The Asgaard Project: A Task-Specific Framework for the Application and Critiquing of Time-Oriented Clinical Guidelines

Yuval Shahar[1], Silvia Miksch[2], and Peter Johnson[3]

Address of contact author:

[1] Section on Medical Informatics,
251 Campus Drive,
Medical School Office Building, x215
Stanford University,
Stanford, CA 94305–5479,
USA
Email: shahar@smi.stanford.edu
Tel: 1-650-725-3393
Fax: 1-650-725-7944

[2] Vienna University of Technology
Institute of Software Technology
Resselgasse 3/E188,
A-10140 Vienna,
Austria
Email: silvia@ifs.tuwien.ac.at
Tel: +43-1-58801-4089
Fax: ++43-1-504 05 32

[3] The Sowerby Centre for Primary Care Informatics,
University of Newcastle,
Newcastle upon Tyne,
NE2 4AA,
UK
email: pete@mimir.demon.co.uk

The Asgaard Project: A Task-Specific Framework for the Application and Critiquing of Time-Oriented Clinical Guidelines

Yuval Shahar, Silvia Miksch, and Peter Johnson

Abstract

Clinical guidelines can be viewed as generic skeletal-plan schemata that represent clinical procedural knowledge and that are instantiated and refined dynamically by care providers over significant time periods. In the Asgaard project, we are investigating a set of tasks that support the application of clinical guidelines by a care provider other than the guideline's designer. We are focusing on application of the guideline, recognition of care providers' intentions from their actions, and critique of care providers' actions given the guideline and the patient's medical record. We are developing methods that perform these tasks in multiple clinical domains, given an instance of a properly represented clinical guideline and an electronic medical patient record. In this paper, we point out the precise domain-specific knowledge required by each method, such as the explicit intentions of the guideline designer (represented as temporal patterns to be achieved or avoided). We present a machine-readable language, called Asbru, to represent and to annotate guidelines based on the task-specific ontology. We also introduce an automated tool for acquisition of clinical guidelines based on the same ontology, developed using the PROTÉGÉ-II framework.

Key Words: Knowledge representation, Knowledge acquisition; Planning; Temporal reasoning; clinical Guidelines; Critiquing; Plan recognition

1. Clinical Guidelines

Clinical guidelines are a set of schematic plans for management of patients who have a particular clinical condition (e.g., insulin-dependent diabetes). The application of clinical guidelines by care providers involves collecting and interpreting considerable amounts of data over time, applying standard therapeutic or diagnostic plans in an episodic fashion, and revising those plans when necessary. Guidelines often involve implicit assumptions about the knowledge of the provider executing the plans. *Skeletal plans* are plan schemata at various levels of detail that capture the essence of a procedure, but leave room for execution-time flexibility in the achievement of particular goals [5]. Thus, they are usually reusable in different contexts. Clinical guidelines can be viewed as reusable

skeletal plans that need to be refined by a reactive planner over significant time periods when applied to a particular patient [21].

1.1. Automated Support to Guideline-Based care

During the past 15 years, there have been several efforts to support guideline-based care over time in automated fashion. Examples of specialized architectures include ONCOCIN [21], T-HELPER [12], DILEMMA [8], EON [13], and the European PRESTIGE project. Other approaches to the task of supporting guideline-based care encode guidelines as elementary state-transition tables or as situation-action rules dependent on the electronic medical record [19], but do not include an intuitive representation of the guideline's clinical logic, and have no semantics for the different types of clinical knowledge represented. Several approaches permit hypertext browsing of guidelines via the World Wide Web [1] but do not use the patient's electronic medical record.

None of the current guideline-based-care systems have a sharable representation of guidelines that (1) has knowledge roles specific to the several guideline-based-care tasks, (2) is machine and human readable, and (3) allows data stored in an electronic patient record to invoke an application that directly executes the guideline's logic and related tasks, such as critiquing. A task-specific human- and machine-readable representation of clinical guidelines, that has an expressive syntax and semantics, combined with the ability to interpret that representation in automated fashion, would facilitate guideline dissemination, real-time accessibility, and applicability. *Task-specific architectures* [4] assign problem-solving *roles* to domain knowledge and facilitate acquisition and maintenance of that knowledge. Such a representation also would support additional reasoning tasks, such as automated critiquing, quality assurance [7], and guideline evaluation, and would facilitate authoring and modifying clinical guidelines.

1.2. Support to Application of Clinical Guidelines as an Interactive Process

Application of guidelines involves an interpretation by the care provider of skeletal plans that have been designed by the guideline's author. Providing automated support implies an interactive process. Typical tasks include assessment of the applicability of the guideline to the patient, guidance in proper application of a selected guideline, monitoring of the application process, assessment of the results of the guideline, critiquing the application process and its results, and assistance in the modification of the original guideline. For instance, clinical guidelines often have an inherent ambiguity or incompleteness. To increase flexibility, an automated assistant should recognize cases in which the care provider's actions, although different from the guideline's prescribed actions, adhere to the overall intentions of the guideline's designer, and should adjust accordingly its critique. To be useful, the language in which clinical guidelines are represented needs to be temporally expressive and should enable designers to express complex sequential, parallel, and cyclical procedures in a manner akin to a programming language (although typically on a higher level of abstraction). The language also requires well-defined semantics for both the prescribed actions and the task-specific annotations, such as the guideline designer's intentions. Thus, the care-provider's actions can be

better supported, leading to a more flexible dialog and to a better acceptance of automated systems for support of guideline-based care. Having clear semantics for the task-specific knowledge roles also facilitates acquisition and maintenance of these roles.

