AsbruView: Capturing Complex, Time-oriented Plans — Beyond Flow-Charts

Robert Kosara and **Silvia Miksch**
Vienna University of Technology, Institute of Software Technology
Resselgasse 3/188, A-1040 Vienna, Austria, Europe
rkosara@wvnet.at, silvia@ifs.tuwien.ac.at

Yuval Shahar and **Peter Johnson**[1]
Section on Medical Informatics, Medical School Office Building, x215
Stanford University, Stanford, CA 94 305 - 5479, USA
shahar@smi.stanford.edu, pete@mimir.demon.co.uk

Abstract

Flow-charts are one of the standard means of representing actions or algorithms in many domains. However, applying flow-charts in dynamically changing environments, like clinical treatment planning, reveals their limitations. Flow-charts do not include the temporal dimension in their design, do not allow complex paths through many components, and scale very badly. These are only some of the requirements for a means of communicating clinical therapy plans. As an alternative, a plan-representation language called **Asbru** was designed, that overcomes all the limitations of flow-charts. It is, however, impossible for a domain expert to work with Asbru directly. Therefore, a visualization is presented here, called **AsbruView**, which uses three-dimensional diagrams and metaphors — running tracks and traffic signs — to make the parts of Asbru easily understandable and usable. Even very complex clinical plans are easy to survey with AsbruView.

Introduction: Clinical Protocols

We are motivated by the demands in medical, real-world environments: improving the quality of health care through increased awareness of proper disease-management techniques. The Health Maintenance Organizations (HMOs) are urged to increase productivity and simultaneously reduce costs without adversely affecting the quality of patient care. One step towards this aim is the implementation of commonly accepted and standardized health care procedures. Treatment planning from scratch typically is not necessary, as general clinical procedures exist which should guide the medical staff. These procedures are called *clinical practice guidelines and protocols*. A guideline can be defined as *'a method, that identifies actions, that are to be performed and that specify conditions that govern when it is appropriate to perform them'* (Pattison-Gordon et al., 1996). A clinical protocol is a more detailed version of a clinical guideline and refers to a class of therapeutic interventions [2]. Such protocols are used for utilization review, for improving quality assurance, for reducing variation in clinical practice, for guiding data collection, for better interpretation and management of the patient's status, and for activating alerts and reminders, for improving decision support (Pattison-Gordon et al., 1996).

Overview

The first section sketches the problem area, namely the design of clinical therapy and treatment plans. Authoring of such plans is a non-trivial task; therefore, a means of representing such plans appropriately, a language called **Asbru**, will be introduced in the second section. Asbru, however, is impossible to use for people working in the domain of therapy planning, and thus a means of representing Asbru-plans in an understandable way had to be found. This visualization of Asbru is called **AsbruView**, and will be discussed in detail in the third section. Flow-charts are most commonly used in the medical domain. A short introduction to flow-charts will be given in the fourth section. Why we didn't choose flow-charts, however, will be made clear in the fifth, final, section. We will end up with the conclusion and future plans.

The Problem Area: Authoring Clinical Protocols

In most cases, physicians and other medical staff need not invent therapy or treatment plans for their patients anew every time. The medical staff can fall back on predefined protocols. Such plans are usually represented in free text, in decision tables, or in flow-charts. Authoring plans, however, is a non-trivial task. Part of its complexity stems from the inappropriate means to represent plans in order to communicate and reason about them.

Difficulties to author a treatment plan are on various levels. Problems are also caused by the different purposes the plans or protocols are used for. The structure and composition of a treatment plan are quite manifold. Many variables must be accounted for and many different conditions must be taken into consideration. However, the instructions (the overall plan) must still remain readable, understandable, and lucid. In a therapy plan, a goal needs to be achieved in a certain time. The way or path to this goal is not always obvious and can be achieved following various paths. It must even be possible to perform actions that are not part of the actions' set, but still follow the underlying intentions of a treatment plan. This means, physicians often do not adhere to protocols, believing their actions to be closer to the in-

[1] Currently, Peter Johnson is working at: The Sowerby Center for Primary Health Care Informatics, University of Newcastle, Newcastle upon Tyne, NE4 6BE, UK.

