DIPLOMARBEIT

Metaphors of Movement — A User Interface for Manipulating Time-Oriented, Skeletal Plans

ausgeführt am Institut für Softwaretechnik
der Technischen Universität Wien

unter Anleitung von
o. Univ. Prof. Dipl.-Ing. Dr. A Min Tjoa
und
Univ.-Ass. Mag. Dr. Silvia Miksch
als verantwortlich mitwirkender Universitätsassistentin

durch

Robert Kosara
Radetzkystraße 21/7
1030 Wien

Wien, am 6. Mai 1999

Abstract

This thesis introduces a user interface that supports the understanding and manipulation of time-oriented, skeletal plans.

This user interface is called AsbruView, and is based on the plan representation language Asbru, which is used for medical therapy planning. Clinical protocols are seen in Asbru as time-oriented, skeletal plans.

AsbruView utilizes Metaphors of running tracks and traffic control to communicate important concepts and uses glyphs to depict the complex time annotations used in Asbru. Two different views show different aspects of the same set of plans: One shows the topology of plans and which parts have been defined, the other captures the temporal dimension and the structure of plans.

We present a number of existing visualization approaches to different problems that we faced and discuss their usefulness for our purpose. We also show why we did not use a knowledge acquisition tool for editing Asbru plans.

We have evaluated AsbruView with six domain experts (physicians), who judged it as usable and easy to understand. The findings of that evaluation are presented and discussed.

Zusammenfassung

Diese Arbeit beschreibt ein User Interface zur Arbeit mit zeitorientierten, skeletalen Plänen.

Dieses Interface heißt AsbruView, und dient als Frontend für die Planrepräsentationssprache Asbru, die zum Einsatz in der medizinischen Therapieplanung entwickelt wurde. Klinische Protokolle werden in AsbruView als zeitorientierte, skeletale Pläne verstanden.

AsbruView verwendet Metaphern aus der Leichtathletik und dem Straßenverkehr, um abstrakte Konzepte leichter verständlich zu machen. Glyphs werden eingesetzt, um die komplexen Zeitannotationen anschaulich zu machen. Zwei Ansichten derselben Daten existieren: Eine stellt die Topologie der Pläne dar, und welche Komponenten eines Plans definiert sind. Die andere Ansicht ermöglicht die präzise Definition der zeitlichen Dimension der Pläne, und stellt deren Struktur dar.

Eine Reihe existierender Visualisierungen für Konzepte, die auch in Asbru zu finden sind, und die teilweise in AsbruView Verwendung finden, wird kurz präsentiert.

Der AsbruView-Prototyp wurde mit sechs ÄrztInnen evaluiert, deren Urteil über die Verständlichkeit und Verwendbarkeit des Systems durchaus positiv ausfiel. Die Ergebnisse dieser Evaluation werden kurz vorgestellt.

Contents

List of Figures

Acknowledgments

First of all, I wish to thank my supervisor Silvia Miksch who has shown great patience and equally great trust in my work. This thesis would not have been possible without her.

I also wish to thank the physicians who took the time to test the system and who gave me valuable input for the further development of AsbruView (in alphabetical order, and using their correct German titles): Dr. Shahram Adel, Dr. Sophie Brandstetter, Dr. Maria Dobner, Dr. Gerhard Miksch, Primarius Dr. Franz Paky, and Ao. Prof. Dr. Christian Popow.

My colleagues in the Asgaard team, Klaus Hammermüller, Andreas Seyfang and Georg Duftschmid also deserve thanks for their contributions and criticism of my early ideas, and for providing a stimulating and challenging environment to develop ideas in.

Our medical collaborators Christian Popow and Franz Paky provided us with many ideas and helped us see beyond our (rather limited) idea of how physicians work and what they need. Werner Horn also supplied us with a lot of information from his vast experience.

Niki Sahling, who provided the hardware used in some of the tests, also deserves thanks for his help and patience (about half of the test appointments were canceled or postponed).

Thanks to Leonore Neuwirth for her help and for being there for me.

Last, but definitely not least, my parents deserve my thanks and respect for their support and love over many years.

This thesis was supported by "Fonds zur Förderung der wissenschaftlichen Forschung" (Austrian Science Fund), grant P12797-INF.

Chapter 1

Introduction

Planning is the substitution
of coincidence by error.
(Source unknown)

In this chapter, we introduce the application domain shortly, give an overview of the contents of this thesis, and present the conventions used in this thesis (including a rationale for using the pronoun *we* in a Master's thesis).

1.1 Motivation

Physicians and other medical staff usually do not need to invent treatment plans anew for every patient, but can fall back on predefined clinical protocols, or treatment plans. Such protocols make knowledge collected by many individuals through experience or clinical studies available to others. They also make documentation easier and make the comparison between different treatments possible.

Clinical protocols are currently represented by means that fail to cover the immense complexity of such treatment plans[1], do not properly capture the temporal dimension and do not allow for automatic verification. There are other plan-representation languages than Asbru (which will be describe briefly in this thesis), and some even provide means of verification, but none of them provides the high complexity needed in this domain.

Asbru is a plan-representation language that does not have these shortcomings. It is, however, a language that is hard to understand and use for somebody without a computer science background.

This is why we set out to create a graphical user interface that would make Asbru more accessible.

Currently, a subset of Asbru's concepts is included in this user interface, and a subset of these interface concepts has been implemented in a prototype. Concepts that are described here but were not implemented, will be clearly identified in order to avoid confusion about the real state of the work, and the area covered.

1.2 Overview of this Thesis

This thesis consists of three parts. In the first part, the problem is introduced. We briefly describe Asbru's main concepts, and give a short example of a plan in both natural language and Asbru. We briefly discuss the shortcomings of existing plan representations, and demonstrate the inappropriateness of knowledge acquisition tools for our purpose. We then present the state of the

[1] Throughout this thesis, the expressions *protocol*, *treatment plan* and *plan* will be used interchangeably.

art in visualization and show which approaches can or can not be used for visualizing Asbru. A chapter on user interaction requirements rounds off the first part.

In the second part, our solution, called AsbruView, is presented. We discuss the ideas behind the representation and the user interface, and briefly describe its implementation in a prototype.

In the third part, the results of an evaluation of that prototype are given, based on tests we performed with physicians.

1.3 Conventions

The work presented here was done in the context of a project, a team. Even though most of the work is mine, some ideas came from, or were influenced by, my supervisor and my colleagues. Most scientific work today is a team effort, and it makes little sense to pretend this thesis was done in complete isolation. This is why I chose to use the pronoun *we* rather than *I* and *our* rather than *my*. I do intend to convince you that my work is worthy of a Master's degree — and that most of it was done by me —, but it was not I alone who did it.

I shall come back to using *I* in the conclusion, where I will discuss some ideas about the perfect user interface. There is nobody else to blame for these ideas than I.

Another convention is which gender to use when speaking of abstract persons ('the user', 'the patient', etc.). [Dupré, 1998] writes:

> Certain people simply use female pronouns for all their characters, assuming that there cannot be any harm in redressing centuries of all-male pronouns holding sway, and that affirmative action is appropriate.

As my humble contribution to the balance of pronouns, and as a sign of respect towards my supervisor, I shall treat all the persons in this thesis as female (except for the participants in the evaluation, which are real persons and therefore deserve to be identified correctly).

Part I

Problem Analysis

Chapter 2

The Language Asbru

The letters are Elvish, of an ancient mode,
but the language is that of Mordor, which I will not utter here.
(J. R. R. Tolkien, *The Lord of The Rings*)

This chapter gives an introduction to the basic concepts of Asbru, describes its parts, and gives a short example of a treatment plan in natural language and in Asbru.

Asbru is the language used for representing treatment plans in the Asgaard/Asbru Project[1]. This project is described in [Miksch et al., 1998] as follows:

> The aim of the Asgaard/Asbru project [...] is to design a planner based on time-oriented, skeletal plans. The planner will support the design and the execution of skeletal plans by a human executing agent other than the original plan designer.

Asbru[2] is a language with a LISP-like syntax for representing time-oriented, skeletal plans [Friedland and Iwasaki, 1985] (i.e., plans that do not contain data of a particular patient, but that specify a general procedure; what parts of the plan are to be performed and how is decided as soon as the data of a patient is known). It was developed by Silvia Miksch together with Yuval Shahar and Peter Johnson [Miksch et al., 1997a, Miksch et al., 1997b, Shahar et al., 1998].

The BNF definition of Asbru consists of over 100 non-terminal symbols, and so is huge compared to most programming languages. But because it is not a general-purpose language, it includes a lot of task-specific parts that would otherwise need to be expressed by means of a lower-level language.

A short introduction to the main concepts of Asbru is given here, because these features must be captured by the user interface called AsbruView, that is presented in the second part.

2.1 Basic Concepts

These are the basic concepts that Asbru is made up of. An in-depth description of all the fine points of Asbru is beyond the scope of this thesis (see [Miksch et al., 1997b], for example).

2.1.1 Hierarchical Decomposition

A plan consists of sub-plans that give its contents in greater detail. This is called decomposition. Thus, one can define a plan once at a more abstract level, and then refine it.

[1] http://www.ifs.tuwien.ac.at/asgaard/

[2] Many of the names in the Asgaard project are based on names from Norse mythology. Asbru (also known as Bifrost), for example, is the rainbow bridge that leads to Asgaard, the home of the gods.

A plan that is not further decomposed is called an action. An action usually is a sufficiently small part that can be performed by medical staff or by a machine without the need for further information.

The sub-plans of a plan are said to be at a lower level than the containing plan. This is merely a convention and does not say anything about the plans, because the same plan may be used at different levels in different plans (and even in the same plan, see *Reuse* below).

Depending on the type of plan, different properties are propagated up or down the hierarchy levels. For example, as soon as all the sub-plans of a plan have completed successfully, the containing plan has completed successfully as well.

2.1.2 Reuse

Once defined, a plan can be used as a building block in any number of other plans. The plan is not copied, though, but rather is reused. This means that any changes made to it automatically affect all the plans using it. Thus, other plans can benefit from improvements in one of the plans they use.

2.1.3 Time Annotations

Any plan can be restricted in terms of its earliest or latest start, its minimum and maximum duration, and its earliest and latest end [Rit, 1986]. Conditions may be defined not only by means of values, but also by means of durations when values must be below or above a threshold.

This is done in Asbru by using time annotations. They consist of seven parts:

Reference Point. This is the point that all the other points in time are defined relative to. It can be an abstract point in time (e.g., *conception*).

Earliest Starting Shift (ESS). The smallest offset from the reference point when the action can take place.

Latest Starting Shift (LSS). The latest point in time when the action must start.

Earliest Finishing Shift (EFS). The earliest point in time when the action can end.

Latest Finishing Shift (LFS). The greatest offset from the reference point when the action must end.

Minimum Duration (MinDu). The minimum amount of time the action or condition must last. This is not necessarily identical with the interval between LSS and EFS. It is bounded, however, by this difference (it can not be shorter) and the maximum duration.

Maximum Duration (MaxDu). The maximum duration that the condition or action may last. It is bounded by the difference between LFS and ESS, and the minimum duration.

Each part may or may not be defined in a time annotation. This is necessary, because during the design phase, some parts may not be known, or they may not be of interest (like the start and end times, when only the duration is important).

In Asbru, a time annotation is written like this: [[ESS, LSS], [EFS, LFS], [MinDu, MaxDu], Reference]. Undefined shifts and durations are denoted by an underscore (‘_’). There are two special reference points: `*now*`, which stands for the current point in time, and `*self*`, for the start of the activation of the current plan. For an example, see Figure 2.3.

2.2 Plan Body

There are five plan types (the names here are based on keywords in Asbru, but were substituted by names that are easier to understand where possible. Names of plan types that are based on

keywords are written in italics.) that will be described here briefly. For an overview, see Table 2.1.

2.2.1 Sequential Plan (*Do-All-Sequentially*)

This kind of plan simply consists of a number of plans that are to be executed one after the other. All plans must be executed, and the order in which they must be performed is predefined (i.e. already known when the plan is authored). When the last sub-plan is completed successfully, the whole plan is completed, too.

This type of plan is mainly used for dividing a treatment plan into phases, when each phase has its unique characteristics and it is unlikely that a patient will return to a previous phase.

2.2.2 *All-Any-Order* Plan

Often, the order of execution of a set of plans cannot be determined beforehand, but depends on whether certain conditions are fulfilled. Such a set of plans can be put into an *all-any-order* plan, where the plan to be performed first, second etc. is only determined when the previous plan has been completed.

Additionally, if one plan in such a set is suspended (see section 2.5), another of the same set can be performed, to treat a serious condition, for example, or to use more aggressive treatment if the state of the patient gets worse.

An *all-any-order* plan is only completed successfully if all plans have been completed.

2.2.3 *Some-Any-Order* Plan

This is a variant of the *all-any-order* plan, where only some of the sub-plans have to be performed. These sub-plans are part of the *continuation condition*. Sub-plans that do not have to be performed are called optional plans in this thesis (there is no special name for them in Asbru).

2.2.4 Parallel Plan (*Do-All-Together*)

As the name suggests, a set of plans are to be executed in parallel (at the same time). These plans must start together, but need not (and usually will not) end at the same time. All of the plans must complete successfully for the containing plan to succeed.

2.2.5 *Some-Together* Plan

If not all sub-plans of a parallel plan must be performed, it is called a *some-together* plan. In that case, plans that are not part of the *continuation condition* (see *some-any-order* plans) do not influence the success of the containing plan if they fail or cannot be started.

2.2.6 Cyclical Plan

Many tasks in a treatment plan are repetitive: a patient is treated until her state gets better, a blood glucose test is performed before every meal, etc. In such a case, a cyclical plan can be used to represent this kind of action. A cyclical plan can be restricted in the maximum number of retries, the minimum and maximum delay between retries, and of course by conditions outside of the plan, like the value of an observed variable.

A cyclical plan can only have one sub-plan. This is no restriction, because that single sub-plan can contain any number of sub-plans.

	All plans must complete to continue	**Some plans must complete to continue**
Start together	Parallel Plans (no continuation-condition, all plans must complete)	*Some-Together* Plans (continuation-condition specified as subset of plans)
Execute in any order	*All-Any-Order* Plans (no continuation-condition, all plans must complete)	*Some-Any-Order* Plans (continuation-condition specified as subset of plans)
Execute in total order (sequence)	Sequential Plans (no continuation-condition, all plans must complete)	

Table 2.1: Plan Types in Asbru.

2.3 Preferences

Preferences are plan parameters that influence the behavior and applicability of a plan. Among these parameters are the strategy (e.g., aggressive or normal), resource specifications (i.e., which resources are needed or must not be used), etc.

2.4 Intentions

Intentions are high-level goals that are to be achieved or maintained by the plan, or avoided during its application. This allows for critiquing, but also for substitution of sub-plans.

There are two reasons for substitution: First, a physician may choose to do a certain part of a plan differently (personal taste, experience, etc.). As long as the intentions of both methods are the same, the overall plan can still be continued. This is not the case with other protocol representations that only describe actions but do not contain more abstract information about them (authors of protocols often try to avoid this deficiency by using more than just one representation, e.g., they add text to flow-charts — but this only leads to the information to become more disorganized and less lucid because of this mix of representations).

Second, a certain treatment may prove to be better than the one contained in the plan. If both methods are equal in terms of their conditions, the protocol designer may choose to use the new method, without having to change other parts of the plan.

2.5 Conditions

Which plans can be applied, and when a plan is completed (or has to be aborted, etc.), can only be decided by looking at conditions that describe the state of the patient that the plan was designed for. These conditions are part of the plans themselves in Asbru, other than in many other representations (e.g., flow-charts), where the decision is made outside of the plan.

There are six types of conditions and one token that control the transition of a plan between different states (see Figure 2.1). These states are similar to state transitions of tasks in operating systems, and so will not be explained here in detail (see [Miksch et al., 1997a] for a detailed explanation of the different states and state transitions).

An alternative view of Figure 2.1 is the flow-chart in Figure 2.2 that describes the flow of control through a plan written in Asbru by means of the conditions defined in it.