Given these requirements, we have developed a text-based, machine-readable language, called *Asbru*. The Asbru language is part of the *Asgaard*[1] project, in which we are developing task-specific problem-solving methods that perform execution and critiquing tasks in medical domains.

In Section 2, we introduce the design-time and execution-time intention-based model, the guideline-application support tasks it comprises and their required knowledge roles, and our framework's overall architecture. Section 3 explains the syntax and the semantics of the Asbru language. Section 4 demonstrates how an automatically generated graphic knowledge-acquisition tool is used to acquire Asbru guidelines from physicians. Section 5 discusses the operation of the Asbru interpreter. We will be illustrating our approach throughout by a guideline for controlled observation and treatment of gestational diabetes mellitus (GDM) Type II.

2. A Design-Time versus Execution-Time Intention-Based Model

During *design time* of a clinical guideline, an *author* (or a committee) designs a guideline (Fig. 1). The author prescribes (1) *actions* (e.g., administer a certain drug in the morning and in the evening), (2) an *intended plan*—the intended intermediate and overall pattern of actions, which might not be obvious from the description of the prescribed actions and is often more flexible than prescription of specific actions (e.g., use some drug from a certain class of drugs twice a day), and (3) the intended intermediate and overall pattern of p*atient states* (e.g., morning blood glucose should stay within a certain range). Intentions are temporal patterns of provider actions or patient states, to be achieved, maintained, or avoided.

[1] In Norse mythology, Asgaard was the home and citadel of the gods, corresponding to Mount Olympus in Greek mythology. It was located in the heavens and was accessible only over the rainbow bridge, called Asbru (or Bifrost).

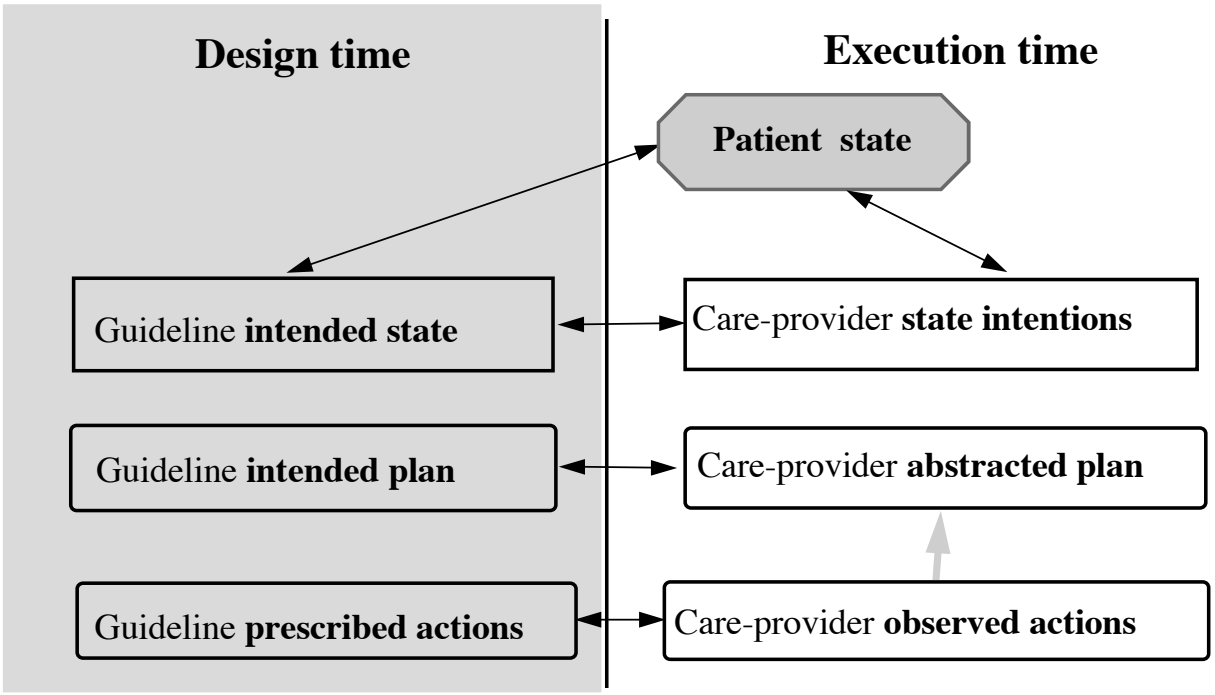

Figure 1. The design-time versus execution-time intention-based model of a clinical guideline. Double-headed arrows denote a potential axis of comparison (e.g., for critiquing purposes) during runtime execution of the clinical guideline. Striped arrows denote an abstracted-into relationship.

During *execution time*, a *care provider* applies the guideline by performing *actions*, which are recorded, observed, and abstracted over time into an *abstracted plan* (see Fig. 1). The *state* of the patient also is recorded, observed, and abstracted over time. Finally, the *intentions of the care provider* might be recorded too—inferred from her actions or explicitly stated by the provider.