[2] In Computer Science, clinical protocols can be interpreted as plans. Therefore, we are using the expressions 'protocol' and 'plan' interchangeably.

tentions of the protocol design. Hence, a treatment plan needs to capture the intentions too, allowing to continue a particular plan even when the performed actions vary.

The plans' intentions can be used for different purposes, like critiquing. Does applying the plan really lead to fulfilling its intentions? Can the plan be applied under different conditions, and if, which are those? These questions are not only needed for the selection of the correct plan for a certain patient, but also for improving the plans, and thus the patient's treatment.

In medicine, new ways of solving problems are being discovered day by day, new side-effects are found, etc. So treatment plans change very often, as new treatments and new conditions are added, while others may be changed or even removed entirely. This leads to the problem, that after the development of a clinical plan has been finished, it may already be out of date.

Regarding these points, it is not surprising that clinical protocols are often vague, incomplete and sometimes even contradictory.

Representing Plans: Historical Synopsis

Clinical protocols or plans can be seen as procedures or algorithms, which need to be executed depending on health conditions of a patient and within a particular time span. An appropriate modelling language and visualization (diagrams) capturing all different features are needed.

On the one hand, in Computer Science (particularly in the research of Programming Languages and Software Engineering), different approaches were introduced to capture such procedures, known as algorithms or programs. It started in the early years of programming. Some milestones: in 1947 the first graphical representations of algorithms were designed, called flowcharts ((Goldstine and von Neumann, 1947)); after a long discussion about styles of programming (e.g., *goto*-less/*goto*-free programming), 'structured programming' and block structured diagrams were introduced (Nassi and Shneiderman, 1973); followed by different modelling techniques, such as petri-nets, graphical (visual) programming, and object-oriented programming (Goldberg and Rubin, 1995). Additionally, different computer-oriented knowledge interchange languages (e.g., KIF (Genesereth and Fikes, 1982)), ontologies (Guarina and Giaretta, 1995), and plan-representation languages (e.g., PROPEL language (Levinson, 1995)) were discussed to represent domain-specific procedural knowledge. In general, these representations have significant limitations and are not applicable in dynamically changing environments, like medical domains (e.g., they assume instantaneous actions and effects; they neglect that actions often are continuous (durative) and might have delayed effects and temporally-extended goals; they overlook that unobservable underlying processes determine the observable state of the world). A more detailed review is given in (Miksch et al., 1997).

On the other hand, workers in Medicine and Medical Informatics have recognized the importance of protocol-based care to ensure a high quality of care. An important approach was the definition of the Arden syntax (Hripcsak et al., 1994), which encodes situation-action rules. This syntax has significant limitations too: it currently supports only atomic data types, lacks a defined semantic for making temporal comparisons or for performing data abstraction, and provides no way to represent clinical guidelines that are more complex than individual situation-action rules (Musen et al., 1995). Therefore, the Arden syntax is not applicable for our purposes.

A common way to overcome these limitations is the representation of the above-mentioned procedural knowledge as a library of skeletal plans. Skeletal plans are plan schemata at various levels of detail that capture the essence of procedures, but leave room for execution-time flexibility in the achievement of particular goals (Friedland and Iwasaki, 1985). However, the basic concepts of skeletal plans are not sufficient in the medical domain, either.

Which Features Do We Need?

First, we need a plan *representation*, which (1) captures a hierarchical decomposition of plans, (2) is expressive with respect to temporal annotations, plan's intentions and effects, (3) has a rich set of sequential, concurrent, and cyclical operators. Thus, it should enable designers to express complex procedures in a manner similar to a real programming language (although typically on a higher level of abstraction), but in a more appropriate and useful way.

Second, we need a plan *visualization* which is able to capture: (1) hierarchical decomposition of plans (which are uniformly represented in a plan-specification library); (2) time-oriented plans; (3) sequential, concurrent, and cyclical execution of plans; (4) continuous (durative) states, actions, and effects; (5) intentions considered as high-level goals; and (6) conditions, that need to hold at particular plan steps. Additionally, all different time-oriented components of skeletal plans should be visualized in an easy to understand way. The domain experts, such as physicians, should understand the basic idea of skeletal plans. However, the domain experts do not need to be familiar with the syntax of skeletal plans (clinical protocols) to author them.

The Plan-Representation Language Asbru: A First Solution

Asbru[3] reflects all the described complexity in a language using a LISP-like syntax.