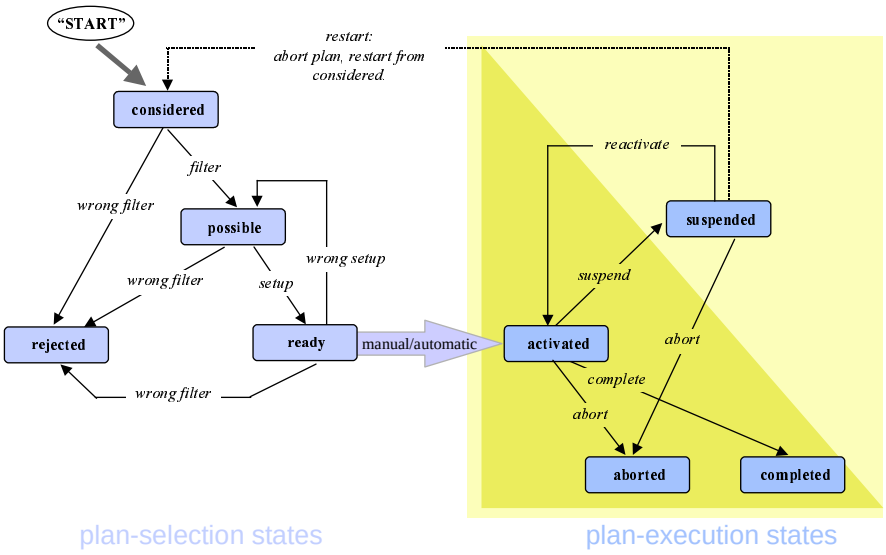


Figure 2.1: State diagram for Asbru plans (from [Miksch et al., 1997b]).

2.5.1 Filter Precondition

In order for a plan to be applicable at all, this condition must be satisfied. This condition cannot, however, be achieved — at least not easily. An example of such a condition would be to require that the patients be female for a plan for treating diabetes during pregnancy[3].

2.5.2 Setup Precondition

This condition must also be met in order for the plan to be applicable. In contrast to the filter precondition, though, this precondition can be achieved. Such a condition might be that a parameter must be below or above a certain value (usually in addition to a weaker constraint for the same parameter in the filter precondition).

When selecting a plan for treating a patient, there usually are several alternatives. Plans whose setup preconditions are met will more likely be used. But in case a plan with an unfulfilled setup-precondition is better in other respects, it can still be used with some additional work to satisfy this condition.

2.5.3 Suspend Condition

This condition specifies when the plan has to be suspended, for example when there is a critical situation that does not lead to the plan being aborted, but some action has to be taken anyway (e.g., the blood oxygen level has suddenly fallen). If a plan is part of an *any-order* plan, another of the set of sub-plans can now be performed, until the patient is stabilized or the whole plan needs to be aborted.

If the plan is not aborted, it can be continued as soon as the following condition is met.

[3]The number of decisions made by medical staff, and the amount of knowledge used in every-day treatment, is tremendous. People involved in such work never notice this until a bunch of computer scientists come along to put their knowledge into a form that can be understood by a machine.

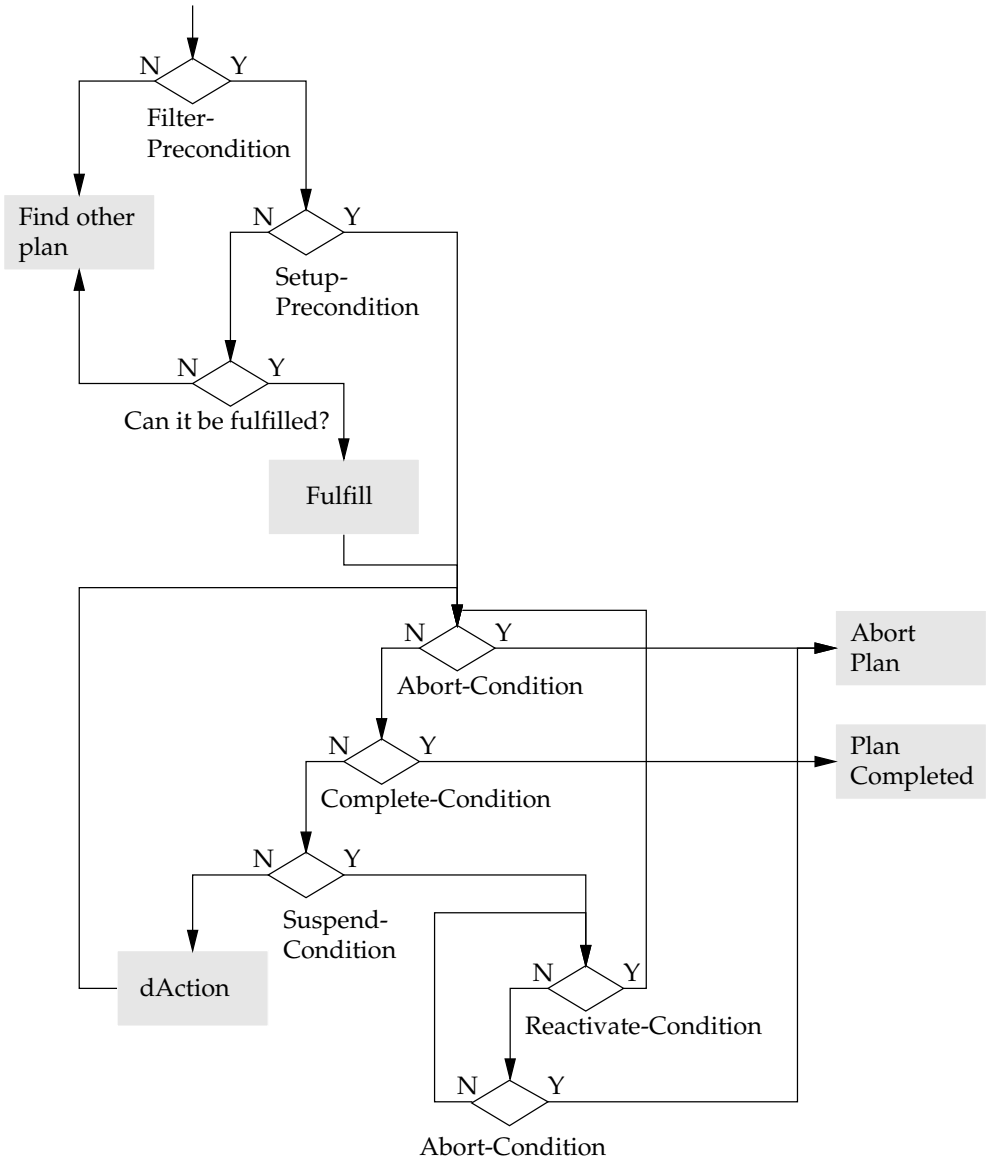


Figure 2.2: Flow-Chart describing the function of the different kinds of conditions. *dAction* means a very small portion of the action contained in the plan, that is performed before the next check if one of the conditions has become true. See section 2.5 for an explanation of the conditions.

2.5.4 Reactivate Condition

A suspended plan can be aborted (see next condition) or it can be resumed, hoping that it will eventually complete successfully.

This condition specifies when the plan can be reactivated. An example would be that the oxygen level has returned to normal and remained there for a certain period of time.

2.5.5 Abort Condition

A plan that clearly does not reach its goals must be aborted so that another kind of treatment can be performed. If that plan is part of the continuation condition or part of a plan that has no optional plans, then the containing plan also fails.

For practical work, this condition may be the most important. If the abort condition is too strict, too few patients can benefit from the treatment, but an overly optimistic condition, on the other hand, can cause much damage.

2.5.6 Complete Condition

The goal of a plan is, of course, to succeed. This condition specifies when a plan has completed successfully. In this case, the treatment can be stopped, and the next part of the containing plan can be performed.

The ultimate goal is to cure the patient. So if the top-level treatment plan has succeeded, she can be discharged from hospital.

2.5.7 Activate Condition

Whether a plan can be started automatically, as soon as all preconditions are met, is specified in this token (which is not really a condition, but distinguishes between two modes of operation). If it is not allowed to start automatically, a physician must first approve of the application of the plan.

2.6 Effects

Effects describe the outcome of a plan in terms of functional relationships between plan arguments and measurable parameters. These relationships can be exact formulas (where possible) or describe trends (e.g., glucose level falling when insulin is administered).

A probability of occurrence of the described effect can also be given.

2.7 Case Study

There is no point in including the BNF definition of Asbru here, but an example should help put some of the ideas introduced above into context. This example is also meant as a rationale for our claim that physicians will not write their plans in Asbru, even if they were aware of the benefits Asbru provides them with.

2.7.1 Natural Language

This example of a treatment plan for I-RDS (infant's respiratory distress syndrome) is taken from [Miksch et al., 1998]. It covers only the highest level of that plan hierarchy.

> After infants' respiratory distress syndrome (I-RDS) is diagnosed, a plan dealing with limited monitoring possibilities is activated, called *initial-phase*. Depending on the

severity of the disease, three different kinds of plans are followed: *controlled-ventilation*, *permissive-hypercapnia*, or *crisis-management*. Only one plan at a time can be activated, however the order of execution and the activation frequency of the three different plans depend on the severity of the disease. Additionally, it is important to continue with the plan *weaning* only after a successful completion of the plan *controlled-ventilation*. After a successful execution of the plan *weaning*, the extubation should be initiated. The extubation can be either a single plan *extubation* or a sequential execution of the sub-plans *cpap* and *extubation*.

The most important part is the sub-plan *controlled-ventilation*. The intentions of this sub-plan are to maintain a normal level of the blood-gas values and the lowest level of mechanical ventilation (as defined in the context of controlled ventilation therapy) during the span of time over which the sub-plan is executed. This sub-plan is activated immediately, if peak inspiratory pressure PIP $\leq$ 30 and the transcutaneously assessed blood-gas values are available for at least one minute after activating the last plan instance *initial-phase* (as reference point). The sub-plan must be aborted, if PIP $>$ 30 or the increase of the blood-gas level is too steep (as defined in the context of controlled ventilation-therapy) for at least 30 seconds. The sampling frequency of the abort condition is 10 seconds. The sub-plan is completed successfully, if $F_iO_2 \leq 50\%$, PIP $\leq$ 23, f $\leq$ 60, the patient is not dyspnoeic, and the level of blood gas is normal or above the normal range (as defined in the context of controlled ventilation-therapy) for at least three hours. The sampling frequency of the complete condition is 10 minutes. The body of the sub-plan *controlled-ventilation* consists of a sequential execution of the two sub-plans *one-of-increase-decrease-ventilation* and *observing*.

2.7.2 Asbru

The following sample of Asbru code shows the plan *controlled-ventilation* mentioned in the last sentence of the natural language example in the previous section. It is put into the context of the plans containing it (*I-RDS-Therapy* and *one-of-controlled-ventilation*), but these are not shown in great detail here.

Try to look at Figure 2.3 (on the next page) through the eyes of a physician, not a computer scientist.

```
(PLAN I-RDS-Therapy
  ...
  (DO-ALL-SEQUENTIALLY
    (initial-phase)
    (one-of-controlled-ventilation)
    (weaning)
    (one-of-cpap-extubation)
  )
)

(PLAN one-of-controlled-ventilation
  ...
  (DO-SOME-ANY-ORDER
    (controlled-ventilation)
    (permissive-hypercapnia)
    (crisis-management)
    CONTINUATION-CONDITION controlled-ventilation
  )
)

(PLAN controlled-ventilation
  (PREFERENCES (SELECT-METHOD BEST-FIT))
  (INTENTION:INTERMEDIATE-STATE (MAINTAIN STATE(BG) NORMAL controlled-ventilation *))
  (INTENTION:INTERMEDIATE-ACTION (MAINTAIN STATE(RESPIRATOR-SETTING) LOW controlled-ventilation *))
  (SETUP-PRECONDITIONS (PIP (<= 30) I-RDS *now*)
    (BG available I-RDS [[_, _], [_, _], [1 MIN,_] (ACTIVATED initial-phase-l#)]))
  (ACTIVATED-CONDITIONS AUTOMATIC)
  (ABORT-CONDITIONS ACTIVATED
    (OR (PIP (> 30) controlled-ventilation [[_, _], [_, _], [30 SEC, _], *self*])
      (RATE(BG) TOO-STEEP controlled-ventilation [[_, _], [_, _], [30 SEC,_], *self*])))
  (SAMPLING-FREQUENCY 10 SEC))
  (COMPLETE-CONDITIONS
    (FiO2 (<= 50) controlled-ventilation [[_, _], [_, _], [180 MIN, _], *self*])
    (PIP (<= 23) controlled-ventilation  [[_, _], [_, _], [180 MIN, _], *self*])
    (f (<= 60) controlled-ventilation [[_, _], [_, _], [180 MIN, _], *self*])
    (state(patient) (NOT DYSPNEIC) controlled-ventilation [[_, _], [_, _], [180 MIN, _], *self*]))
    (STATE(BG) (OR NORMAL ABOVE-NORMAL) controlled-ventilation
        [[_, _], [_, _], [180 MIN,_], *self*])
    (SAMPLING-FREQUENCY 10 MIN))

  (DO-ALL-SEQUENTIALLY
    (one-of-increase-decrease-ventilation)
    (observing))
)
```

Figure 2.3: An example of Asbru code (part of a clinical treatment protocol for Infants' Respiratory Distress Syndrome (I-RDS)).

Chapter 3

The State of the Art

Do not meddle in the affairs of Wizards,
for they are subtle and quick to anger.
(J. R. R. Tolkien, *The Lord of The Rings*)

Before our own solution to the problem of how to make Asbru usable by people other than computer scientists is introduced, we present and discuss the most common representations for treatment plans today, as well as existing approaches to visualizing selected types of data.

3.1 Existing Methods for Protocol Representation

In this section, the three main methods that are currently used for protocol design are introduced briefly, and their main deficiencies are discussed. These representations are: natural language, decision tables and flow-charts.

This section serves as a rationale both for Asbru itself, and for the fact that flow-charts (or Petri nets, for that matter) were not chosen for its representation.

3.1.1 Natural Language

The most obvious representation for a protocol is to describe it using a natural language. Due to the complexity of such protocols, such a description is often hard to understand and overly complicated. Natural language is badly suited for expressing cascades of *if-then* rules with many exceptions. It is very difficult to get an overview of such a plan, and to keep track of what actions have been done and what is needed at the moment. This can lead to incomplete and even contradictory plans.

This representation also leads to descriptions at a very high level that are difficult to use for any kind of automation support (and also by inexperienced physicians).

3.1.2 Decision Tables

To overcome the high complexity of protocols defined in natural language, decision tables are used. These consist of matrices of conditions and corresponding actions, where the conditions describe when an action is to be performed. Such tables do not capture the temporal dimension very well and do not offer a more abstract description of the task than the concrete actions.

Lacking a formal way of defining actions, they often contain a lot of text that is difficult to grasp quickly (see the previous description).