2.1. The Guideline-Design And -Application Tasks

Given the intention-based model of clinical guidelines, we can describe a set of tasks relevant to the design and execution of these guidelines and analyze the knowledge requirements of problem-solving methods that perform these tasks (Table 1). The verification and validation tasks are relevant only during design time; the rest of the tasks are relevant during execution time. Each task can be viewed as answering a specific question (see Table 1).

Task	Questions to be answered	Required Knowledge
Verification of a guideline	Are the intended plans achievable by following the prescribed actions? (*a syntactic check*)	Prescribed actions; intended overall action pattern (i.e., the plan)
Validation of a guideline	Are the intended states achievable by the prescribed actions and intended plan? (*a semantic check*)	Prescribed actions, intended overall action pattern; intended states; action/plan effects
Applicability of guidelines	What guidelines or protocols are applicable at this time to this patient?	Filter and setup preconditions; overall intended states; the patient's state
Execution (application) of a guideline	What should be done at this time according to the guideline's prescribed actions?	Prescribed actions and their filter and setup preconditions; suspension, restart, completion, and abort conditions; the patient's state
Recognition of intentions	Why is the care provider executing a particular set of actions, especially if those deviate from the guideline's prescribed actions?	Executed actions and their abstraction to executed plans; action and state intentions; the patient's state; action/plan effects; revision strategies; preferences
Critique of the provider's actions	Is the care provider deviating from the prescribed actions or intended plan? Are the deviating actions compatible with the author's plan and state intentions?	Executed actions and their abstraction to plans; action and state intentions of the original plan; the patient's state; action/plan effects; revision strategies; preferences
Evaluation of a guideline	Is the guideline working?	Intermediate/overall state intentions; the patient's state; intermediate/overall action intentions; executed actions and plans
Modification of an executing guideline	What alternative plans are relevant at this time for achieving a given state intention?	Intermediate/overall state intentions; action/plan effects; filter and setup preconditions; revision strategies; preferences; the patient's state

Table 1: Several guideline-support tasks and the knowledge required to solve them. Common knowledge roles can be viewed as shareable by the methods requiring them.

Each task can be performed by a problem-solving method [4] that has an *ontology*—a set of entities, relations, and domain-specific knowledge requirements assumed by the method. Since knowledge requirements (*roles*) are often common to several of the problem-solving methods, we combine them into a task-specific *knowledge cluster*. Examples of knowledge roles include plan intentions, several types of preferences, and state-transition conditions.

The semantics of the specific knowledge roles used in the Asbru language are discussed in the Section 3. Given these knowledge roles, we can define what knowledge is required to solve each task (see Table 1).

In this paper, we focus on the dynamic (runtime) execution, plan-recognition, and critiquing tasks. However, are also looking at the static verification and validation tasks. Static verification of a guideline plan involves the examination of an uninstantiated plan for logical consistency. An Asbru plan is logically consistent, if none of its explicit or implicit logical constraints are violated; in particular, the plan intentions are compatible with the prescribed actions. Such a verification is essentially *syntactic* in nature (e.g., is visiting a dietician each Tuesday consistent with the intention of visiting a dietician at least four times a month?) Validation involves a *semantic* check that the intended patient states are compatible with the prescribed actions and intended plans. Of course, it is often difficult or impossible to ascertain that any particular patient state will result from performance of certain actions; but gross inconsistencies can be detected, given sufficient domain-specific knowledge, such as expected effects of drugs and guidelines.

A subtask implicit in several of the tasks in Table 1 is the abstraction of higher-level concepts from time-stamped data during the execution of the skeletal plan. Possible candidates for solving this subtask include the RÉSUMÉ system and the temporal data-abstraction component in the VIE-VENT system. The *RÉSUMÉ* system [17] is an implementation of a formal, domain-independent problem-solving method, the *knowledge-based temporal-abstraction method* [18] and has been evaluated in several clinical domains [17]. *VIE-VENT* is an open-loop knowledge-based monitoring and therapy planning system for artificially ventilated newborn infants, which includes context-sensitive and expectation-guided temporal data-abstraction methods [10].

2.2. Plan Recognition And Critiquing In The Application Of Clinical Guidelines

The following example demonstrates the tasks of plan-recognition and critiquing in the domain of monitoring and therapy of patients who have insulin-dependent diabetes.

During therapy of a diabetes patient, hyperglycemia (a higher than normal level of blood glucose) is detected for the second time in the same week around bedtime. The diabetes-guideline's prescribed action might be to increase the dose of the insulin the patient typically injects before dinner. However, the provider recommends reduction of the patient's carbohydrate intake during dinner. This action seems to contradict the prescribed action. Nevertheless, the automated assistant notes that increasing the dose of insulin decreases the value of the blood-glucose level directly, while the provider's recommendation *decrease* the value of the same clinical parameter by *reducing* the magnitude of an action (i.e., ingestion of carbohydrates) that *increases* its value. The assistant also notes that the state intention of the guideline was "avoid more than two episodes of hyperglycemia per week." Therefore, the provider is still following the intention of the guideline. By recognizing this high-level intention and its achievement by a different

plan, the automated assistant can accept the provider's alternate set of actions, and even provide further support for these actions.

We consider a *plan-recognition* ability, such as demonstrated in the example, an indispensable prerequisite to the performance of *plan critiquing*. Such an ability might increase the usefulness of guideline-based decision-support systems to clinical practitioners, who often follow what they consider as the author's intentions rather than the prescribed actions. Note that we assume knowledge about the *effects of interventions* on clinical parameters, and knowledge of legitimate domain-independent and domain-specific *guideline-revision strategies*. Both intervention effects and revision strategies can be represented formally [16].