In Asbru, the following parts of a plan can be specified: *preferences*, *intentions*, *conditions*, *effects*, and *plan body (actions)*.

Preferences

Preferences constrain the applicability of a plan (e.g., select-criteria: exact-fit, roughly-fit) and express a kind of behaviour of the plan (e.g., kind of strategy: aggressive or normal).

[3] Asbru is part of a larger project, called *Asgaard*. In Norse mythology, *Asbru* (or *Bifrost*) was the bridge to *Asgaard*, the home of the gods.

Intentions

Intentions are high-level goals that should be reached by a plan, or maintained or avoided during its execution. These intentions are very important not only for selecting the right plan, but also for critiquing treatment plans as part of the ever ongoing process of improving the treatment. This makes intentions one of the key parts of Asbru.

Conditions

Conditions need to hold in order for a plan to be *started*, *suspended*, *reactivated*, *aborted*, or *completed*. Two different kinds of conditions (called preconditions) exist, that must be true in order for a plan to be started: filter-preconditions cannot be achieved (e.g., subject is female), setup-preconditions can. After a plan has been started, it can be suspended (interrupted) until either the restart-condition is true (whereupon it is continued at the point where it was suspended) or it has to be aborted. If a plan is aborted, it has failed to reach its goals. If a plan completes, it has reached its goals, and the next plan in the sequence is to be executed.

Effects

Effects describe the relationship between plan arguments and measurable parameters by means of mathematical functions. A probability of occurrence is also given.

Plan Body (Actions)

The plan body contains plans or actions that are to be performed if the preconditions hold. A plan is composed of other plans, which must be performed in sequence, in any order, in parallel, or periodically (as long as a condition holds, a maximum number of times, and with a minimum interval between retries).

A plan is decomposed into subplans until a nondecomposable plan — called an action — is found. This is called a *semantic stop condition* for the decomposition of plans. All the subplans consist of the same components as the plan, namely preferences, intentions, conditions, effects, and the plan body itself. An in-depth discussion of Asbru can be found in (Miksch et al., 1997).

Asbru Syntax

Plans in Asbru are written like in a programming language, as text that follows a very strict syntax. An example for a plan in Asbru syntax is given in Figure 1.

Asbru for Users: AsbruView

Its LISP-like syntax makes Asbru easy to understand for people familiar with programming languages, but physicians usually do not have degrees in Computer Science. So a way of visualizing Asbru had to be found that would make plans easy to overlook, while making all of Asbru's features available.

These graphics are impossible to draw by hand, but the use of a computer opens new possibilities, like metaphor graphics and the use of colors.

We evaluated the Asbru language and our visualization AsbruView with scenario-based techniques (Caroll, 1995) with collaborating physicians.

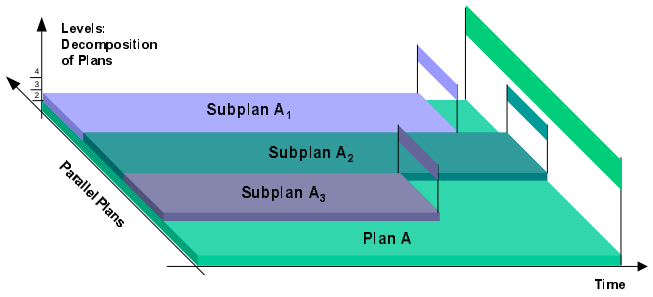

Figure 3: The representation of parallel plans.

Metaphor Graphics: Running Tracks and Traffic Control

The most basic part of Asbru and AsbruView is the plan. Inspired by examples in (Tufte, 1990; Tufte, 1997), a rather unusual metaphor was found to represent these plans: running tracks (see Figure 2 to Figure 4, and Figure 6). Additionally, traffic signs and other symbols from the world of traffic control were taken to symbolize certain concepts (Miksch et al., 1998).

We are using three-dimensional objects. The width represents the time axis, the depth represents plans on the same level of decomposition, and the height represents the decomposition of plans into subplans: The subplans of a plan appear 'on top' of it. (see Figure 3).

The 'base plane' of the diagram is the highest-level plan. Underneath it a scrollbar is situated that can be used to look at different parts of the plan (in the time-dimension) when only a part of it is being displayed (zoom). The highest-level plan is supported by two thin feet that act as controls for moving the scrollbar (see Figure 4, 6, and 7).