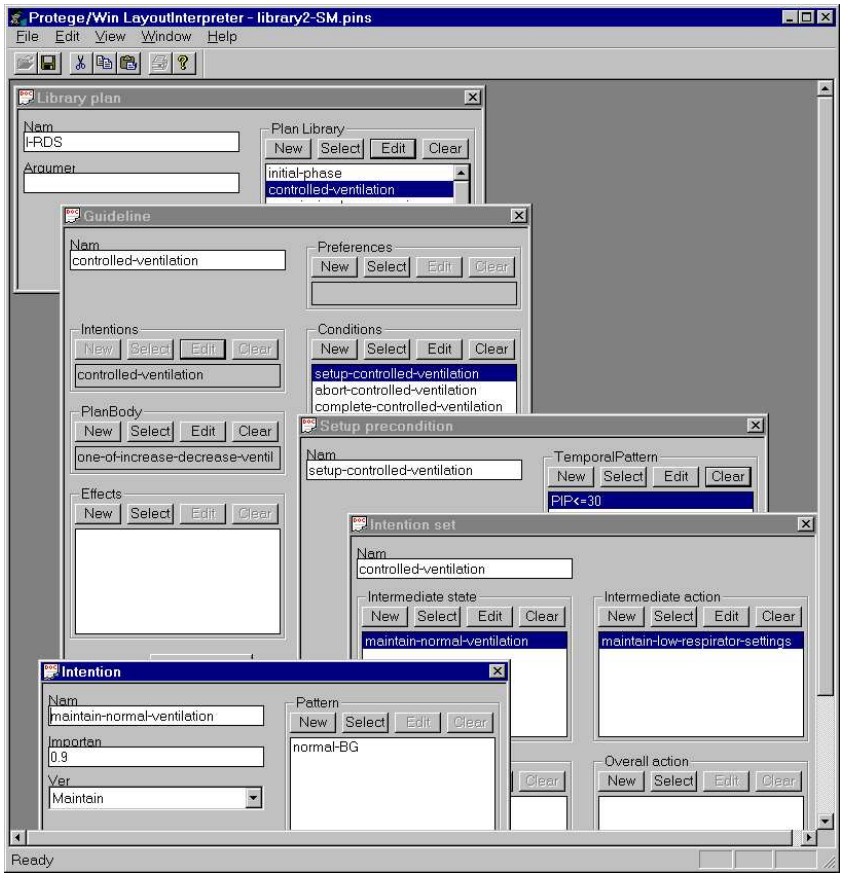


Figure 3.1: A screen-shot of PROTÉGÉ-II while an Asbru plan is edited. It is difficult to keep track of what one wants to do after a few levels of dialogs. (from [Miksch et al., 1998])

3.1.3 Flow-Charts

Flow-charts [Goldstine and von Neumann, 1947, Martin, 1973] are a very intuitive and lucid representation for computer programs, and small decision trees in general. But they lack the concept of time, and do not scale very well, i.e., become unreadable as soon as they exceed a certain size. They also lack a way of specifying actions that should be performed in parallel.

Figure 2.2 is a sketch of how a plan in Asbru would look like if specified using a flow-chart. In practice, not all conditions are needed for every plan, but most are. So, if a large number of plans or a complicated plan is drawn using flow-charts, the amount of detail presented makes comprehension of the overall plan almost impossible. This clearly contradicts the purpose of flow-charts, that were meant to make the connections between parts of a process easier to understand and to grasp at one glance.

3.2 Knowledge Acquisition Tools

As a further alternative for authoring plans, Asbru itself could be used, with the assistance of a knowledge acquisition tool like PROTÉGÉ-II [Musen et al., 1995]. This, however, proved impossible because of the high complexity of the language, and the little amount of abstraction that PROTÉGÉ is capable of. Users quickly got lost in the many windows that opened with fields to fill out (see Figure 3.1).

3.3 Visualization

We identified the following main challenges that would have to be solved in order to develop a usable visualization of Asbru plans (this is a refined and expanded version of the overview given in [Kosara and Miksch, 1999]; the descriptions of AsbruView will follow its structure).

An enormous amount of work has been done in the field of scientific and information visualization in the last few years, but most of these approaches focus on large amounts of multi-dimensional data. For this kind of problem, a number of good visualizations exist now, that make data accessible [Gross et al., 1997, Inselberg, 1997, Mukherjea et al., 1996]. An overview of the state of the art can be found in [Purgathofer and Löffelmann, 1997].

The specific combination of problems faced here, however, has apparently never before been investigated. Solutions (or at least basic approaches) exist only for parts of the problem. These will be discussed here briefly together with the descriptions of the problems themselves.

3.3.1 Hierarchical Decomposition

The connection between a plan and its sub-plans must be made clear, i.e., that a plan is made up of its sub-plans. The difficulty here does not so much lie in this problem alone, but in the fact that it must be communicated together with the other concepts described in this section. Any kind of tree view, like it is now used in many programs (especially file managers), could be used. A special method for this kind of information can be found in [Shneiderman, 1992], and another (more obvious one) in [McKinney et al., 1998].

Ways of displaying more *and* less detailed information in the same view at the same time are fish-eye views [Furnas, 1981], stretchable rubber sheets [Sarkar et al., 1993], and the perspective wall [Mackinlay et al., 1991]. The latter does not distort the diagram as much as the other two approaches, and is also better suited for implementation with hardware support.

Another interesting idea is to use a logarithmic time-scale, and display cruder information the cruder the time scale gets. This way, [Powsner and Tufte, 1994] manage to display an enormous amount of information on just one sheet of paper.

3.3.2 Plan Types

A plan's type should be easy to tell from its graphical representation, especially if it has sub-plans. This is a contradiction to the previous definition of a plan — which either has sub-plans or is an action (which, by definition, has no type. The type only specifies the way its sub-plans are to be performed). For practical work with plans, however, one will need to define a plan's type before any sub-plans are added to it.

3.3.3 Temporal Order ('Topology')

The way a plan's sub-plans are to be performed must be communicated somehow by the visualization.

For *any-order* plans, only the set of plans to be used is known, but not the order in which they will be performed. A way of depicting a plan has to be found where the order in which they are depicted does not necessarily correspond to the order in which they will be executed.

Flow-charts [Goldstine and von Neumann, 1947, Martin, 1973] are usually used for this purpose (order of execution), but they do not cover parallel plans or sets of plans that can be performed in any order (the latter is possible[1], but only with considerable effort that leads to diagrams that are impossible to read — which definitely is not what flow-charts were intended for). Additionally, flow-charts scale very poorly, i.e. become unreadable when a large number of plans is defined, and they do not cover the temporal aspect (see below).

[1] By defining one path for every possible permutation of the plans. For n plans, this means $n!$ different paths.

Another interesting idea is the way railroad schedules are drawn for the Japanese Shinkansen trains [Tufte, 1990]. On such a diagram, there is a horizontal time axis, and the train stations are put next to each other on the vertical axis. A line is drawn for every train that connects the points in time when the train stops at a station. This way, a large number of trains can be drawn on one diagram without sacrificing readability. It is also easy to see connections between trains. But any-order plans are still hard to draw, and the duration of a plan (or even temporal uncertainty) is next to impossible to include in such a diagram.

3.3.4 Compulsory vs. Optional Plans

A sub-plan can be used in two different ways: it either *must* be executed (compulsory plan) or it *can* be (optional). While a compulsory plan is easy to understand (and to depict), a way of indicating that a plan is optional is a lot more difficult, especially if it must be different from the representation of temporal uncertainty (see below). A blurred depiction of plans [MacEachren, 1992] therefore cannot be used.

3.3.5 Cyclical Plans

Cyclical plans were described in section 2.2.6. These are the most difficult, because not only their duration and end times (see next section) can vary over a long time, the number of applications of their single sub-plan is not known, either.

We tried sphere and cylinder metaphors (inspired by [Gross et al., 1997]), but that did not lead to usable representations.

3.3.6 Temporal Uncertainty

The time a plan takes, but also time spans that are considered for the relevance of symptoms are not defined in terms of exact durations. Therefore, a way of visualizing time spans, where only part of the information (e.g. the minimum duration) is known, must be found. This information may be refined later; this is called a *minimum-commitment approach* [Stevenson et al., 1996] (even though the term *late commitment* would be more appropriate here).

A related problem is that of temporal granularity. It should be possible to tell to what accuracy a point in time has been defined (e.g., seconds, minutes, etc.).

Ways of indicating uncertainty can be found in [MacEachren, 1992, Stevenson et al., 1996], but are very limited. These approaches only tell the reader that the data is uncertain, but not to which degree.

A very versatile — albeit difficult to understand — solution to this problem can be found in [Rit, 1986]. While the methodology proposed there is very powerful, it is badly suited for *displaying* more than a few plans, especially when they are to be executed in parallel or when they overlap.

A time annotation in Asbru consists of seven values, and thus can be understood as a point in seven-dimensional space. Probably the most usable approach to visualizing this kind of data are parallel coordinates [Inselberg and Dimsdale, 1987, Inselberg, 1997]. They are, however, not useful here since they do not clearly indicate the relations between the different quantities, and are generally better suited for independent data (the parts of a time annotation are, of course, highly dependent of each other).

The most promising way of visualizing temporal uncertainty are glyphs [Pang et al., 1996, Chuah and Eick, 1997], or Chernoff faces [Chernoff, 1973], which is the solution we finally used.

Chapter 4

Basic User Interaction Requirements

Go not to the Elves for counsel,
for they will say both no and yes.
(J. R. R. Tolkien, *The Lord of The Rings*)

AsbruView must not only be able to provide a visualization of plans, but also the means to manipulate them. This chapter lists some of the basic requirements of a modern user interface.

4.1 Direct Manipulation

AsbruView is meant to contrast the manual entry of plans using the language Asbru. One of the key problems with editing a document in any formal language is that one has to obey a very strict syntax. This probably was the main reason for the development of graphical user interfaces.

The alternative to a graphical interface is a knowledge-acquisition tool (see Figure 3.1) or a structure editor. These do not, however, provide a higher abstraction level than just the language itself — they only help in avoiding syntax errors. Graphical interfaces enable the user to decide the sequence of actions to provide all the information needed for a task.

Identifying an object is easier with a graphical user interface than when one has to remember a name or a more obscure way of identification. Usually, the user also has direct access to a number of actions she can perform with the objects. Shneiderman calls this *the disappearance of syntax* [Shneiderman, 1997a], because actions that once required the user to remember and apply a complicated syntax are now performed by pointing and clicking (and usually without the need to follow a certain sequence of actions). This more immediate way of working with objects is called *direct manipulation* [Shneiderman, 1997b].

Direct manipulation has been criticized [Gentner and Nielson, 1996], because it restricts the versatility of a system. But because of the target group (physicians with little or no experience with computers), most other ways of interactions (especially command line interfaces) cannot be used.

The need for more powerful tools (like scripting) will perhaps emerge as more and more plans are entered, and more and more people work with the system and become more proficient in its use. At the moment, however, ease of use and a quick overview is more important.

4.2 Forms of Plan Manipulation

A number of different operations must be possible with plans.

It must be possible, of course, to create and delete plans. The user must be able to move plans in order to change the sequence of a plan's sub-plans, or to move a plan to another containing plan.

There also must be a means to change a plan's attributes (its intentions, conditions, etc.). These attributes are very different, so there needs to be more than just a form to fill out. Separate editors are needed to capture the differences appropriately, and to make the specific features of every of those aspects apparent.

But more than that, the user must be able to change a plan's type at any time. During the design of a larger plan, plans will be moved and changed, but also the decision, which plan type to use, will be delayed or rethought and changed. In such a case, it would be very frustrating to copy all the sub-plans, conditions, etc. of that plan to a new one with the correct type, and then delete the old one. Instead, a plan change must be possible so that the plan retains as much information as possible (e.g., also the continuation condition, when the change is made from a *some-any-order* plan to a *some-together* plan).

4.3 Level of Detail

The hierarchical structure of Asbru plans must be accessible through the interface, to provide the natural modularization inherent in the language.

Thus, the interface must provide the user with the possibility to change the part that is seen, and the amount of information that is shown at the same time [Shneiderman, 1996].

So in order to get a better overview, the user might decide that she does not want to see the sub-plans of a plan, or that she only wants to see two levels of plans. But in order to see the details of a particular plan, it must still be possible to expand just this one plan.

4.4 Clipboard, Copy & Paste

Many programs today allow the use of a clipboard to store information in temporarily. This concept should be available in AsbruView as well, but in a manner that is compatible with the concepts of that language.

Copying plans is a mechanism that contradicts the idea of reuse in Asbru. But for many tasks, it is convenient to copy a plan and change some of its attributes instead of entering all the information again.

Copying must also be possible for parts of plans, especially time annotations and conditions.

4.5 Storing of Plans

In Asbru, every plan is stored in a separate file. The user interface should hide this fact from the user, so that she only works with one plan and its sub-plans, but not with many separate plans. Still, it must be possible to insert an existing plan anywhere in the plan currently being edited.

Later on, plans will be stored in a database, so that the concept of separate files will not exist any more.

Part II

Design and Implementation

Chapter 5

AsbruView

One View to rule them all,
One View to find them,
One View to bring them all
and in the darkness bind them
(J. R. R. Tolkien, *The Lord of The Rings* (slightly edited))

The solution to (most of) the problems described in the previous part is called AsbruView. Its name is somewhat inappropriate, because the system not only deals with visualization of plans, but also with their manipulation.

This chapter covers the visualization part of AsbruView.

We could not find one way of representing all aspects of Asbru, so we developed two separate views of the same underlying model (see Figure 7.1) that are quite complementary, and that serve different purposes.

These two views (earlier versions were presented in [Miksch et al., 1998, Kosara et al., 1998]; a version that is almost identical to the one presented here appears in [Kosara and Miksch, 1999]) are introduced in the next two sections; in section 5.3, a discussion of the common concepts of both views follows. The structure of this discussion follows the layout of section 3.3.

5.1 Topological View

This view deals with the topology, or layout, of the plans. It shows, which plans are to be performed in sequence, in parallel, etc. It does not deal with temporal issues.

In order to make this view easier to understand and remember, a number of metaphors are used for Asbru's concepts.

A lot of theory has been produced about metaphors [Ortony, 1993]; but for the purposes of this discussion, a simple classification into three categories shall suffice.

The first class of objects that is sometimes called metaphors (and that originally inspired the development of the metaphors used here [Cole and Stewart, 1994]) is what is called a glyph in section 3.3.6: A graphical object whose features express the values of certain attributes that are to be shown.

The second class are metaphors that closely match the concepts that they depict. This kind of metaphor is quite common in graphical user interfaces, for example rubbish bins (or more politically correct: recycle bins) are used for deleting files, and simple depictions of mailboxes are used to show the user whether or not there is new email (even though this metaphor is often used in the wrong way [Mullet and Sano, 1995]).

The third kind of metaphor — the type used here — are different in the sense that they try to make abstract concepts easier to understand by using symbols from everyday life that have no

direct connection with the concepts they stand for (in contrast to the close relationship between email and real mail, for example).

5.1.1 Anatomy of a Plan

The basic metaphor behind a plan is that of a running track. Each plan is depicted as a running track with a finishing flag (see Figure 5.1).

During the execution of the plan, the physician (or the patient) is considered to be running along the plan, starting at the left side, until she reaches the finishing flag when the plan has succeeded.

Perspective

The metaphorical three-dimensional objects are depicted using parallel projection. The way this is done here is different from the usual views, where objects that are farther away from the viewer are drawn more to the right (rather than to the left, like here) than objects that are closer. The reason for this strange perspective is that, in our view, the user finds herself at the beginning of the plan, other than (in what we might call the traditional view) at its end. This is, of course, dependent on where the starting point of the running track is. The time axis in the western world is drawn from left to right (which obviously is a consequence of the way we read and write), so the logical choice for a starting point is the left side.

From this point of view, the user also can identify objects that lie ahead of her on the way along the plan, like traffic signs signifying conditions (see section 5.1.8).

To add to the impression of real three-dimensional objects in a small world, a background pattern is drawn (see Figure 5.10). There are three choices: a check-board pattern, a grid, or no pattern at all.

Because of the parallel projection, parts of plans might be hidden by the finishing flags of preceding plans. To avoid this, the finishing flags are drawn semi-transparent (except when conditions are shown, see section 5.1.8).

Dimensions

The three dimensions used in this view represent different abstract dimensions of the underlying plan.

The most obvious is the temporal (or length) dimension, which in this view is not used as a precise scale, but rather as a general direction. That means that the length (i.e., the extent along the time axis) of plans in the temporal dimension does not correspond to their actual temporal duration. But plans that are put next to each other in the time direction are to be performed in that order.

A second dimension is that of parallel plans (width). The notion of a direction rather than a scale is easier to grasp here, because there is nothing that can be associated with the width of a plan. Plans that are put alongside[1] each other are performed concurrently.

Different plan levels are put along the third dimension, on top of each other (height). Sub-plans are put on top of their super-plans — thus effectively reversing the original way of seeing levels. But it is very hard to understand something growing upside down, and to manipulate it, because it is too far off from daily experience.

There is also a fourth dimension: color, which is heavily used. For a description of the meaning of color in AsbruView, see section 5.3.1.