Intentions have been examined in philosophy [2] and in artificial intelligence [14]. As we explain in more detail in Section 3.2, we are viewing intentions formally as temporally extended goals, comprising action or state patterns, at various abstraction levels.

The example also demonstrates a specific *execution-critiquing* model. In this model, five comparison axes exist: the guideline's prescribed actions versus the provider's actual actions; the guideline's intended plan versus the provider's (abstracted) plan; the guideline's intended patient state versus the provider's state intention; the guideline's intended state versus the patient's (abstracted) actual state; and the provider's intended state versus the patient's (abstracted) actual state. Combinations of the comparison results imply a set of different *behaviors* of the guideline application by the provider. Thus, a care provider might not follow the precise *actions*, but still follow the intended *plan* and achieve the desired states. A provider might even not follow the overall plan, but still adhere to a higher-level *intention*. Alternatively, the provider might be executing the guideline correctly, but the patient's state might differ from the intended, perhaps indicating a complication that needs attention or a failure of the guideline. In theory, there might be up to 32 different behaviors, assuming binary comparisons along five axes. However, the use of consistency constraints prunes this number to approximately 10 major behaviors. (We also are investigating the use of continuous, rather than binary, measures of matching).

2.3. The Conceptual Architecture

In the Asgaard project, we are developing different task-specific reasoning modules that perform the guideline-support tasks shown in Table 1. Figure 2 presents the overall architecture.

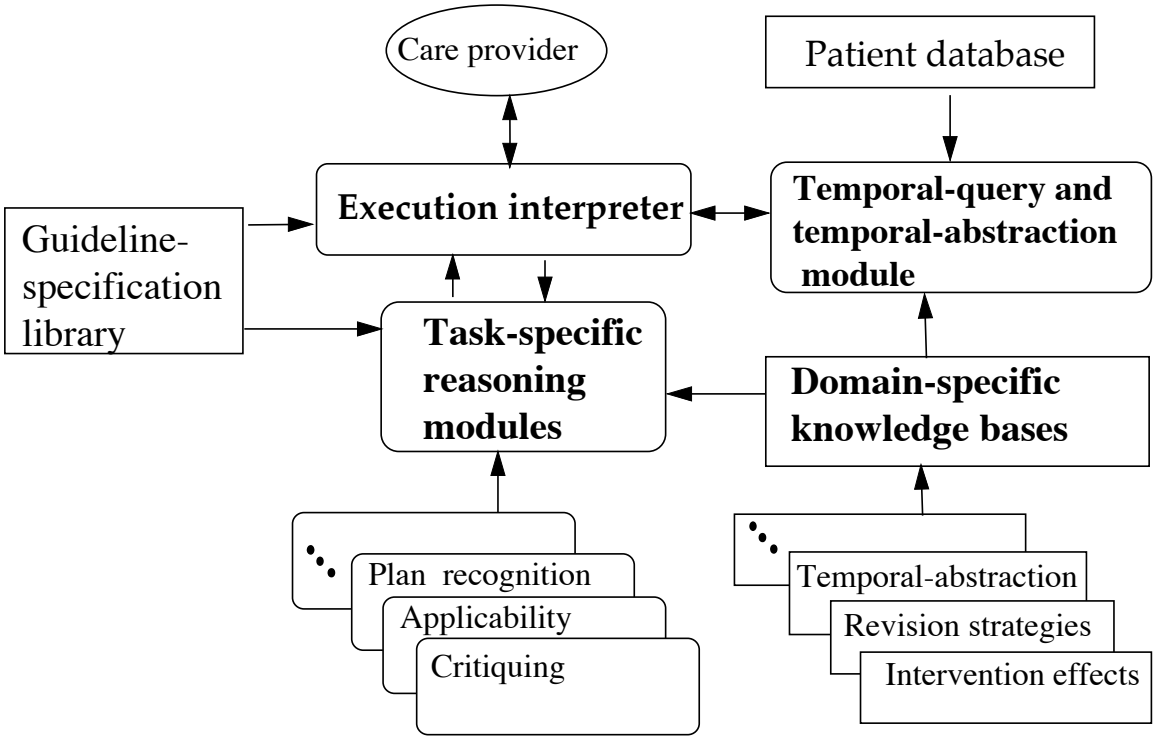

Figure 2. The guideline–support architecture. Arrows denote data or knowledge flow.

The task-specific reasoning modules require different types of knowledge, often outside of the scope of the guideline-tasks ontology. For instance, the knowledge-based temporal-abstraction method implemented by the RÉSUMÉ module requires knowledge about temporal-abstraction properties of measurable clinical parameters, such as persistence of their values over time when these values are not recorded [18]. These properties exist in the domain's *temporal-abstraction ontology* [17]. The RÉSUMÉ temporal-abstraction module is part of a temporal-query and temporal-abstraction module that is used to query the time-oriented patient record for both raw darta and higher-level temporal abstractions and patterns. Currently, we are using the Tzolkin temporal mediator [22] as the temporal-query and temporal-abstraction module. The Tzolkin module includes, besides the RÉSUMÉ temporal-abstraction module, the Chronus [23] temporal-query module and a controller that parses temporal queries and coordinates the two modules.