A control panel shows the user, which plan levels are currently displayed, and which aspect of the plans' design (e.g. conditions, intentions, etc.) are shown (Figure 2 and Figure 4).

A patient is considered to be allowed to enter the running track when all preconditions become true for him. These preconditions are represented by a 'no entrance with exceptions' sign for the *filter-preconditions*, and a turnpike for the *setup-preconditions*. The characteristics of the two preconditions are reflected in the metaphors: the *setup-precondition* can be achieved by performing some action, so the turnpike opens. The *filter-precondition* cannot be achieved, just as the 'exceptions' sign cannot be changed to gather access.

As long as no conditions become true that suspend the plan (the corresponding condition is symbolized by the yellow light of the traffic light (see Figure 5)), the patient 'runs' (i.e. the plan's actions are performed). When the *suspend-condition* becomes true, the runner must wait for the green light (*reactivate-condition*). Should the *abort-condition* (represented by the red light) become true, the runner is stopped and not allowed to continue. When the plan is completed (i.e., the *complete-condition*, represented by the finishing flag, has become true), he has reached the finishing line — and can proceed to the

```
(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-1#)]))
  (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 1: An example of Asbru code (part of a clinical treatment protocol for Infants' Respiratory Distress Syndrome (I-RDS)).

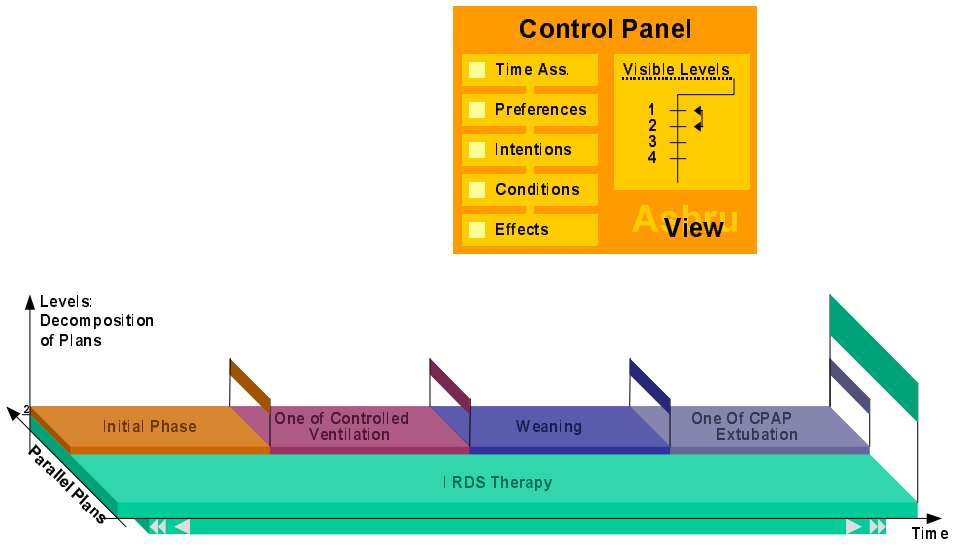

Figure 2: The decomposition of plans into subplans, and how sequential plans are represented.

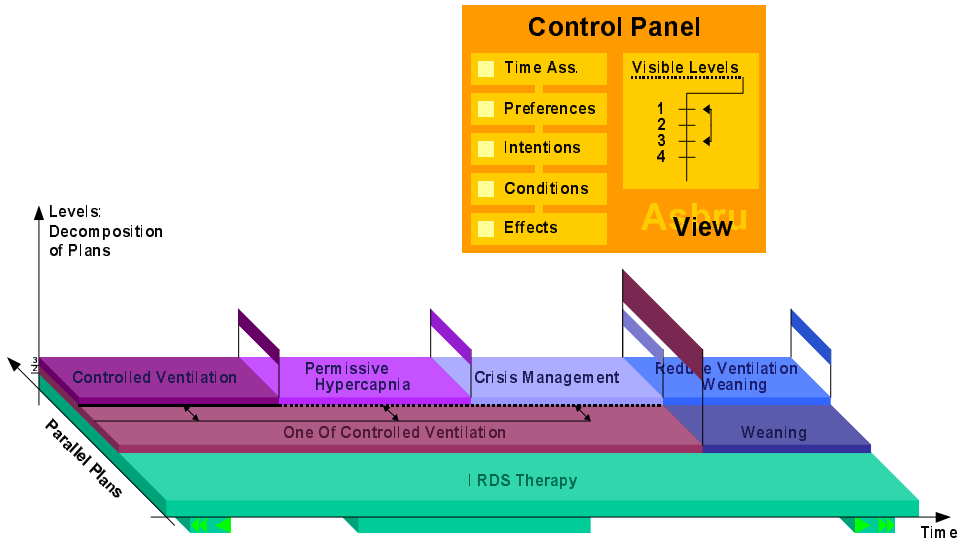

Figure 4: Some of the plans are to be executed in any order. The solid line marks the plans that must complete.

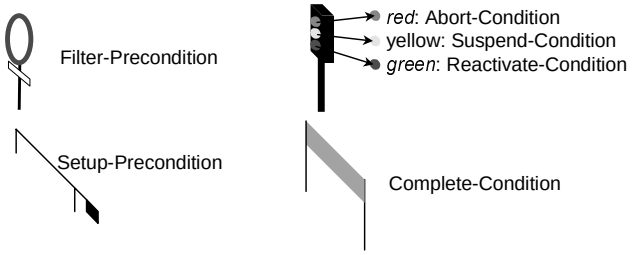

Figure 5: Metaphor graphics used in AsbruView.

next plan.

Plans can be arranged in different ways: sequentially, concurrently and cyclically. Plans that should be executed one after the other are simply put next to each other in the time dimension. No arrows are needed to link the plans.

Following Asbru-keywords, plans that should be executed in parallel are called *all-together* or *some-together* plans (depending on whether all or only a subset of them must be executed). If the order of execution makes no difference, the plans are called *all-any-order* and *some-any-order* plans, respectively.

Plans that should be executed in parallel are put alongside each other, so that more than one running tracks exist for a certain 'length' of time (see Figure 3). To execute plans in parallel means, that these plans should be started at the same time and be performed in parallel. However, these plans may complete at different time points, depending on their *complete-conditions*.

Additional symbols are needed to visualize the other operators. If not all plans of a set have to be executed, but some can be omitted, a single solid line along the base of a plan marks the plans that have to be performed (called the *continuation-condition*), and a dotted line marks the optional ones. In case of 'any-order' plans, a line is drawn on the underlying plan, providing an alternative way. Plans that must be executed are marked with a solid line, while optional ones are indicated by a dotted line (Figure 4).