[1] The heavy use of metaphor is not confined to visual metaphors, but is also visible in the language used to describe the graphical representation.

The Name Sign

The name of a plan is displayed on a small sign that faces the user. This sign is centered on the plan, and changes its size with the length of the name. Should the name be too long (so that the sign would have to be longer than the plan itself), the name is shortened to fit, and an ellipsis ("...") is appended so the user can tell that she does not see the full plan name.

But the name sign is only centered on the plan if that does not mean that a part of it is hidden. In case part of the plan is not visible, the name sign moves as far in the direction of view center as possible (i.e., so as not to leave the plan). So, when the user scrolls horizontally, name signs move so that as much information is displayed as possible. Otherwise, the name signs of large plans would only be visible at certain scrolling positions. For an illustration of this point, see Figure 5.2.

5.1.2 Hierarchical Decomposition

Plans are stacked on top of each other (*Levels* dimension, see previous section) to depict the hierarchical decomposition of plans.

This decomposition at the same time provides a kind of abstraction hierarchy: a plan's sub-plans are more detailed than the containing plan. Thus, in order to get a better overview or to see more detail, one can hide or display the contents of a plan, respectively [Shneiderman, 1996].

This is done by clicking on the small triangle on the right of the plan's front face. This triangle only appears when a plan has sub-plans, and then can be used to 'open' or 'close' the plan, i.e., show or hide, respectively, its sub-plans (Figure 5.3).

5.1.3 Plan Types

A plan's type is not indicated by any symbol, but can be told from the way its sub-plans are arranged — see the next section (the different plan types are shown in Figure 5.4).

5.1.4 Temporal Order ('Topology')

In this view, the temporal order is the only way a plan's type is indicated (except for cyclical plans, see section 5.1.6). Figure 5.4 shows examples of the different plan types/layouts.

Sequential Plan

The sub-plans are put next to each other parallel to the time dimension, so that as soon as the metaphorical runner reaches the end of one sub-plan, she steps on the next one.

This simple chronological order of actions is used in many other systems, and also in virtually every diagram that includes time.

Parallel Plan, Some-Together Plan

In this case, the sub-plans are put next to each other along the parallel plans axis, and so have a common start time. In this view, they also have a common end time, but this is just an arbitrary decision. If the plans would have been given different lengths to illustrate the fact that they do not have to end at the same time, those lengths would have been as arbitrary (and probably more misleading — why is Plan B longer than Plan C?).

Any-Order Plans

Plans are put on the containing plan in a pattern that is meant to show that it has nothing to do with the real sequence of the plans. The fact that there is space between the plans is meant to add to the impression that the way the plans are put on their super-plan is arbitrary. The length

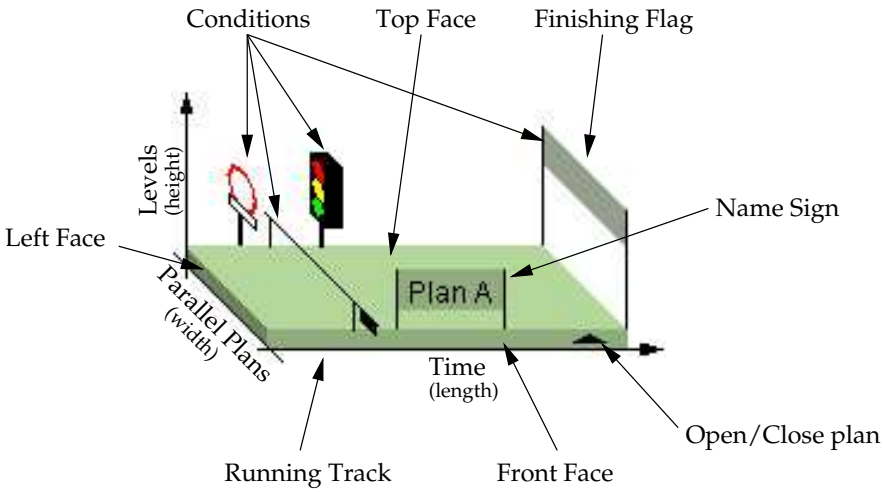


Figure 5.1: Anatomy of a plan in Topological View.

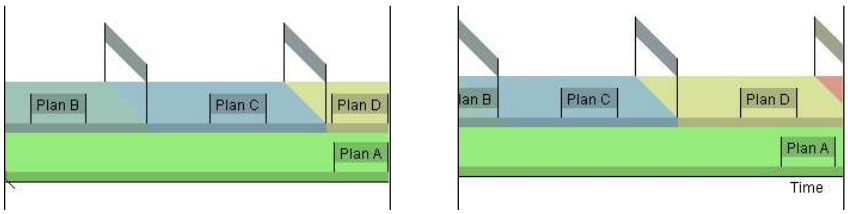


Figure 5.2: Plans' name signs moving about. The two parts show the same plans, but from the left to the right part, the viewing window has been moved to the right. This has caused the name sign of Plan B to move to the right and Plan D to become centered again.

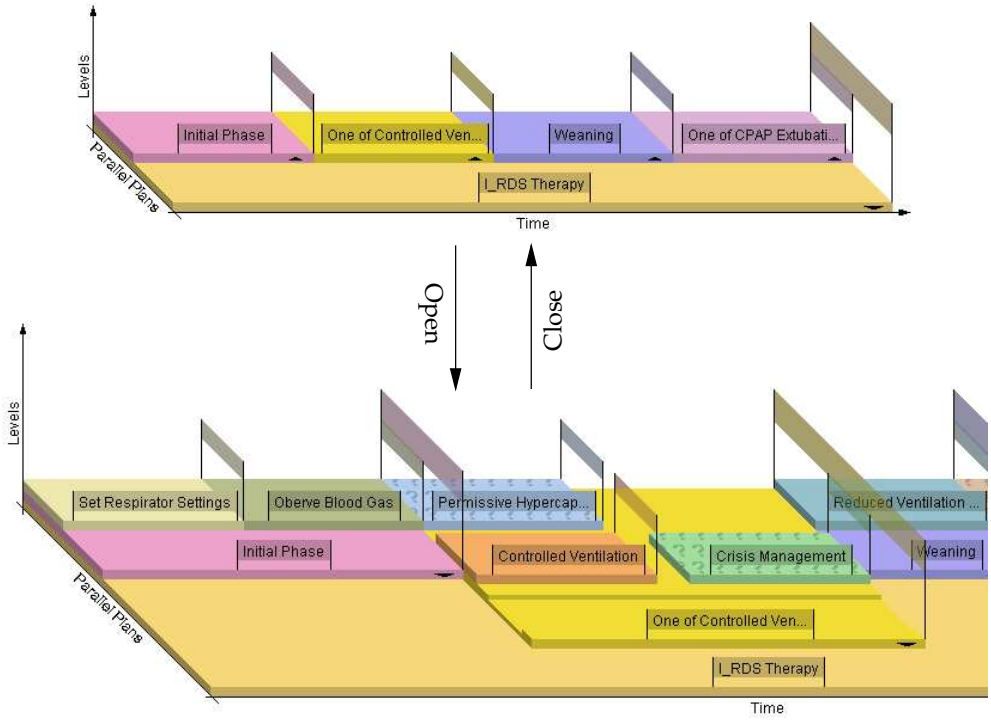
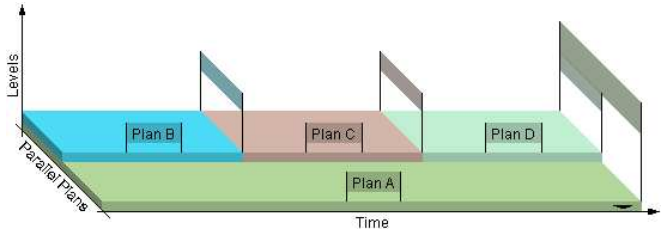
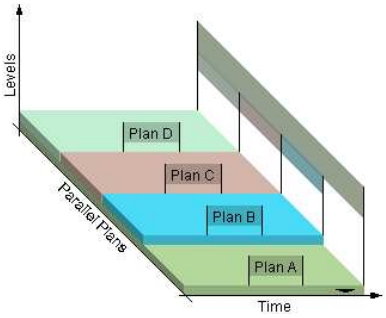


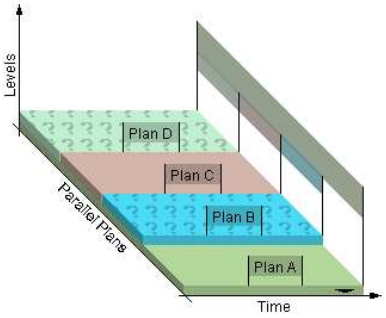
Figure 5.3: Opening and closing of plans in Topological View. The plans in the upper image are closed, and thus show a higher level of abstraction. The small triangles on every plan indicate that they have sub-plans. The plans in the lower image are opened, revealing more detail, but making it harder to get an overview.



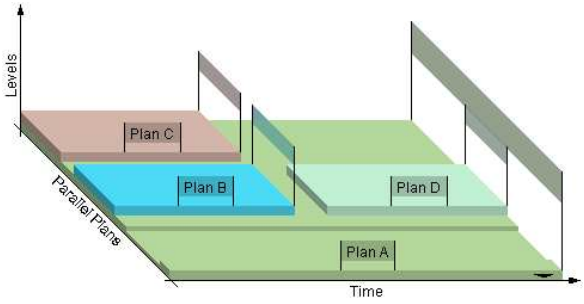
(a) Sequential Plan



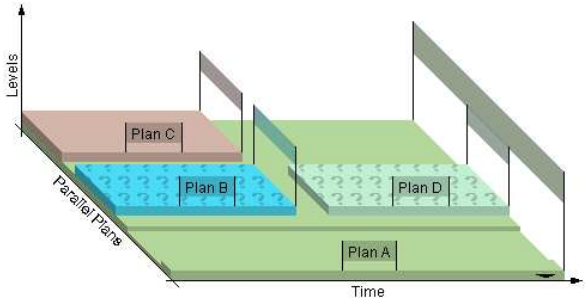
(b) Parallel Plan



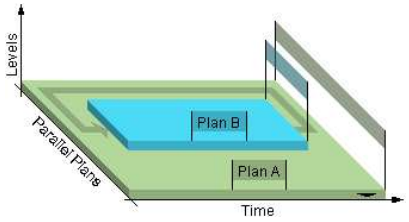
(c) Some-Together Plan



(d) All-Any-Order Plan



(e) Some-Any-Order Plan



(f) Cyclical Plan

Figure 5.4: All of Asbru's plan types in Topological View.

of the containing plan is not — as is the case with sequential plans — the sum of the lengths of its sub-plans. This is done for esthetic reasons, but also adds to the perception of a random plan placement.

The plans can be put in a groove at the front of the plan as soon as the order of application is clear.

Cyclical Plans

This type of plan is described in detail in section 5.1.6.

5.1.5 Compulsory vs. Optional Plans

Optional plans are marked by a question-mark texture on the top face. An alternative would have been to draw optional plans semi-transparent, but this proved problematic when a number of plans (optional and compulsory) were stacked on top of each other.

Because the continuation condition (which decides about whether a plan is optional) is part of the containing plan, the optional mark is independent of whether or not the plan has sub-plans.

5.1.6 Cyclical Plans

The single sub-plan of a cyclical plan is put on top of that plan, and an arrow is drawn from the end of the sub-plan to its beginning. In addition to the way this is done in the prototype, more information should be available in the Topological View, like the minimum or maximum number of retries. This can be done by putting a further flag on the arrow with a short description, like "<4" for a maximum of four tries (i.e., the arrow is used less than four times). This additional flag was not implemented in the prototype.

5.1.7 Temporal Uncertainty

This aspect is not covered by the Topological View. Due to the perspective distortion, it is impossible to use a precise time scale, and draw plans so that they reflect temporal constraints correctly. The view to deal with this issue is the Temporal View described in section 5.2.

5.1.8 Conditions

Conditions themselves are not shown in this view, but the information, which conditions are defined, is. Metaphors are used here as well (see Figure 5.5). These come from the world of traffic control, which is not very closely related to track-and-field sports (where the running tracks come from). But both have to do with movement (and most people are familiar with the used symbols), so the connection should be easy to make.

These elements are drawn in gray when the corresponding condition is not defined, and in color otherwise (see Figure 5.6).

The first traffic sign the runner encounters is a "no entrance with exceptions" sign, which is used for the filter precondition. The traffic sign means that nobody may enter the road this sign is put next to, except people who are listed on a small additional sign — which is very similar to the way the filter precondition works (see section 2.5.1).

A barrier is used as a metaphor for the setup precondition (section 2.5.2). If this condition is not fulfilled, the barrier is closed; but it opens as soon as the patient meets the criteria.

A traffic light is used for three conditions: The red light stands for the abort condition (section 2.5.5). In this metaphorical world, a red light never changes back to yellow or green.

The yellow light is used for the suspend condition (section 2.5.3). A yellow light means "Attention!", and this is in a way similar to the meaning of the suspend condition (where an emergency plan may be performed while this plan is suspended). And it is also easy to understand

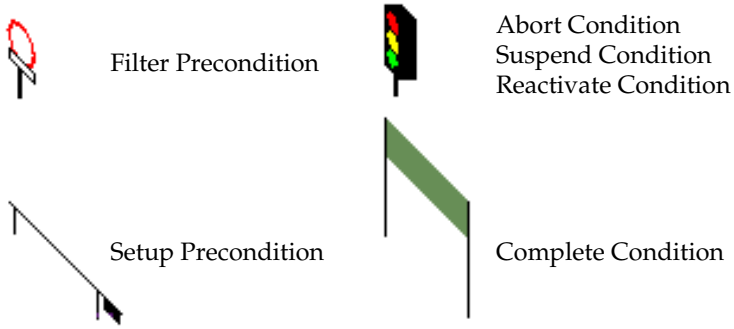


Figure 5.5: Condition Metaphors.

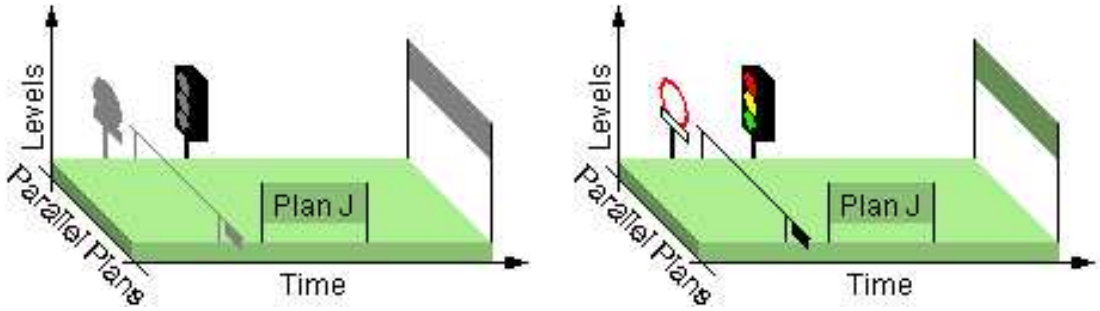


Figure 5.6: Undefined (left) and defined conditions (right).

that often a plan will first be suspended, and if the emergency plan does not work, it will ultimately be aborted from the suspended state.

The green light symbolizes the reactivate condition (section 2.5.4). When after a suspension the criteria for continuing the plan are met again, the green light signals that normal work can continue now.

The finishing flag that the runner must pass in order to win (or at least reach the goal) is used for the complete condition. This finishing flag is usually drawn semi-transparent in order not to obstruct the view to objects behind it. It is drawn with solid color (or solid gray) only when conditions are shown.

In the prototype, these condition metaphors only appear when the "Show Conditions" checkbox is checked, and then conditions are only shown for plans without visible sub-plans (i.e., plans without sub-plans or closed plans). This helps avoid cluttering the display with too much information (and also speeds up rendering).

5.2 Temporal View

This view is two-dimensional and strongly based on the idea of LifeLines ([Plaisant et al., 1996, Plaisant et al., 1998]), which in turn is based on an old and quite familiar (from video editing software, for example) concept often called Time Lines or TimeLines [Tufte, 1983] (an application of this idea to immunization planning can be found in [Brandt et al., 1997]).