Similarly, the plan-recognition and critiquing methods require generic and domain-specific plan-revision knowledge [16]; much of that knowledge might not be part of the guideline specification, but can be represented in a separate knowledge base accessible to the appropriate problem-solving methods (reasoning modules). Effects of specific interventions, such as insulin administration (and, in general, of guidelines) can be represented as part of the guideline, but can also be viewed as a separate knowledge base (Fig. 2). The specifications of clinical guidelines and of their independent components (we refer to either of these entities as *plans* in this paper) are all represented

uniformly and organized in a *guideline-specification library*. The library is a set of execution plans expressed in our task-specific language. During the guideline-execution phase, an applicable guideline plan is instantiated with runtime arguments.

3. Asbru: A Global Ontology For Guideline-Application Tasks

We have developed a language specific to the set of guideline-support tasks and the problem-solving methods performing these tasks, which we call *Asbru*. Asbru enables a designer to represent a clinical guideline, including all of the knowledge roles useful to one or more of the problem-solving methods performing the various tasks supporting the application of clinical guidelines. The major features of Asbru are that prescribed actions can be continuous; plans might be executed in parallel, in sequence, in a particular order, or every time measure; temporal scopes and parameters of guideline plans can be flexible, and explicit intentions and preferences can underlie the plan. These features are in contrast to most traditional plan-execution representations, which have significant limitations and are not applicable to dynamic environments such as clinical domains. Medical domains have certain characteristic features: (1) actions and effects are not necessarily instantenuous: actions are often continuous (have duration) and might have delayed effects; (2) goals often have temporal extensions; (3 there is uncertinty regarding the effect of available actions; (4) unobservable, underlying processes determine the observable state of the world; (5) a goal may not be achievable; (6) parallel and periodic execution of plans is common. The requirements of plan specifications in clinical domains [21] are a superset of the requirements of typical toy domains used in planning research. We have defined a formal syntax for the Asbru language in Backus-Naur form [11]. The Asbru language combines the flexibility and expressivity of procedural languages (e.g., the Arden syntax [9]) with the semantic clarity of declaratively expressed knowledge roles. These roles (e.g., preferences and intentions) are specific to the ontology of the methods performing the guideline-support tasks.

3.1. Time Annotation

The time annotation we use allows a representation of uncertainty in starting time, ending time, and duration of a time interval [3], [15]. The time annotation supports multiple time lines by providing different *reference annotations*. The reference annotation can be an absolute reference point, a reference point with uncertainty (defined by an uncertainty region), a function (e.g., completion time) of a previously executed plan instance, or a domain-dependent time point variable

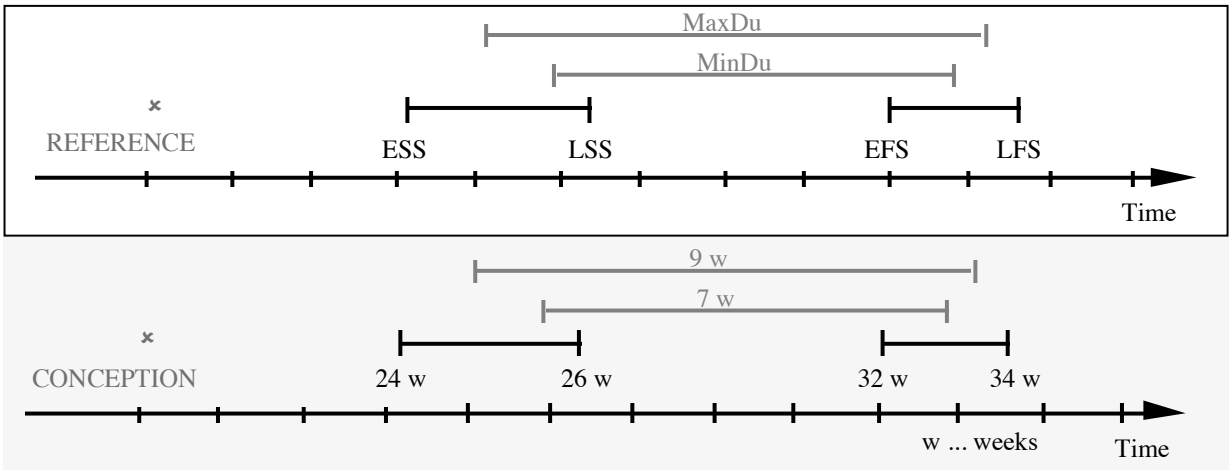

Figure 3. A schematic illustration of the Asbru time annotations. The upper part of the Fig. presents the generic annotation. The lower part shows a particular example representing the time annotation [[24 WEEKS, 26 WEEKS], [32 WEEKS, 34 WEEKS], [7 WEEKS, 9 WEEKS], CONCEPTION]), which means "starts 24 to 26 weeks after conception, ends 32 to 34 weeks after the conception, and lasts 7 to 9 weeks." REFERENCE = reference annotation, ESS = earliest starting shift, LSS = latest starting shift, EFS = earliest finishing shift, LFS = latest finishing shift, MinDu = minimal duration, MaxDu = maximal duration. The annotation is thus ([ESS, LSS], [EFS, LFS], [MinDu, MaxDu], REFERENCE).