In Asbru, the amount of time a plan takes to execute is not necessarily known beforehand. Thus, plans can not only be rectangular, but also take irregular shapes in order to show their different start/end times (Figure 6).

The representation used for these time-annotations is a more abstract one. Two horizontal bars, one above the other, show the minimum and maximum duration of a plan. The earliest and latest starting and ending times are labeled, and two diamonds support the upper bar (Figure 6). This is meant to indicate that the minimum duration is dependant on the latest starting time and the earliest finishing time: if they moved farther apart, the upper bar would 'fall down'. However, the minimum duration can be longer than the difference between latest starting time and earliest finishing time. If the earliest finishing time moves beyond the latest finishing time, the diamond 'falls off' the lower bar. So, despite its abstract nature, this representation can be used to explain the contraints as well as the possibilities of time annotations in Asbru.

When a plan has many subplans, which have many subplans themselves, it becomes difficult to navigate between the levels. Colors are used as a 'fourth dimension' to make plans and their relations easier recognizable.

AsbruView will be used to author clinical plans. It is therefore important to draw the planner's attention to *undefined components*. The general rule of undefined components is that undefined icons appear in gray. For example, the gray *setup-* and *filter-condition* in subplan *Reduced Ventilation Weaning* in Figure 7. Gray components can easily be spotted among the different colors of other parts of the diagram.

This rule also applies to control buttons that cannot be used at the moment (compare the 'left' and 'right'

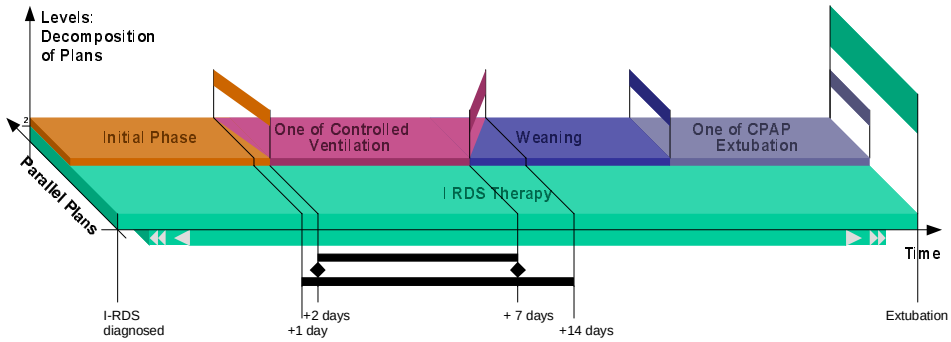

Figure 6: Plans can have irregular shapes depending on their time-behavior.

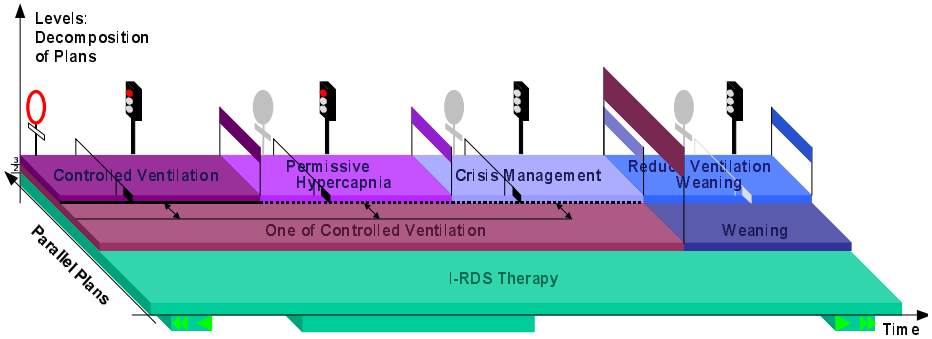

Figure 7: Plans' conditions.

arrows for the scrollbar in Figure 6 and Figure 7).

What are Flow-Charts?

A flow-chart is a diagram for representing algorithms and showing the decision structure of a program. The basic elements of a flow-chart are little boxes and arrows. The arrows connect the different boxes and sketch the control flow of statements (actions). These connectors can be used to sequence particular statements, to loop over one or more statements, or to go to another path of statements (*goto* statement). The different shapes of boxes denote the various kinds of statements (e.g., circles are start and stop buttons, little rectangles are commands or actions; diamonds are decisions; advanced rectangles are input and output data). Boxes can be numbered to refer to other diagrams that refine their contents. When introduced in (Goldstine and von Neumann, 1947), the little boxes and their contents served as a high-level language, grouping the inscrutable machine-language statements into clusters of significance.