The basic idea is to draw a diagram with a horizontal time axis, and then divide the space above the axis vertically into regions for every event that is to be depicted. In this vertical region, a line is drawn over the time span of the corresponding event.

While this works very well for events whose temporal extent is known (i.e., past events), it

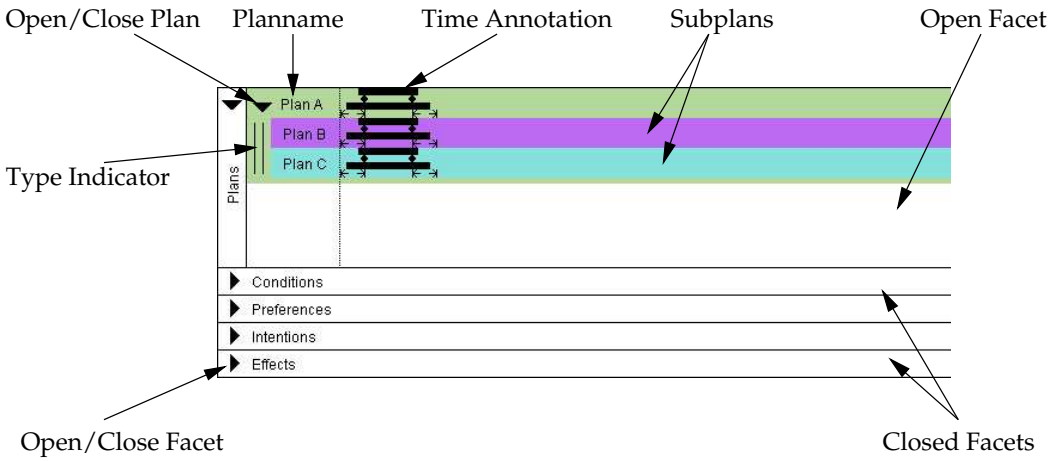


Figure 5.7: Anatomy of a plan in Temporal View.

is not usable for the kind of temporal uncertainty used in Asbru. Therefore, a special glyph was developed that replaces the simple line in order to make LifeLines more powerful (see section 5.2.7).

Another idea taken from LifeLines is that of facets. A facet is a collection of information about an object that is displayed on the same time axis as other facets, and can be expanded or hidden in order to get more detail information or a better overview, respectively.

There is one facet for the plan layouts, one for the conditions, etc. In the prototype, only the plan layout and condition facets have been implemented.

This view can also be understood as a kind of "bird's eye view" of the Topological View. Although the representation does not follow the same rules, it has some elements in common with a projection along the levels axis.

Yet another way of looking at this view — from the conceptual rather than from the visual point of view — is as a block-structured diagram [Nassi and Shneiderman, 1973]. The hierarchical structure is identical to a program where sub-routines are in-lined, and the type of operator for a block is shown in the left part of the block, like in loops (even though most programming languages do not have such a rich set of different loop constructs).

5.2.1 Anatomy of a Plan

A plan here consists of a rectangle that may contain other rectangles (sub-plans). If a plan has sub-plans, a small triangle appears on its very left, pointing to the right, if the plan is closed (i.e., its sub-plans are not shown), or pointing down, if it is opened (see Figures 5.7 and 5.8).

The plan's name appears right of the triangle. Right of the name, a dotted line is drawn that marks the begin of the time axis. The part of the plan to the right of that line can scroll horizontally when the user moves the time window that is visible. Here, a time annotation glyph (see section 5.2.7) is drawn for the time span in which the plan may be executed.

5.2.2 Hierarchical Decomposition

A plan's sub-plans are drawn a little further to the right, inside the rectangle of the containing plan. If this view is understood as a bird's eye view of the Topological View, this is very similar to the depiction there, with sub-plans stacked on top of plans.

This leads to a tree structure that is quite similar to tree views that are now used in many programs for displaying hierarchies of data (directories, settings, etc.).

5.2.3 Plan Types

When a plan has sub-plans, its type is indicated by a small symbol left of its sub-plans. When it has no sub-plans, no such symbol is drawn (a plan's type mainly influences the order of execution of its sub-plans).

Sequential Plans

For a sequential plan, bullets like in a bullet list are drawn to show that all the plans are to be performed, one after the other.

Parallel Plans

In case of a parallel plan (*all-together* or *some-together*), two parallel lines are drawn.

Any-Order Plans

For an *any-order* plan, two arrows pointing into opposite directions are drawn left of the sub-plans to show that the order depicted is not necessarily the real order of execution. These point to other points in time when the plan might be executed instead of the time where it is drawn. This is not entirely correct, though, because the sub-plans of an *any-order* plan can not only be executed in one piece, but one can interrupt the other (when that other plan is suspended). But this is not known at design time, of course.

Cyclical Plans

A cyclical plan draws a small circular arrow left of its single sub-plan.

5.2.4 Temporal Order ('Topology')

The temporal order of plans here is represented by Time Annotation glyphs (see section 5.2.7) that are put on a time axis. These glyphs not only specify the duration of a plan, but also its position in time.

5.2.5 Compulsory vs. Optional Plans

Just like in Topological View, a question-mark texture is applied to plans that are optional, compulsory plans are just filled with one color.

5.2.6 Cyclical Plans

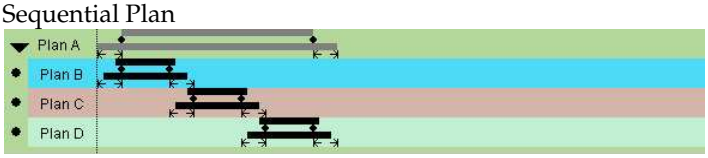
In the existing prototype, only a small circular arrow is drawn next to a cyclical plan's sub-plan. But in Asbru, a cyclical plan has more properties that could be represented graphically. One is the maximum number of retries. Similar to *any-order* plans, arrows could be drawn to other possible occurrences of the plan, but there a "shadow" of the time annotation would have to be drawn in order to differentiate between the semantics of *any-order* and of cyclical plans. Currently, only the circular arrow is drawn.

5.2.7 Temporal Uncertainty

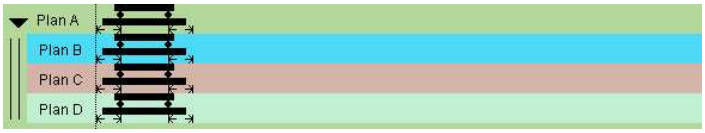
Time Annotations are depicted using a glyph [Chernoff, 1973, Chuah and Eick, 1997] that can capture all of their complexity (see Figure 5.9).

The four time shifts ESS, LSS, EFS and LFS (see section 2.1.3) are shown as vertical lines, on which a bar representing the MaxDu rests. On top of the MaxDu bar, two diamonds (or circles, depending on whether LSS and EFS are defined) support another bar representing the

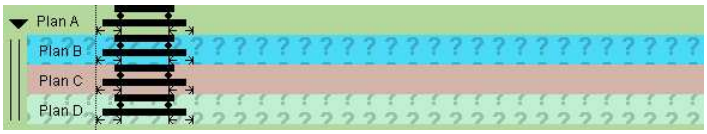
(a) Sequential Plan



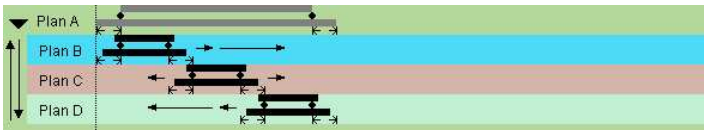
(b) Parallel Plan



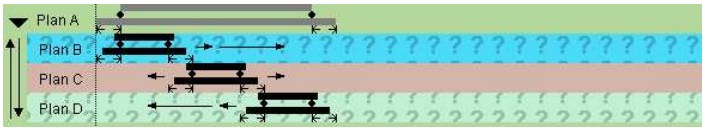
(c) Some-Together-Plan



(d) All-Any-Order Plan



(e) Some-Any-Order Plan



(f) Cyclical Plan

Figure 5.8: All of Asbru's plan types in Temporal View.

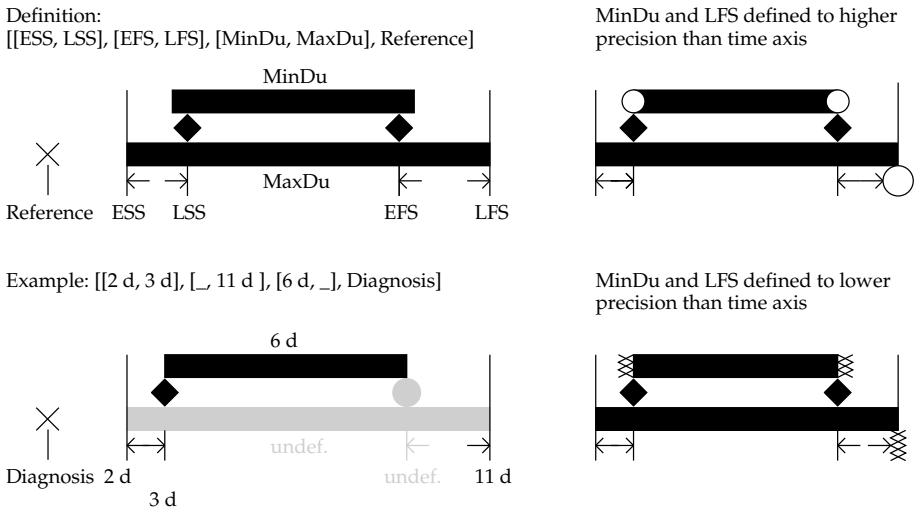


Figure 5.9: Time Annotations, schematic

MinDu. The arrowheads that point towards the vertical bars for the different time shifts are used to determine whether or not EFS is before LSS, which would not be visible otherwise.

If one of LFS, ESS is not defined, the corresponding diamond supporting the MinDu bar becomes a circle, which will move if the MinDu or the other time shift the MinDu is anchored on, changes. So in this case, an implicit definition of that time shift is given based on the known shifts.

The length of the MinDu bar is constrained by the difference between EFS and LSS, and so cannot become shorter than this difference. Because in that case, it would not be supported by the diamonds any more, and would fall down.

The different time granularities have not been implemented in the prototype, only the basic time annotation glyph. If the whole time annotation becomes too small, however, it is displayed as a circle in the program.

Time annotations are propagated from sub-plans to their containing plans, and there are a number of conditions (which also can involve more than one time annotation) that must be fulfilled in order for a plan to be correct. This was the topic of a Ph.D. thesis [Duftschmid, 1999], and was not implemented in the prototype.

5.3 Common Concepts

This section sums up the common features of both views. Such common features make the system easier to learn, especially when it is as far off most standard software as AsbruView (Figure 5.10 shows a screen-shot of the prototype).

5.3.1 Colors

Each plan is represented by a graphical object, which needs to be assigned a color. These colors are the same for a given plan in both views, thus making identification across views (and also within the same view when a plan is reused) easier.

The way colors are chosen needs to be looked into more closely. At the moment, colors are chosen randomly, with certain constraints (i.e., the color must not be darker than a certain value, that is, the average of all three components of the color must be greater than a constant). This usually leads to colors that work together quite well, but sometimes, very bad colors are generated

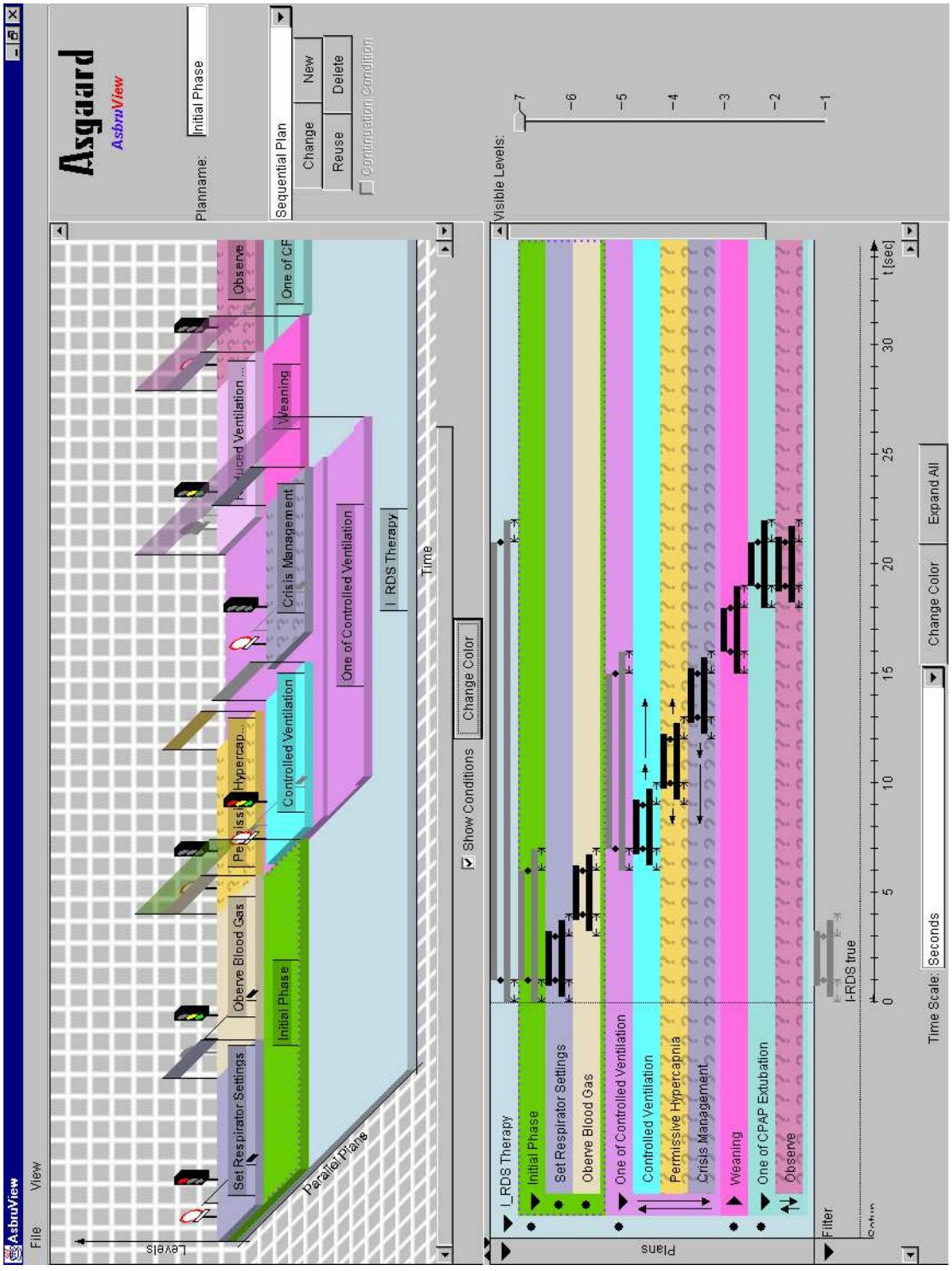


Figure 5.10: A screen-shot of the AsbruView prototype showing a part of a plan for ventilation of new-born infants.

(bright yellow or highly saturated red).

Because of the heavy use of color, gray can be used in contrast for every part that has not been defined, or that is not known.

One question that arose quite early in the project was that of color-blind people, who are not able to distinguish between certain colors. Another problem is that of "color psychology" — different colors have different meanings. So if a plan appears red or yellow, one might think of it as a particularly important plan, or an emergency plan.

So while colors are a very important part of a user interface, their current use in AsbruView must be questioned. There are several interesting ideas how to use color, but these did not make it into this thesis.

One example would be to assign different color ranges to different dimensions, thus creating colors that fit together better, and that add to the perception of the plan layout. In the levels dimension, plans' colors could range from white to saturated colors, thus embodying a metaphor of fog, that mountain tops stick out of. This would also make the notion clearer, that plans at lower levels are more general and abstract than higher level ones (especially actions).

5.3.2 Tree Structure

The structure of plan definitions in Asbru is not exactly that of a tree, because reuse can create connections between branches of such a tree. The notion of a tree (which, in the case of the Topological View, even grows downside up, in contrast to trees in computer science in general) is present, though.