(e.g., CONCEPTION). Temporal shifts from the reference annotation represent uncertainty in the starting time, the ending time, and the overall duration (Fig. 3). Thus, the temporal annotation represents for each interval the earliest starting shift (ESS), the latest starting shift (LSS), the earliest finishing shift (EFS), the latest finishing shift (LFS), the minimal duration (MinDu) and the maximal duration (MaxDu). Temporal shifts are measured in time units. Thus, a temporal annotation is written as ([ESS, LSS], [EFS, LFS], [MinDu, MaxDu], REFERENCE). All temporal-shift constraints can be unknown (unbound, denoted by an underscore,"_") to allow incomplete time annotations.

To allow temporal repetitions, we define sets of cyclical time points (e.g., MIDNIGHTS, which represents the set of midnights, where each midnight occurs exactly at 0:00 A.M., every 24 hours) and cyclical time annotations (e.g., MORNINGS, which represents a set of mornings, where each morning starts at the earliest at 8:00 A.M., ends at the latest at 11:00 A.M., and lasts at least 30 minutes). In addition, we allow certain short-cuts such as for the current time, whatever that time is (using the symbol *NOW*), or the duration of the plan (using the symbol *). Thus, the Asbru notation enables the expression of interval-based intentions, states, and prescribed actions with uncertainty regarding starting, finishing, duration, and the use of absolute, relative, and even cyclical (with a predetermined granularity) reference annotations.

3.2. The Semantics of the Asbru Task-Specific Knowledge Roles

A (guideline) *plan* in the guideline-specification library is composed of a set of plans with arguments and time annotations. A decomposition of a plan into its subplans is attempted by the execution interpreter, unless the plan is

not found in the guideline library, thus representing a nondecomposable plan. This can be viewed as a "semantic" halting condition, which increases runtime flexibility, since the same plan might imply an atomic action for one clinical site, but might be decomposable into more primitive actions at another clinical site. A nondecomposable plan is executed by the user or by an external call to a computer program. The library includes a set of *primitive plans* to perform interaction with the user or to retrieve information from the medical patient record (e.g., OBSERVE, GET-PARAMETER, ASK-PARAMETER, DISPLAY, WAIT)). All plans have return values.

Generic library plans that are mentioned as part of an executing guideline have states (`considered`, `possible`, `rejected`, and `ready`), that determine whether the plan is applicable and whether a plan instance can be created (Fig. 4).

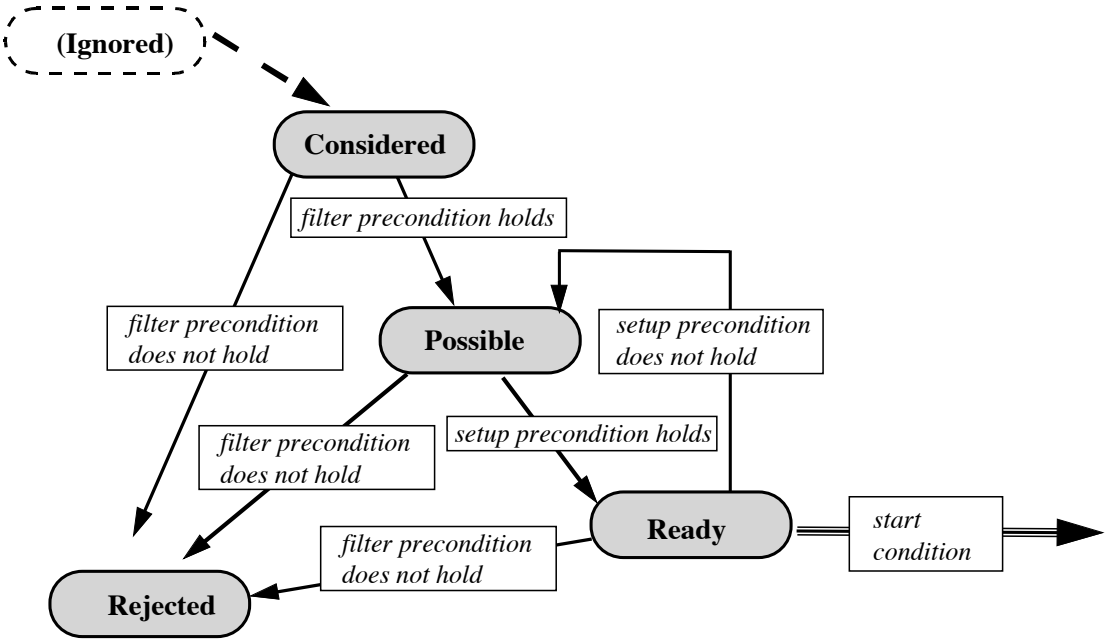

Figure 4. Plan-selection states and their state-transition conditions in Asbru. These states are relevant to library plans that are considered for execution.

At execution time, a *ready* plan is instantiated. A set of mutually exclusive *plan states* describes the actual status of the plan instance during execution. Particular *state-transition criteria (conditions)* specify transition between neighboring plan-instance states. Thus, if a plan is `activated`, in can only be `completed`, `suspended`, or `aborted` depending on the corresponding criteria; the `suspended` state is optional and available for complex plans (Fig. 5).