When talking about flow-charts, we refer to the 'classic' flow-charts defined by (Goldstine and von Neumann, 1947) and standardized in DIN 66001, not one of the many extensions (e.g., (Martin, 1973; Nassi and Shneiderman, 1973)). These 'classic' flow-charts have a number of drawbacks, which may be of minor or no importance at all in other areas, but which make them impossible to use for our problem domain.

Why Not Flow-Charts?

Time

Clinical treatment involves one very important variable that is not accounted for in flow-charts: Time. It can be of vital importance to not only treat the patient right, but also to apply the treatment on time, or a certain number of times (with a certain interval in between). Time also plays an important role for the conditions that must be met in order to make use of a plan. A certain observable may not trigger a plan simply because of its value, but because of its behavior over time (e.g., rising or falling at a certain pace, or for a certain time).

The progress of time is simply not visible in flow-charts, but in AsbruView, it is a part of the design.

Intentions

Another problem is what may be called the 'real world' problem. People don't always follow plans, however well designed they may be. Physicians and nurses make their own decisions, often without being aware of them. And even if they don't follow the proposed actions of a plan, they may still follow its intentions. In medicine, there never is only one way to reach a goal, but there are many. It is impossible for the designer of the plans to know and to include all the different paths into her skeletal plan.

Using flow-charts, there would be no way of giving physicians the freedom to choose alternatives that have not been planned for by the plan's designers. Once the proposed way is left, there is no going back to the plan.

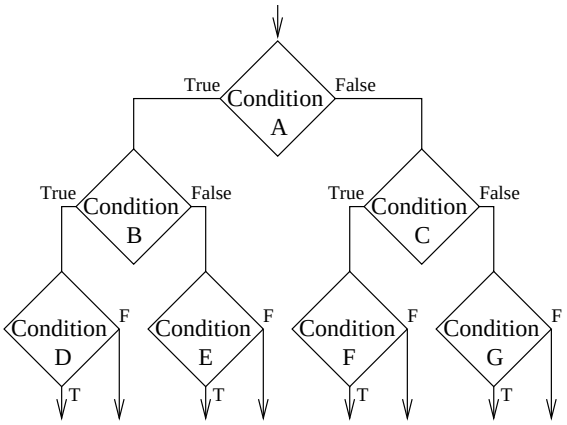

Figure 9: A decision-tree in a flow-chart.

With Asbru(View), physicians can choose different ways of achieving a goal, even such that the designers of the plan have not thought of. As long as the intention is the same, a physician or nurse may use any means instead of the proposed ones, and still continue using the plan as usual (for an example of intentions, see Figure 8).

Scalability and Object Orientation

In Asbru, therapies are broken up into small, manageable parts. Once a part is defined, it can be used as a building block for more complicated plans, that can themselves be used as building blocks, etc.

In order to use these building blocks, one simply includes them in the plan (as an alternative, for example). The conditions for applying the plan are part of the 'building block' plan itself, and thus need not be redefined in every plan that uses it.

With flow-charts, the decision would have to be made on the 'using plan' level, not that of the used plan. This leads to huge 'decision trees', once a larger number of subplans needs to be selected from (see Figure 9).

When a patient is treated, a virtually unlimited number of complications may occur. In such a case, an Asbru-plan is aborted, and the appropriate plan is sought to deal with the complication. It is impossible to add conditions and links for every complication when using flow-charts. So another means has to be used, which shows how ineffective and insufficient flow-charts are for this domain.