The depiction of the tree in Temporal View is very similar to the way directory trees are shown in many file manager programs. This makes it easier to learn, even though many people do not recognize this similarity at once.

5.3.3 Optional Plans

In both views, optional plans are drawn with a question-mark texture instead of just one color.

Thus, the depictions of the same plan in both views can be recognized easier and users remember the concept more easily.

Chapter 6

User Interaction

Ho! Ho! Ho! to the bottle I go
To heal my heart and drown my woe.
(J. R. R. Tolkien, *The Lord of The Rings*)

This chapter describes a few user interaction concepts implemented in AsbruView, especially those, that are different from most other programs.

6.1 Plan selection

In both views, a plan is selected by clicking on it. A border consisting of moving points ("ants trail") is drawn around it so that the user knows which plan is currently selected. This is also the same in both views, and so should be easy to recognize.

6.2 Three-Dimensional Navigation

In order to move a plan from one location to another in the Topological View, the user would have to be able to move an object (plan) in three dimensions. This is not possible with standard input devices (mice), so a trick is used.

If the user presses the mouse button while the mouse pointer is over a plan object, that plan is selected. If the user then moves the mouse without releasing the mouse button, that plan is moved with the mouse (and is drawn semi-transparent in order not to block sight to the plans that it will be put on). But the plan does not simply follow the mouse, because that would be hard to understand in the context of a three-dimensional world. Rather, a shadow in the shape of a small cross appears under the plan, which is the point where the plan will be put when the user releases the mouse button.

The shadow follows the mouse like this (horizontal and vertical movement is defined as movement that does not deviate more than 20 degrees from a purely horizontal or vertical movement, respectively. All other movement is considered oblique):

- Vertical Movement. This kind of movement is ignored. Thus, the user can adjust the position of the plan when she has moved it outside of the view, or when it has "crashed" into the other plans.
- Horizontal Movement. The shadow follows the horizontal part of the movement, ignoring any vertical components.
- Oblique Movement. The shadow follows the mouse movement exactly. Thus, the shadow seems to move back and forth "into" or "out of" the screen.

If the cross is over a plan, the program decides where that position will put the plan currently being moved: If it is over the middle half of the plan, it will be put onto that plan (i.e., added as a sub-plan) if dropped. If it is over the quarter of the plan near an edge, it will be put before or after that plan, respectively (where *before* and *after* mean different locations in the direction of parallel plans for parallel and any-order plans). The plans are marked accordingly by an ants trail.

If there is no plan under the shadow, the plan originally containing the plan currently being moved is considered the target. Thus, the user can cancel a move by releasing the mouse button while the cross is on the background.

A plan does not accept as sub-plan a reused instance of itself or of one of its sub-plans. Thus, an infinite recursion is avoided.

6.3 Clipboard

The German translation for clipboard is *Zwischenablage*, which, retranslated, means *temporary storage* or *storage tray*.

This meaning could be made into a metaphor by drawing a tray on the screen. Blocks representing plans could then be put into it or taken out of it like objects in the real world. This would probably be easier to understand than an invisible clipboard.

This idea was not implemented in the prototype.

6.4 Changing a Plan's Type

A plan's type can be changed at any time (see Figure 6.1). Depending on the types before and after the change, different amounts of information are retained — if a *some-any-order* plan is changed into a *some-together* plan, for example, not only the sub-plans persist, but also the information, which sub-plans where part of the continuation-condition.

This is something that cannot be found in many programs, but that is very important for actual work (another example of minimum commitment [Stevenson et al., 1996]).

6.5 Editing Time Annotations

For editing time annotations, a simple editor was written. Editing time annotations in-place is difficult because of the small size of the glyphs and the "strange" resolution of the screen. Depending on the selected time scale, one pixel might correspond to one second, but it also might correspond to 17.319 seconds. Another problem is that the MinDu and MaxDu bars are not aligned with any time shift, but are centered in the glyph. Thus, it is difficult to read the time span that is covered by that bar.

So the editor shown in Figure 6.2 was implemented that allows the specification of all the parts of a time annotation except the reference point. By clicking on the check-boxes, a part of a time annotation can be specified as undefined.

6.6 Renaming Plans

At the moment, a plan's name can only be changed by editing the name in a separate text field on the right of the view (see Figure 5.10, top right). This is very inconvenient, and not a reasonable solution for actual work.

6.7 Choosing a Time Scale

The time scale of the Temporal View can be chosen by selecting a time resolution from the drop-down list under the view. The resolutions available are natural resolutions, i.e., whole time units like seconds, minutes, etc.

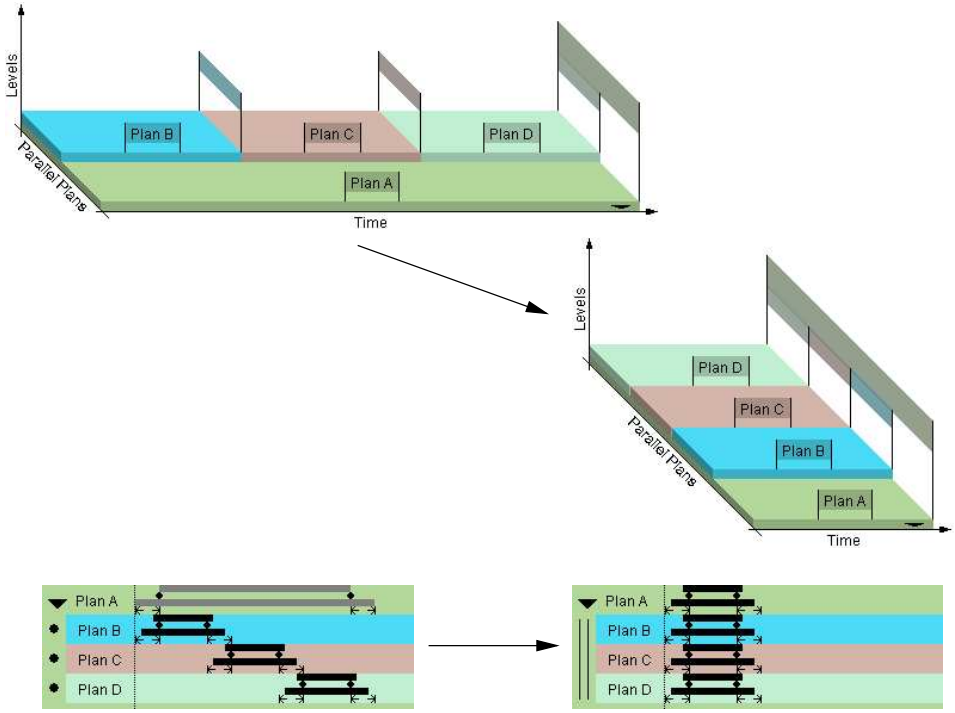


Figure 6.1: Changing a plan's type: A sequential plan is changed into a parallel plan, retaining all its attributes except the type.

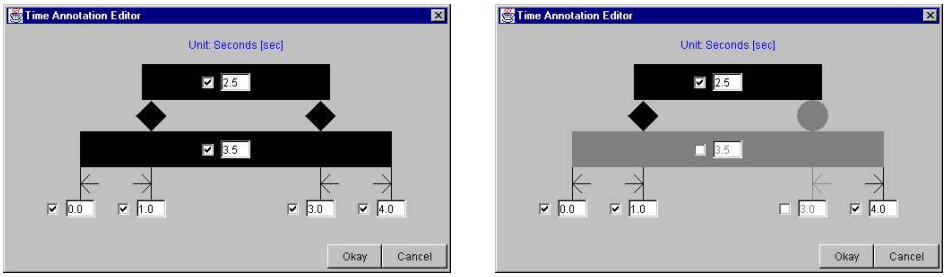


Figure 6.2: Time Annotation Editor.

Chapter 7

Program Design

At first they were no more than pale gossamer-threads,
so fine that they only twinkled fitfully where the Moon caught them,
but steadily they grew broader and clearer, until their design could be seen.
(J. R. R. Tolkien, *The Lord of The Rings*)

This chapter presents some of the key design concepts of the prototype implementation of AsbruView. Because we consider the concepts more important than the implementation and other technical issues, a complete description of the program design has not been included.

7.1 Model-View-Controller

The program consists of three main components: The model, a controller, and any number of views (see Figure 7.1).

The model contains the raw data, i.e. the representation of the plans themselves. It can also perform actions such as inserting or deleting a plan. Listeners can be attached to many properties of the model that are notified when the corresponding property changes.

The controller provides a basic user interface for certain actions, and also handles all changes made to the model. No changes should be applied by the views, but only by the controller. The controller can notify views of changes in the model that are on a higher level than a single plan.

Views display the model and accept user inputs that they convert into method calls to the controller.

Views register callback functions with the controller and with the model. So when there are any changes, the views are notified and can react.

Communication between views and controller is performed through interfaces. A controller can have any number of views attached, but a view can only belong to one controller.

7.1.1 Listeners

The data represented in the model is used by views as well as by the controller. The data can be changed by any of its users (in theory), so a mechanism is needed to notify the users of changes.

This is done with listeners that are attached to the data. A listener is a class that implements a certain interface, by means of which a method is called to notify the class of changes in the model. Typically, not only an identification of the changed value is passed, but also its old and new values.

Thus, all views can react to changes without any explicit redraw-requests from the controller. Different views may react in different ways to changes in the data (only partial redraws, or now change at all if the changed data is not visible).

Listeners are a concept that is heavily used in Java's AWT and Swing libraries.

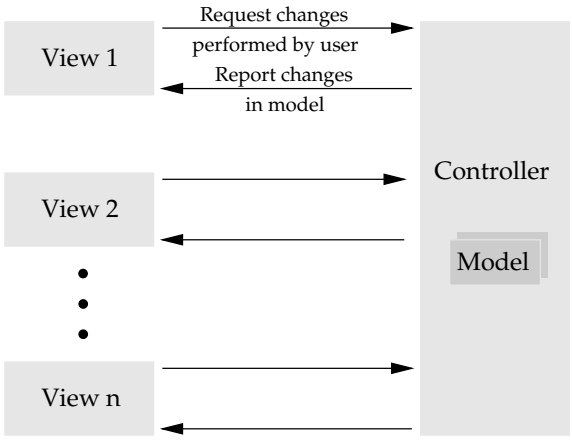


Figure 7.1: AsbruView's principal architecture: A controller containing a model and communicating with a number of views.

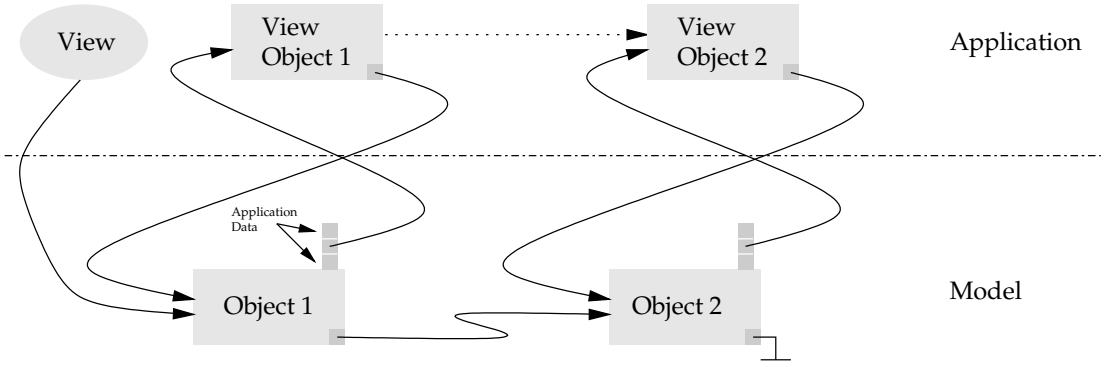


Figure 7.2: An example of using Application Data. The dotted link does not really exist in the application, but can be reconstructed from the structure of the underlying model. Thus, the application does not have to mirror and maintain the structure of the model

7.1.2 Application Data

Any user of a part of the model may need to associate additional information with it. The user, however, may not want to store that information with some connection to the model, so it does not have to mirror — and maintain — the connections between objects in the model (for an example, see Figure 7.2).

The mechanism implemented in AsbruView is quite simple. Any view or other entity can request an application data index. This index is used for storing and retrieving data by parts of the entity. In the example of AsbruView, each view has its own index. Such an index cannot be 'given back', however, but remains valid for the whole runtime of the application.

An application can store one object (of type `java.lang.Object`) in the application data array of every object in the model. If it needs more than one object, it can store a list or other container there.

The application data arrays grow dynamically, so that as little space as possible is wasted, while no restrictions are put on the number of concurrent users of the model.

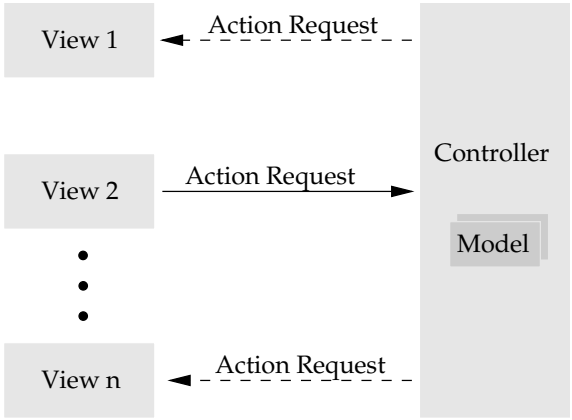


Figure 7.3: Action Request: The request is passed to all views except the one which submitted it.

7.1.3 Interfaces

Not only listeners use interfaces, but they are also used for communication between a controller and its views.

These interfaces define methods that are used to attach or detach a view to or from a controller; and also contain methods to perform actions like adding a new plan, moving a plan, or transmitting a change that only views of the data are interested in, but not the controller (see next section).

7.1.4 Action Request

Any changes the user wants to make to the data is done with graphical objects in one of the views, which tell the controller which changes to perform. The controller applies these changes to the model, which notify the views, which in turn adjust their representation.

But there are changes the user can make that are not part of the underlying model, but that only exist for the graphical representation. Some of these changes should be propagated to other views, like change of plan color or opening or closing of a plan.

For this purpose, a simple mechanism called an *action request* is used. A call to the corresponding function of a controller will call the change request function of all attached views except the one initiating the call. The controller does not use the contents of the message for any changes of the model, but only passes it on to the other views (see Figure 7.3).

7.2 System Requirements

These requirements are not so much requirements in the sense of other programs, but rather requirements in terms of computing power.

The prototype makes heavy use of the possibilities of the Swing classes of Java 1.2 (aka Java Platform 2). Because this library implements all its drawing in pure Java, it is inherently slow (even when using a just-in-time compiler (JIT)). Especially drawing on a canvas when the points are transformed first (skewed and stretched, for example, for the traffic signs) or objects are semi-transparent is very slow and leads to long delays between a user action (mouse move) and the reaction.

A number of buffers are used to decrease the need of complex redraws as much as possible.

So not only a fast computer is needed, but also one with sufficient resources in terms of memory and memory throughput between main memory and the graphics subsystem.

Part III

Evaluation and Conclusion

Chapter 8

Evaluation

However, most of the studies reported
have more authors than patients treated.
(from [Andersen, 1989])

This chapter reports the results of the evaluation we did to assess the usability of the user interface presented in this thesis.

Before implementing the prototype, we did some scenario-based evaluations [Carroll, 1995] using paper mock-ups [Lauesen et al., 1995] of the screen designs that we had developed. While this is considered good software engineering style, it cannot be called an evaluation — but it helped us decide which approaches were better suited, and also to make some of Asbru's ideas more understandable to the physicians we work with.

8.1 Questions to be Answered

The evaluation was meant to answer the following questions.

- Do physicians understand the meaning of the metaphors involved?
- Is the depiction of time annotations easy to understand, and do physicians understand the effects of undefined parts?
- Does the depiction of plans make understanding their temporal relation easier?
- Do physicians understand the underlying concepts of Asbru, and are they able to apply them?

8.2 Sample

All the participating persons (four male, two female) are practicing physicians in a number of different fields (intensive care of newborn infants, intensive care, psychiatry, paediatry).