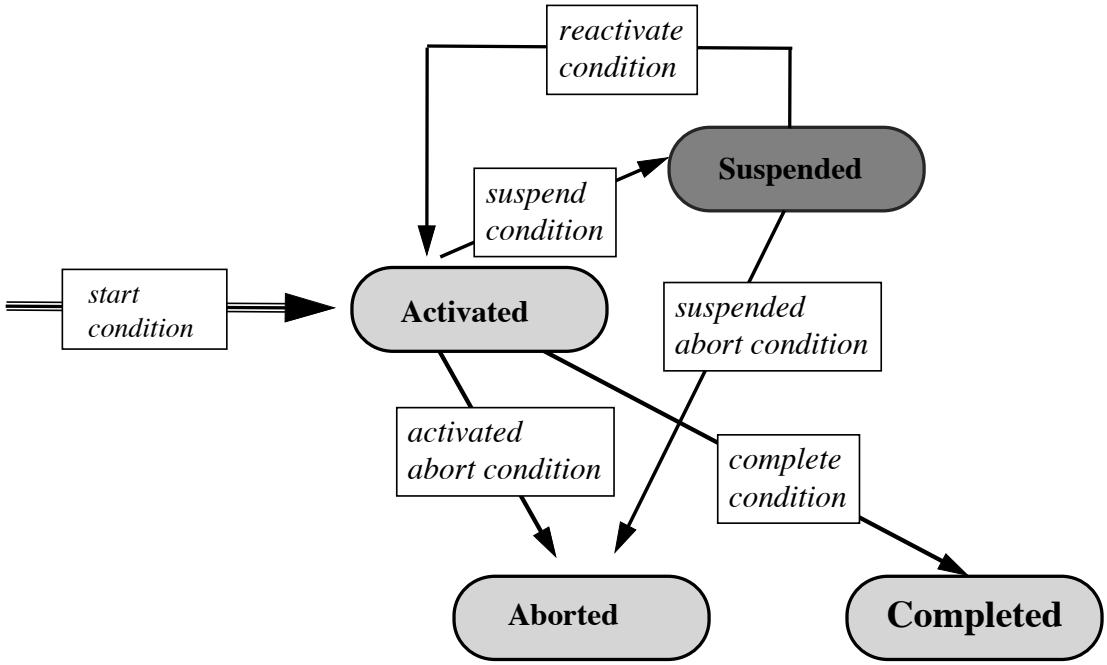

Figure 5. Plan-execution states and their state-transition conditions in Asbru. These states are relevant to instances of library plans that are being executed.

State-transition conditions are explained below. A plan consists of a name, a set of arguments, including a time annotation (representing the temporal scope of the plan), and five (optional) components: *preferences*, *intentions*, *conditions*, *effects*, and a *plan body* which describes the actions to be executed. All components are optional. Every subplan has the same structure. Thus, a sequential plan can include several potentially decomposable concurrent or cyclical plans.

We now examine in more detail each of the knowledge roles represented in Asbru.

Preferences: Preferences bias or constrain the selection of a plan to achieve a given goal. Examples include:

(1) *Strategy*: a general strategy for dealing with the problem (e.g., aggressive, normal);

(2) *Utility*: a set of utility measures (e.g., minimize the cost or the patient inconvenience);

(3) *Select-method*: a matching heuristic for the applicability of the whole plan (e.g., exact-fit);

(4) *Resources*: a specification of prohibited or obligatory resources (e.g., in certain cases of treatment of a pulmonary infection, surgery is prohibited and antibiotics must be used);

(5) *Start-conditions*: an indication whether transition from a `ready` state of the generic plan to an `activated` state of the plan instance is automatic or requires approval of the user.

Intentions: Intentions are high-level goals at various levels of the plan, an annotation specified by the designer that supports special tasks such as critiquing and modification. Intentions are temporal patterns of provider actions and patient states, at different levels of abstraction, that should be maintained, achieved, or avoided. We define four categories of intentions:

(1) *Intermediate state:* the patient state(s) that should be maintained, achieved, or avoided during the applicability of the plan (e.g., weight gain levels are slightly low to slightly high);

(2) *Intermediate action:* the provider action(s) that should take place during the execution of the plan (e.g., monitor blood glucose once a day);

(3) *Overall state pattern:* the overall pattern of patient states that should hold after finishing the plan (e.g., patient had less than one high glucose value per week);

(4) *Overall action pattern:* the overall pattern of provider actions that should hold after finishing the plan (e.g., patient had visited dietitian regularly for at least three months).

Conditions: Conditions are temporal patterns, sampled at a specified frequency, that need to hold at particular plan steps to induce a particular state transition of the plan instance. They are used for actual execution (application) of the plan. We do not directly determine conditions that should hold during execution; we specify conditions that activate the change of a particular plan state (see Fig. 5). A plan instance is completed when the complete conditions become true, otherwise the plan instance's execution suspends or aborts (often, due to failure). Conditions are optional. We distinguish six types of conditions:

(1) *filter-preconditions*, which need to hold initially if a generic plan is applicable; these conditions are not goals to be achieved (e.g., patient is a pregnant female), and must be true to achieve a `possible` state (see Fig. 4);

(2) *setup-preconditions*, which need to be achieved (usually, within a given time delay relative to the initial time of consideration of the plan) to enable a plan to start (e.g., patent had a glucose-tolerance test) and allow a transition from a `possible` plan to a `ready` plan (see Fig. 4);

(3) *suspend-conditions*, which determine when an active plan instance has to be suspended (e.g., blood glucose has been high for four days); these are informally the inverse of *protection* conditions in the planning literature, which have to hold during certain time periods (see Fig. 5);

(4) *abort-conditions*, which determine when an active or suspended plan has to be aborted (e.g., there is an insulin-indicator condition: the patient cannot be controlled by diet) (see Fig. 5);

(5) *complete-conditions*, which determine when an active plan is completed, typically, but not necessarily, successfully (e.g., delivery has been performed) (see Fig. 5);

(6) *reactivate-conditions*, which determine when a suspended plan has to be reactivated (e.g., blood glucose level is back to normal or is only slightly high) (see Fig. 5).