This certain 'object orientation' for conditions has another dimension as well: Plans inherit the using plan's preferences, intentions, conditions, and effects — but not their actions. This increases the possible modularity of the plans and decreases redundancy.

Complex Constructs

Flow-charts don't support different kinds of decisions, either. There is only one 'if condition A is true then goto α else goto β' construct. For practical purposes, it is impossible to create a 'do some of (α, β, γ)', or a 'do α, β, and γ in any order', even if one could build more complex parts from the available primitives. Some of these constructs are possible, but make the resulting flow-charts impossible to read — which clearly contradicts the concept of flow-charts.

Computer Aided Protocol Design

With most flow-charting tools, one gets very little support when changing parts of a chart. Suppose there are three plans, A, B and C in that order. After we have drawn the arrows accordingly, we realize that C should go in between A and B. So we have to delete the arrows from A to B and from B to C, move C between A and B, and then connect A to C and C to B.

AsbruView supports such changes: Simply take plan C and drop it on the edge between A and B — it will be inserted in the correct spot.

Conditions that have yet to be defined are displayed in gray. This is an important help for the plan designer to keep track of what has been done already and what still needs to be done.

Miscellaneous

As mentioned, plans need not only be applied, but their usefulness needs to be analyzed in order to improve the quality of treatment. Flow-charts supply no means of including a plan's intentions — AsbruView does. This is, of course, also true for the different kinds of conditions (that cannot be further differentiated with flow-charts), preferences, and effects.

Real-world clinical plans tend to consist of a huge number of actions and conditions and to be of a quite complicated structure. Therefore, more complicated plans are difficult if not impossible to oversee, when drawn using flow-charts. Additionally, flow-charting tools typically provide no means for splitting a larger plan into parts — especially in an 'incremental' manner, i.e., the decision to split the plan (and how) is made *after* the design has been started.

Conclusion and Future Plans

We have described the various problems, which authors of clinical plans face. Flow-charts are a very commonly used representation for clinical plans. However, flow-charts are not the appropriate means for representing such complex, time-oriented plans. We presented three-dimensional diagrams, called AsbruView, which overcome these limitations. The proposed diagrams utilize the metaphor graphics of 'running tracks' and 'traffic control', to visualize such clinical plans in an easy to understand way. The benefits of AsbruView are: (1) to handle all temporal dimensions of plans, conditions, intentions, and effects, (2) to cope with all possible, as well as unpredictable, orders of plan execution, (3) to deal with all the exception conditions that might arise, and (4) to deal with domain-specific features, like plans' intentions.

