors, *First International Workshop on Theory and Applications of Process Visualization (TAProViz'12)*, Lecture Notes in Business Information Processing (LNBIP). Springer, Springer, 2012.

[43] Yuval Shahar, Silvia Miksch, and Peter Johnson. The asgaard project: a task-specific framework for the application and critiquing of time-oriented clinical guidelines. *Artificial intelligence in medicine*, 14(1):29–51, 1998.

[44] Yuval Shahar, Ohad Young, Erez Shalom, Alon Mayaffit, Robert Moskovitch, Alon Hessing, and Maya Galperin. Degel: A hybrid, multiple-ontology framework for specification and retrieval of clinical guidelines. In *Artificial Intelligence in Medicine*, pages 122–131. Springer, 2003.

[45] Erez Shalom and Yuval Shahar. A graphical framework for specification of clinical guidelines at multiple representation levels. In *AMIA Annual Symposium Proceedings*, volume 2005, page 679. American Medical Informatics Association, 2005.

[46] Erez Shalom, Yuval Shahar, Meirav Taieb-Maimon, Guy Bar, Avi Yarkoni, Ohad Young, Susana B Martins, Laszlo Vaszar, Mary K Goldstein, Yair Liel, et al. A quantitative assessment of a methodology for collaborative specification and evaluation of clinical guidelines. *Journal of biomedical informatics*, 41(6):889–903, 2008.

[47] Kang Shi, Pourang Irani, and Ben Li. An evaluation of content browsing techniques for hierarchical space-filling visualizations. In *Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on*, pages 81–88. IEEE, 2005.

[48] Ben Shneiderman. Treemaps for space-constrained visualization of hierarchies, 1998.

[49] John Stasko, Richard Catrambone, Mark Guzdial, and Kevin McDonald. An evaluation of space-filling information visualizations for depicting hierarchical structures. *International Journal of Human-Computer Studies*, 53(5):663–694, 2000.

[50] John Stasko and Eugene Zhang. Focus+ context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations. In *Information Visualization, 2000. InfoVis 2000. IEEE Symposium on*, pages 57–65. IEEE, 2000.

[51] David Turo and Brian Johnson. Improving the visualization of hierarchies with treemaps: Design issues and experimentation. In *Visualization, 1992. Visualization'92, Proceedings., IEEE Conference on*, pages 124–131. IEEE, 1992.

[52] Peter Votruba, Silvia Miksch, and Robert Kosara. Linking clinical guidelines with formal representations. *Artificial Intelligence in Medicine*, pages 152–157, 2003.

[53] George Weisz, Alberto Cambrosio, Peter Keating, Loes Knaapen, Thomas Schlich, and Virginie J Tournay. The emergence of clinical practice guidelines. *Milbank Quarterly*, 85(4):691–727, 2007.

[54] www.ims.tuwien.ac.at. Support for authoring and transformation of clinical guidelines and protocols, 2010.

[55] Shengdong Zhao, Michael J McGuffin, and Mark H Chignell. Elastic hierarchies: Combining treemaps and node-link diagrams. In *Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on*, pages 57–64. IEEE, 2005.