Only one of the participants knows a programming language at all; this person does not only know several languages, but also is an able spare-time programmer. All but one participant use computers for their work, and all but two have computers at home.

All the participants are familiar with word processors, and all but one have used spread sheet software. Only one has experience with project planning software, and none has ever used a ray tracer or a VR browser.

Half of the participants use clinical protocols in their daily work. These are written down in plain text or drawn as flow-charts. Those that do not use protocols expect an increase in treatment quality from using them.

Considering the size of the sample, it is rather representative in terms of age distribution. It is, however, not clear whether it is representative in terms of familiarity with computers and the use of clinical protocols.

8.3 Evaluation Method

Evaluation was done separately with every participant. He or she was told what to expect, and then asked to fill out a short questionnaire (see Appendix A) about his or her basic computer skills. Then, an introduction to the ideas of Asbru and how they were represented in AsbruView was given. This usually lasted about 45 minutes to one hour. After this, the test person was asked to try to author a simple plan on his or her own. At the end, the participant was asked to fill out another questionnaire about his or her impression of the system (Appendix A). Tests usually lasted about two hours.

Two different test systems were used. One Intel Celeron 333 MHz PC with 128 MB RAM and a 1024x768 LCD Display, and one Intel Pentium II 266 MHz portable PC with 64 MB RAM and a 1024x768 LCD Panel. A separate mouse was used in both configurations. None of the LCD panels is perfect: The non-portable display is small and lacks color resolution, whereas with the portable one color is highly dependent of the viewing angle.

The speed of these systems also was less than what the prototype required (because of the use of Java, it was rather demanding) — which made using the program difficult at times.

8.4 Conceptual Findings

8.4.1 Topology View

All participants found the metaphors easy to understand, and remembered the different plan types after only one explanation.

Most participants said they could understand the topology of a plan from the graphical depiction.

Hierarchical decomposition and the depiction of optional and cyclical plans was found easy to understand by all participants.

Manipulation of plans was judged good, okay or bad by equal amounts of participants. Manipulation suffered very much from the lack of speed of the implementation.

The overall impression of most participants was good, only one participant judged it okay.

8.4.2 Temporal View

Most participants found the time annotation glyph easy to understand, and did not have any problems understanding what the effects of different undefined part of a time annotation would be.

Temporal uncertainty was found easy to understand in this view.

All participants found the hierarchical decomposition and the depiction of optional and cyclical plans easy to understand. Many, however, did not understand the word *tree view* in the questionnaire. This happened even in the later tests where this name was stressed in the introduction of the concepts of AsbruView.

The overall impression of this view was good or okay.

8.4.3 Overall Impression

All participants found the program rather usable and easy to understand.

All but one participant said they could imagine using the program for their daily work, and considered the program usable. All participants were of the opinion that the program would be better suited for work with clinical protocols.

The use of color was considered helpful by all participants, even though some remarked that colors might be misleading. The use of grey in contrast for undefined components was also understood by all.

Even the meaning of reuse and how changes in a plan affect all its other instances were understood, which is rather surprising. It is well known that a similar mechanism in word processors (called templates there) is hardly used by casual users.

One conceptual problem that became apparent during the evaluation is the fact that a plan defines the way its sub-plans are to be executed, and that a plan can only have one type — so it is not possible, for example, to have one sub-plan be executed first, and then afterwards two sub-plans in parallel. In such a case an intermediate parallel plan would be needed that would contain the latter two plans. But this means that sometimes artificial plans must be inserted that do not have any meaning for physicians.

This is a clear consequence of the fact that Asbru was designed by computer scientists, but AsbruView should try to bridge the gap here. One possible solution would be to allow any plan layouts, and to insert invisible artificial plans where necessary in order to comply with the rules of the language.

Several participants also remarked that there are many factors in medical decision making that cannot be quantified, and so are not accessible to a computer (this differs, of course, between different domains). This criticism does not only apply to AsbruView, but Asbru in general. Yet, it is based on a misunderstanding: Asbru can work with qualitative values as well as quantitative ones. Asbru is not supposed to work in a closed-loop system, so there is no reason why a physician or a nurse should not be able to report a condition such as “patient is dyspnoeic” to the system to help it decide which plan to propose.

8.5 Technical Findings

In the current implementation of the prototype, users cannot change a plan’s name where it is displayed, but rather have to use an input field on the right side of the window. This has proved very difficult to use, and has led to many mistakes. The name must be entered in a text field to the right (see Figure 5.10) — this, together with the long delay between entering a new name and its display on the screen, makes the connection between the plan and its name in this text field difficult to understand.

Navigation in three-dimensional space (as described in section 6.2) was accepted by most participants as quite usable, although they criticized the speed (the plan followed the mouse pointer only with a considerable delay that made navigation very difficult). All participants were able to move and place plans after only a few unsuccessful tries.

All users liked the fact that they could change a plan’s type at any time. Interestingly, most of them assumed that this would not be possible. Being able to change a plan’s type means less stress when authoring, because changes can be easily made.

All participants criticized the speed (or lack thereof) of the system — this was mostly due to its implementation in Java, but also because the test environment was not always perfect.

There was criticism from one participant (who knows a number of programming languages, and has done some programming himself) that the Temporal View could have been done using standard Windows controls, rather than the non-standard way it was actually implemented.

It should be possible to copy time annotations between plans or parts of plans. Often, similar patterns are needed, and in this case, it is very frustrating to enter the same or similar information over and over again (this also applies to conditions, which are often needed in more than one plan, or in different conditions of the same plan).

The time annotation editor should enable the user to enter the unit for every value separately — this is possible in Asbru, but was not implemented this way in the prototype.

An undo/redo function should be available. This was not implemented in the prototype, but proved to be quite important to enable users to try something out without losing their work, and also in case they made a mistake.

Another simple thing that was left out was a confirmation dialog that would ask if the user really wished to delete a plan when a plan with many sub-plans was to be deleted. For daily work, such dialogs can be very annoying, when they appear every time one wants to delete a file (or a plan). But if that function would only ask if the plan to be deleted had sub-plans or had many defined conditions, that would decrease user frustration.

Even though colors were generally considered a very important part of the interface, the way they are used at the moment must be questioned. Colors should reflect a parameter, like the level of a plan, or its relationship with other plans (especially with reused plans).

There are two "new" functions: New in the File menu, and the New button. The former deletes all plans, and presents the user with a blank workspace; the latter creates a new plan on top of the currently selected plan. Some users were confused by two different functions of the same name, which slowed down their understanding of the other functions considerably. While it is easy to change the name on a button, the impact of a bad name can be huge. The caption of the "Change" button was also criticized, because it does not say what it changes (it changes the type).

In the current implementation, it is not possible to create a new plan "underneath" the current root plan. Instead, a new plan can be created as a sub-plan of the current root plan, and all the sub-plans and other attributes can be copied to that new plan. Now the old root plan can be renamed and changed to act as the new root plan. This is awkward, so a way of inserting a new root plan will have to be found. But this is a nice example of how the architecture of the underlying model can influence the user interface.

One participant suggested that a context menu be available for actions to take on plans, conditions, etc., that would only show the actions available for that particular object. While this is easy in the temporal view, it is difficult in topological view: A menu popping up in the middle of a three-dimensional depiction can completely destroy the impression of a small world. On the other hand, users expect the right mouse button or a double click to have an effect.

8.6 Discussion

The overall rather positive rating of the prototype is surprising considering the fact that the participants had to endure a 45 minute lecture on the intrinsics of Asbru. But even a graphical user interface requires a certain amount of knowledge (see section 9.1) to work with it.

The participants were all either working with clinical protocols or at least expected improvements to their work from using them. When introduced to the basics of Asbru(View), they very quickly understood how they could apply its methods to their own domain, and how they could benefit from them.

Even those that criticized AsbruView were clearly in favor of a powerful system for working with clinical protocols. This means, we are on the right track.

None of the participants seemed to believe that it was possible to change a plan's type at any time. Most of them asked during their own first tries if it really was possible to do that. This seems to be an important feature, which people miss in other programs, even though they might not be aware of it.

Most of the technical problems that were uncovered during the evaluation, as well as some other bugs, will be eliminated in the coming months. The key problem, however, is speed. This will also be addressed before any new features are added.

Chapter 9

Conclusion

Books ought to have good endings. How would this do: and they all settled down and lived together happily ever after*?*
(J. R. R. Tolkien, *The Lord of The Rings*)

This last section of this thesis contains a conclusion, which does not only sum up the findings of this thesis, but also presents some thoughts that lead further.

9.1 Reflections on User Interfaces

There are some lessons to be learned from the way AsbruView has turned out to work.

The first lesson is that speed is paramount. A user interface must react in real-time, otherwise it is unusable. This becomes more important the more heavily the interface uses three dimensional objects: navigation and manipulation must be fast enough to follow any action of the user, or she will be hindered more than helped, and ultimately refuse to work with the system.

The second lesson is that visualization and user interfaces are very closely related fields when it comes to user interfaces for complex tasks. The two different problems of depicting complicated structures and of manipulating them must be developed together in order to end up with a usable system, that not just presents the data well, but then requires the user to use an entirely different system to manipulate the data.

The third lesson is that a user interface cannot replace knowledge — knowledge only the user can have. The following quotation was taken from [Shneiderman, 1997a]:

> An expert computer user who has not studied architecture will not be able to use a building-design package any more than a computer-savvy amateur can make reliable medical diagnoses.

Or, using examples that more users are familiar with: A word processor does not make everybody a writer any more than a paint program will make anyone a painter. This is a mistake that is quite often made, the more powerful software becomes. But all a computer can do at the moment is relieve the user of learning a craft, but never of learning an art (and often not even of the finer points of a craft).

Fourth lesson: the user interface must be able to hide the underlying structure of the program to make it usable by people with different thought patterns than computer scientists. An example of such a concept that should be hidden is that Asbru plans can only have one type, and hence only one plan layout. This is hard to understand, and should be hidden by the user interface. Such changes may require some changes in the way data is handled, and even in other parts of a system, not only the user interface.

Another example is a waiting period: Computer scientists would simply define a plan *wait*, but that is not logical for anybody else. Waiting is not perceived as a task by physicians, and thus should not be required to be defined as one.

The fifth lesson is about using standard controls in user interfaces. In what I consider a landmark article, [Gentner and Nielson, 1996] make a strong case against direct manipulation. Indeed, direct manipulation limits the user in many ways, especially when it comes to handling lengthy tasks with many objects that are to be processed in a similar way. This is why scripting languages and text-based user interfaces still exist (and indeed prosper) in a world of visual interfaces. But especially for expert users and members of the "post-Nintendo Generation" (as [Gentner and Nielson, 1996] call people that have grown up with computers), more powerful tools are needed that neither are restricted in such a manner, nor require the user to go back to the command line.

One participant in the test asked why AsbruView does not use standard controls, like tree views as known from file managers, or other standard interface techniques (like multiple documents). The answer to this question is that interface standards were designed for standard applications, which AsbruView is not. A number of de facto standards have emerged in the design of office packages that make many of the existing packages almost indistinguishable. While this makes life easier for people who know one of those packages and have to use another, it also makes any progress or innovation impossible. This would be fine if these designs were perfect and known to be perfect. But they are not — and we know that.

So even in the field of standard applications one must ask if existing standards really are so useful after all, or if they only hinder development towards usable interfaces.

So the design presented in this thesis can be understood partly as a different approach to user interfaces than is usually done in software engineering (where the user interface still plays a minor role in many projects).

The last lesson (at least the last presented here) is that it is very important to consequently implement direct manipulation, including the possibility to change the type of an object. This has proven a feature that was very easy to implement (if included in the design — it is probably very hard to add afterwards) and has made work with the system a lot easier and less frustrating (when a decision had to be revised).

This is another aspect that should be seen from the user's point of view, rather than from the program designer's. Programmers may see objects as having a type from their creation to their disposal, but users are more accustomed to objects being able to change.

Appendix A

Questionnaires

A.1 Description

The questionnaire shown on the next few pages was used for the evaluation of AsbruView. It consists of two parts: One to be filled out before the test, and one after the test. Ideas for this questionnaire were taken from [Shneiderman, 1997a].

The idea of this questionnaire was not only to get a comparable set of ratings of parts of the program, but also to have a basis for further questions during and after the evaluation sessions. The first questionnaire helped us in deciding what we had to tell the participant, in order not to bore him or her with concepts they already knew, but also to tell them all they needed to know.

After the second questionnaire was filled out, we asked a few further questions depending on the answers given, to find out more details about which parts were considered particularly good or bad.

The original (German) version of the questionnaires (Figures A.1, A.2, and A.3) was used for the actual evaluation.

This English translation (Figures A.4, A.5, and A.6) is only meant as a first approximation of a proper English version of the questionnaire. This translation has not been used in evaluation.

Fragebogen zur Evaluierung von AsbruView

Fragen *vor* dem Test

Wie würden Sie Ihre Computerkenntnisse einstufen?	☐ Anfänger ☐ Fortgeschritten ☐ Profi	
Verwenden Sie beruflich einen Computer?	☐ Ja	☐ Nein
Verwenden Sie privat einen Computer?	☐ Ja	☐ Nein
Sind Sie mit der Verwendung einer Maus vertraut?	☐ Ja	☐ Nein
Sind Sie mit Drag & Drop vertraut?	☐ Ja	☐ Nein
Mit welchen der folgenden Systeme sind Sie vertraut?		
Textverarbeitung	☐ Ja	☐ Nein
Tabellenkalkulation	☐ Ja	☐ Nein
Projektplaner	☐ Ja	☐ Nein
objektorientiertes Zeichenprogramm (CorelDraw, PowerPoint)	☐ Ja	☐ Nein
Windows Explorer	☐ Ja	☐ Nein
Ray Tracing Software	☐ Ja	☐ Nein
VR Browser	☐ Ja	☐ Nein
Beherrschen Sie eine Programmiersprache?	☐ Ja	☐ Nein
Wenn ja: Welche?		
Verwenden Sie klinische Protokolle bei Ihrer Arbeit?	☐ Ja	☐ Nein
Wenn ja: Wie werden diese bisher dargestellt? Sind Sie mit diesen Darstellungen zufrieden?	☐ Ja	☐ Nein
Wenn nein: Erwarten Sie sich Verbesserungen durch die Verwendung von klinischen Protokollen?	☐ Ja	☐ Nein

Name des Testers/der Testerin:
Datum:
Verwendeter Rechner (CPU/Takt/RAM/Bildschirm/Auflösung):

Figure A.1: Evaluation Questionnaire. German original, questions before the test.

Fragebogen zur Evaluierung von AsbruView

Fragen *nach* dem Test

Allgemeiner Eindruck		
Wie ist Ihr allgemeiner Eindruck vom System?	☐ Gut ☐ Mittel ☐ Schlecht	
Wie beurteilen Sie die Geschwindigkeit des Systems?	☐ Gut ☐ Mittel ☐ Schlecht	
Wie beurteilen Sie die Bedienung des Programms?	☐ Praktisch ☐ Unpraktisch	
Hilft oder stört die Verwendung von Farben bei der Arbeit?	☐ Hilft ☐ Egal ☐ Stört	
Ist die Verwendung von Grau für undefinierte Conditions/Time Annotations verständlich?	☐ Ja	☐ Nein
Ist der Zusammenhang zwischen den Ansichten verständlich?	☐ Ja	☐ Nein
Ist die Bedeutung von Reuse verständlich?	☐ Ja	☐ Nein
Topologieansicht		
Sind die Metaphern für Pläne einleuchtend?	☐ Ja	☐ Nein
Halten Sie die Metaphern für zu verspielt?	☐ Ja	☐ Nein
Können Sie sich aufgrund der graphischen Darstellung die zeitlichen Zusammenhänge zwischen den Plänen vorstellen?	☐ Ja	☐ Nein
Ist die Unterteilung in Subpläne verständlich?	☐ Ja	☐ Nein
Sind optionale Pläne verständlich dargestellt?	☐ Ja	☐ Nein
Sind zyklische Pläne verständlich dargestellt?	☐ Ja	☐ Nein
Wie beurteilen Sie die Manipulation von Plänen?	☐ Gut ☐ Brauchbar ☐ Schlecht	
Wie beurteilen Sie die Ansicht insgesamt?	☐ Gut ☐ Brauchbar ☐ Schlecht	
Zeitansicht		
Sind die Auswirkungen undefinierter Teile von Time Annotations verständlich?	☐ Ja	☐ Nein
Sind die zeitlichen Unsicherheiten verständlich?	☐ Ja	☐ Nein
Erhöht die Baumansicht das Verständnis für die hierarchische Unterteilung der Pläne?	☐ Ja	☐ Nein
Ist die Unterteilung in Subpläne verständlich?	☐ Ja	☐ Nein
Sind optionale Pläne verständlich dargestellt?	☐ Ja	☐ Nein
Sind zyklische Pläne verständlich dargestellt?	☐ Ja	☐ Nein
Wie beurteilen Sie die Manipulation von Plänen?	☐ Gut ☐ Brauchbar ☐ Schlecht	

Figure A.2: Evaluation Questionnaire. German original, questions after the test, first page.