Currently, we are implementing AsbruView in Java. During the design phase, physicians have been consulted to improve AsbruView's usefulness for them and we are continuously evaluating the features of AsbruView with our clientele.

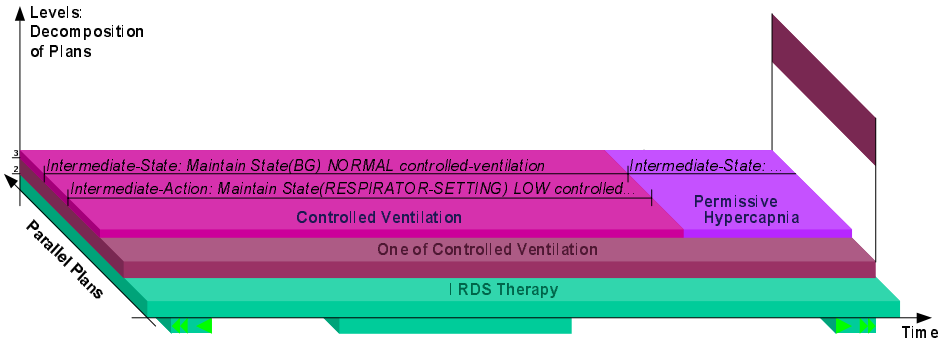

Figure 8: A plan's intentions.

Our final aim is to use AsbruView during the design and the execution phase. Therefore, we will adapt AsbruView to be used to author a protocol during the design phase as well as to visualize the performed protocols during the execution phase in a user-appropriate and task-specific way.

Acknowledgements

The authors thank Johannes Gärtner, Werner Horn, Christian Popow, Franz Paky, Georg Duftschmid, and Klaus Hammermüller for helpful comments and discussions; and Leonore Neuwirth for her help with getting hold of an important paper.

References

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

Friedland, P. E. and Iwasaki, Y. (1985). The concept and implementaion of skeletal plans. *Journal of Automated Reasoning*, 1(2):161–208.

Genesereth, M. R. and Fikes, R. E. (1982). Knowledge interchange format, version 3.0 reference manual. Tech.report logic-92-1, Computer Science Department, Stanford University.

Goldberg, A. and Rubin, K. S. (1995). *Succeeding with objects: decision frameworks for project management.* Addison-Wesley, Reading, Massachusetts.

Goldstine, H. and von Neumann, J. (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: McMillian, pp. 80-151.

Guarina, N. and Giaretta, P. (1995). Ontologies and knowledge bases. In Mars, N. J. I., editor, *Towards Very Large Knowledge Base.* IOS Press, Amsterdam.

Hripcsak, G., Ludemann, P., Pryor, T. A., Wigertz, O. B., and Clayton, P. D. (1994). Rationale for the Arden syntax. *Computers and Biomedical Research*, 27:291–324.

Levinson, R. (1995). A general programming language for unified planning and control. *Artificial Intelligence*, 76(1-2):319–275.

Martin, J. (1973). The 'natural' set of basic control structures. *SIGPLAN Notices*, 8(12):5–14.

Miksch, S., Kosara, R., Shahar, Y., and Johnson, P. (1998). Asbruview: Visualization of time-oriented, skeletal plans. In *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems 1998 (AIPS-98)*, Pittsburgh Pennsylvania, USA, Menlo Park, CA. Carnegie Mellon University, AAAI Press.

Miksch, S., Shahar, Y., and Johnson, P. (1997). 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.

Musen, M. A., Gennari, J. H., Eriksson, H., W.Tu, S., and R.Puerta, A. (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, I. and Shneiderman, B. (1973). Flowchart techniques for structure programming. *SIGPLAN Notices*, 8(8):12–26.

Pattison-Gordon, E., Cimino, J. J., Hripcsak, G., Tu, S. W., Gennari, J. H., Jain, N. L., and Greenes, R. A. (1996). Requirements of a sharable guideline representation for computer applications. Report no. smi-96-0628, Stanford University.

Tufte, E. R. (1990). *Envisioning Information.* Graphics Press, Cheshire, CT.

Tufte, E. R. (1997). *Visual Explanations.* Graphics Press, Cheshire, CT.