Wie beurteilen Sie die Ansicht insgesamt?	☐ Gut ☐ Brauchbar ☐ Schlecht	
Verwendbarkeit		
Halten Sie das System insgesamt für die tägliche Arbeit für brauchbar?	☐ Ja	☐ Nein
Könnten Sie sich vorstellen, Ihre Protokolle mit AsbruView zu schreiben?	☐ Ja	☐ Nein
Würde es Ihre Arbeit gegenüber anderen Formen (Text, Ablaufdiagramme, etc.) erleichtern?	☐ Ja	☐ Nein

Figure A.3: Evaluation Questionnaire. German original, questions after the test, second page.

Questionnaire

Questions *before* the Test

How would you rate your computer skills?	☐ Beginner ☐ Intermediate ☐ Professional
Do you use a computer for work?	☐ Yes ☐ No
Do you use a computer at home?	☐ Yes ☐ No
Do you know how to use a mouse?	☐ Yes ☐ No
Do you know *Drag and Drop*?	☐ Yes ☐ No
Which of the following systems are you familiar with?	
Word Processor	☐ Yes ☐ No
Spread Sheet	☐ Yes ☐ No
Project Planner	☐ Yes ☐ No
Object-oriented Drawing Program (CorelDraw, PowerPoint)	☐ Yes ☐ No
Windows Explorer	☐ Yes ☐ No
Ray Tracing Software	☐ Yes ☐ No
VR Browser	☐ Yes ☐ No
Do you know a programming language?	☐ Yes ☐ No
If Yes: Which?	
Do you use clinical protocols for your work?	☐ Yes ☐ No
If Yes: How are they represented?	
Are these representations satisfactory?	☐ Yes ☐ No
If No: Do you expect an improvement in treatment from the use of clinical protocols?	☐ Yes ☐ No

Name of Participant:
Date:
Computer (CPU/Clock/RAM/Screen/Resolution):

Figure A.4: Evaluation Questionnaire. English translation, questions before the test.

Questionnaire

Questions *after* the Test

Overall Impression		
What is your overall impression of the system?	☐ Good ☐ Okay ☐ Bad	
How is the speed of the program?	☐ Good ☐ Okay ☐ Bad	
Is the program practical or impractical to work with?	☐ Practical ☐ Impractical	
Does the use of color help or hinder work?	☐ Helps ☐ No effect ☐ Hinders	
Is the use of gray for undefined conditions/time annotations easy to understand?	☐ Yes	☐ No
Is the connection between the two views easy to understand?	☐ Yes	☐ No
Is the meaning of reuse easy to understand?	☐ Yes	☐ No
Topology View		
Are the metaphors used for plans clear?	☐ Yes	☐ No
Are the metaphors too playful?	☐ Yes	☐ No
Can you imagine the temporal order of plans from the graphical depiction?	☐ Yes	☐ No
Is the decomposition into sub-plans clear?	☐ Yes	☐ No
Is the depiction of optional plans easy to understand?	☐ Yes	☐ No
Is the depiction of cyclical plans easy to understand?	☐ Yes	☐ No
How is the manipulation of plans?	☐ Good ☐ Okay ☐ Bad	
What is your overall impression of the view?	☐ Good ☐ Okay ☐ Bad	
Temporal View		
Is it clear how undefined components influence the behavior of a time annotation?	☐ Yes	☐ No
Are the temporal uncertainties easy to understand?	☐ Yes	☐ No
Does the tree view make the hierarchical decomposition of plans easier to understand?	☐ Yes	☐ No
Is the decomposition into sub-plans easy to understand?	☐ Yes	☐ No
Is the depiction of optional plans easy to understand?	☐ Yes	☐ No
Is the depiction of cyclical plans easy to understand?	☐ Yes	☐ No
How is the manipulation of plans?	☐ Good ☐ Okay ☐ Bad	

Figure A.5: Evaluation Questionnaire. English translation, questions after the test, first page.

What is your overall impression of the view?	☐ Good ☐ Okay ☐ Bad	
Usability		
Do you believe that the system is usable for every-day work?	☐ Yes	☐ No
Could you imagine using AsbruView to author clinical protocols?	☐ Yes	☐ No
Would it make your work easier compared with other representations (free text, flow-charts, etc.)?	☐ Yes	☐ No

Figure A.6: Evaluation Questionnaire. English translation, questions after the test, second page.

Bibliography

[Andersen, 1989] Jens B. Andersen (1989). Ventilatory Strategy in Catastrophic Lung Disease. Inversed Ratio Ventilation (IRV) and Combined High Frequency Ventilation (CHFV). *Acta Anaesthesiol Scand*, 33:145–148.

[Brandt et al., 1997] C. A. Brandt, S. J. Frawley, S. M. Powsner, R. N. Shiffman, and P. L. Miller (1997). Visualizing the Logic of a Clinical Guideline: A Case Study in Childhood Immunization. *Methods of Information in Medicine*, 36:179–83.

[Carroll, 1995] John M. Carroll, editor (1995). *Scenario-Based Design — Envisioning Work and Technology in System Design*. John Wiley & Sons, New York.

[Chernoff, 1973] Herman Chernoff (1973). The use of faces to represent points in k-dimensional space graphically. *Journal of the American Statistical Association*, 68:361–368.

[Chuah and Eick, 1997] Mei C. Chuah and Stephen G. Eick (1997). Glyphs for Software Visualization. In *5th International Workshop on Program Comprehension (IWPC '97) Proceedings*, pages 183–191. IEEE Computer Society Press, Dearborn, Michigan.

[Cole and Stewart, 1994] William G. Cole and James G. Stewart (1994). Human Performance Evaluation of a Metaphor Graphic Display for Respiratory Data. *Methods of Information in Medicine*, 33:390–396.

[Duftschmid, 1999] Georg Duftschmid (1999). *Knowledge-based Verification of Clinical Guidelines by Detection of Anomalies*. PhD thesis, Vienna University of Technology.

[Dupré, 1998] Lyn Dupré (1998). *BUGS in Writing*. Addison Wesley Longman Inc.

[Friedland and Iwasaki, 1985] Peter E. Friedland and Yumi Iwasaki (1985). The Concept and Implementaion of Skeletal Plans. *Journal of Automated Reasoning*, 1(2):161–208.

[Furnas, 1981] George W. Furnas (1981). The FISHEYE View: A New Look at Structured Files. Technical Memorandum #81-11221-9, Bell Laboratories, Murray Hill, New Jersey 07974, U.S.A.

[Gentner and Nielson, 1996] Don Gentner and Jakob Nielson (1996). The Anti-Mac Interface. *Communications of the ACM*, 39(8):70–82.

[Goldstine and von Neumann, 1947] Herman H. Goldstine and John von Neumann (1947). Planning and Coding Problems for an Electronic Computing Instrument. Part II, vol. 1, U.S. Army Ordinance Department. reprinted in von Neumann, J., 1963. Collected Works Vol. V. New York: McMillan, pp. 80-151.

[Gross et al., 1997] M. H. Gross, T. C. Sprenger, and J. Finger (1997). Visualizing Information on a Sphere. In *Proceedings of the 1997 Conference on Information Visualization*.

[Inselberg, 1997] Alfred Inselberg (1997). Multidimensional Detective. In *Proceedings of the 1997 Conference on Information Visualization*, pages 100–107.

[Inselberg and Dimsdale, 1987] Alfred Inselberg and Bernard Dimsdale (1987). Parallel coordinates for visualizing multi-dimensional geometry. In Tsiyasu L. Kunii, editor, *Computer Graphics 1987 (Proceedings of CG International '87)*, pages 25–44. Springer-Verlag.

[Kosara and Miksch, 1999] Robert Kosara and Silvia Miksch (1999). Visualization Techniques for Time-Oriented, Skeletal Plans in Medical Therapy Planning. In Werner Horn, Yuval Shahar, Greger Lindberg, Steen Andreassen, and Jeremy Wyatt, editors, *Artificial Intelligence in Medicine: Proceedings of the Joint European Conference on Artificial Intelligence in Medicine and Medical Decision Making (AIMDM'99)*, Berlin. Springer. Forthcoming.

[Kosara et al., 1998] Robert Kosara, Silvia Miksch, Yuval Shahar, and Peter Johnson (1998). AsbruView: Capturing Complex, Time-oriented Plans — Beyond Flow-Charts. In *Thinking with Diagrams 98*.

[Lauesen et al., 1995] Søren Lauesen, Susanne Salbo, and Ann Thomsen (1995). Usefulness of Paper Mockups. In *Proceedings of OZCHI'95, the CHISIG Annual Conference on Human-Computer Interaction*, Full Papers, pages 82–87.

[MacEachren, 1992] Alan M. MacEachren (1992). Visualizing Uncertain Information. *Cartographic Perspective*, 13:10–19.

[Mackinlay et al., 1991] Jock D. Mackinlay, George G. Robertson, and Stuart K. Card (1991). The Perspective Wall: Detail and Context Smoothly Integrated. In *Proceedings of ACM CHI'91 Conference on Human Factors in Computing Systems*, Information Visualization, pages 173–179.

[Martin, 1973] Johannes J. Martin (1973). The 'Natural' Set of Basic Control Structures. *SIGPLAN Notices*, 8(12):5–14.

[McKinney et al., 1998] Kathleen McKinney, Martin Fischer, and John Kunz (1998). Visualization of Construction Planning Information. In *Proceedings of the 1998 International Conference on Intelligent User Interfaces*, Intelligent Database Interfaces, pages 135–138.

[Miksch et al., 1998] Silvia Miksch, Robert Kosara, Yuval Shahar, and Peter Johnson (1998). AsbruView: Visualization of Time-Oriented, Skeletal Plans. In *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems 1998 (AIPS-98)*. Carnegie Mellon University, AAAI Press.

[Miksch et al., 1997a] Silvia Miksch, Yuval Shahar, Werner Horn, Christian Popow, Franz Paky, and Peter Johnson (1997a). Time-Oriented Skeletal Plans: Support to Design and Execution. In *Fourth European Conference on Planning (ECP'97)*. Springer.

[Miksch et al., 1997b] Silvia Miksch, Yuval Shahar, and Peter Johnson (1997b). Asbru: A Task-Specific, Intention-Based, and Time-Oriented Language for Representing Skeletal Plans. In *Proceedings of the 7th Workshop on Knowledge Engineering: Methods & Languages (KEML-97)*. Milton Keynes, UK, Open University.

[Mukherjea et al., 1996] Sougata Mukherjea, Kyoji Hirata, and Yoshinori Hara (1996). Visualizing the Results of Multimedia Web Search Engines. In *Proceedings IEEE Symposium on Information Visualization 1996*, pages 64–65.

[Mullet and Sano, 1995] Kevin Mullet and Darrell Sano (1995). *Designing Visual Interfaces*. Sunsoft Press (Prentice Hall).

[Musen et al., 1995] Mark A. Musen, John H. Gennari, Henrik Eriksson, Samson W. Tu, and Aangel R. Puerta (1995). PROTÉGÉ-II: A Computer Support for Development of Intelligent Systems from Libraries of Components. In *Proceedings of the 8th World Congress on Medical Informatics (MEDINFO-95)*, pages 766–770.

[Nassi and Shneiderman, 1973] Isaac Nassi and Ben Shneiderman (1973). Flowchart Techniques for Structured Programming. *SIGPLAN Notices*, 8(8):12–26.

[Ortony, 1993] Andrew Ortony, editor (1993). *Metaphor and Thought*. Cambridge University Press, Cambridge, 2nd edition.

[Pang et al., 1996] Alex Pang, Craig Wittenbrink, and Suresh Lodha (1996). Approaches to Uncertainty Visualization. Technical Report UCSC-CRL-96-21, University of California, Santa Cruz, Jack Baskin School of Engineering.

[Plaisant et al., 1996] Catherine Plaisant, Brett Milash, Anne Rose, Seth Widoff, and Ben Shneiderman (1996). LifeLines: Visualizing Personal Histories. In *Proceedings of ACM CHI 96 Conference on Human Factors in Computing Systems*, volume 1 of *PAPERS: Interactive Information Retrieval*, pages 221–227.

[Plaisant et al., 1998] Catherine Plaisant, Richard Mushlin, Aaron Snyder, Jia Li, Dan Heller, and Ben Shneiderman (1998). LifeLines: Using Visualization to Enhance Navigation and Analysis of Patient Records. In *Proceedings of the 1998 American Medical Informatic Association Annual Fall Symposium*, pages 76–80.

[Powsner and Tufte, 1994] Seth M. Powsner and Edward R. Tufte (1994). Graphical Summary of Patient Status. *The Lancet*, 344:386–389.

[Purgathofer and Löffelmann, 1997] Werner Purgathofer and Helwig Löffelmann (1997). Selected New Trends in Scientific Visualization. Technical Report TR-186-2-97-17, Vienna University of Technology, Computer Graphics, Visualisation and Animation Group.

[Rit, 1986] Jean-Francois Rit (1986). Propagating temporal constraints for scheduling. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 383–388. Morgan Kaufman Publishers, Inc.

[Sarkar et al., 1993] Manojit Sarkar, Scott S. Snibbe, Oren J. Tversky, and Steven P. Reiss (1993). Stretching the Rubber Sheet: A Metophor for Visualizing Large Layouts on Small Screens. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, Visualizing Information, pages 81–91.

[Shahar et al., 1998] Yuval Shahar, Silvia Miksch, and Peter Johnson (1998). The Asgaard Project: A Task-Specific Framework for the Application and Critiquing of Time-Oriented Clinical Guidelines. *Artificial Intelligence in Medicine*, 14:29–51.

[Shneiderman, 1992] Ben Shneiderman (1992). Tree Visualization with Tree-Maps: A 2-D Space-Filling Approach. *ACM Transactions on Graphics*, 11(1):92–99.

[Shneiderman, 1996] Ben Shneiderman (1996). The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In *Proceedings of the IEEE Symposium on Visual Languages*, pages 336–343, Washington. IEEE Computer Society Press.

[Shneiderman, 1997a] Ben Shneiderman (1997a). *Designing the User Interface: Strategies for Effective Human-Computer-Interaction*. Addison Wesley Longman, 3rd edition.

[Shneiderman, 1997b] Ben Shneiderman (1997b). Direct Manipulation for Comprehensible, Predictable and Controllable User Interfaces. In *Proceedings of the 1997 International Conference on Intelligent User Interfaces*, Debate: Direct Manipulation vs. Interface Agents, pages 33–39.

[Stevenson et al., 1996] David A. Stevenson, Xiaohong Guan, Kenneth J. MacGallum, and Alex H. B. Duffy (1996). Sketching on the back of the computational envelope ... and then posting it? In *Proceedings of the Workshop on Visual Representation, Reasoning and Interaction in Design, Fourth International Conference on Artificial Intelligence in Design 1996 (AID'96)*. Stanford University, USA.

[Tufte, 1983] Edward R. Tufte (1983). *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, CT.

[Tufte, 1990] Edward R. Tufte (1990). *Envisioning Information*. Graphics Press, Cheshire, CT.