Effects: Effects describe the functional relationship between either (1) each of the relevant plan arguments and measurable parameters it affects in certain contexts (e.g., the *dose* of insulin is inversely related in some fashion to the level of blood glucose) or (2) the overall plan and the clinical parameters it is expected to effect (e.g., the insulin-administration plan decreases the blood-glucose level). Effects can have a likelihood annotation—a probability of occurrence. Effects can be part of the guideline library (when they annotate plans) and can also be stored in a domain-specific knowledge base (especially for common plans, such as administration of drugs).

Plan-Body: The plan body is a set of plans to be executed in parallel, in sequence, in any order, or in some frequency. We distinguish among three types of plans: *sequential*, *concurrent*, and *cyclical*. Only one type of plan is allowed in a single plan body. A sequential plan specifies a set of plans that are executed in sequence; for continuation, all plans included have to be completed successfully. Concurrent plans can be executed either together, in parallel, or in any order. We distinguish two dimensions for classification of sequential or (potentially) concurrent plans: the number of plans that should be completed to enable continuation and the order of plan execution. Table 2 summarizes the dimensions of the two plan types. Using the two dimensions, we define the plan subtypes DO-ALL-TOGETHER, DO-SOME-TOGETHER, DO-ALL-ANY-ORDER, DO-SOME-ANY-ORDER, DO-ALL-SEQUENTIALLY. The continuation condition specifies the names of the plans that must be completed to proceed with the next steps in the plan.

Continuation condition --> Ordering Constraints	All plans should be completed in order to continue	Some plans should be completed in order to continue
Start together	DO-ALL-TOGETHER (no continuation-condition; all plans must complete)	DO-SOME-TOGETHER (continuation-conditions specified as subset of plans)
Execute in any order	DO-ALL-ANY-ORDER (no continuation-condition; all plans must complete)	DO-SOME-ANY-ORDER (continuation-conditions specified as subset of plans)
Execute in total order	DO-ALL-SEQUENTIALLY (no continuation-condition; all plans must complete)	------

Table 2: Categorization of plan types by continuation conditions and ordering constraints.

A cyclical plan (a DO-EVERY type) includes a plan that can be repeated, and optional temporal and continuation arguments that can specify its behavior. *Start* and *end* specify a starting and ending time point. *Time base* determines the time interval over which the plan is repeated and the start time, end time, and duration of the

particular plan instance in each cycle (e.g., starting with the first Monday's morning, until next Tuesday's morning, perform plan A every morning for 10 minutes). The *times-completed* argument specifies how many times the plan has to be completed to succeed and the *times-attempted* argument specifies how many attempts are allowed. Obviously, number of attempts must be greater or equal to the number of successful plans. A temporal pattern can be used as a stop condition of the cyclic plan. Finally, the plan itself is associated with its own particular arguments (e.g., dose). The start time, the time base, and the plan name are mandatory to the specification of a cyclic plan; the other arguments are optional.

3.3. Example: A Gestational Diabetes Mellitus Guideline

We represented in Asbru a Stanford University guideline for controlled observation and treatment of gestational diabetes mellitus (GDM) type II (non insulin dependent). The guideline prescribes several concurrent monitoring and management plans following a glucose tolerance test (GTT) between 140 and 200 mg/dl (Fig. 6). The plan body consists of three plans that are executed in parallel (*glucose monitoring*, *nutritional management*, and *monitoring for insulin indication*), exist in the guideline-specification library, and are decomposable into other library plans.

```
(PLAN observing-GDM

;; the following time-annotations are local to the GDM example
(DOMAIN-DEPENDENT TIME-ASSIGNMENT
   (SHIFTS DELIVERY <- 38 WEEKS) ;; domain-specific time shift from the CONCEPTION point
   (POINT CONCEPTION <- (ask (ARG "what is the conception-date?")))
(ABSTRACTION-ASSIGNMENT
   (CYCLIC
      MIDNIGHTS <- [0, 0 HOURS, 24 HOURS]
      BREAKFAST-START-TIME <- [0, 7 HOURS, 24 HOURS]))
(PREFERENCES
   (SELECT-METHOD EXACT-FIT)   ;; The match in the filter conditions needs to be exact
   (START-CONDITION AUTOMATIC)) ;; the plan starts as soon as it is ready, no user input
(INTENTION:INTERMEDIATE-STATE
   (MAINTAIN blood-glucose-post-meal (<= 130) GDM-Type-II
      [[24 WEEKS, 24 WEEKS], [DELIVERY, DELIVERY], [_,_], CONCEPTION])
   (MAINTAIN blood-glucose-fasting (<= 100) GDM-Type-II
      [[24 WEEKS, 24 WEEKS], [DELIVERY, DELIVERY], [_,_], CONCEPTION])
(INTENTION:OVERALL-STATE
   (AVOID STATE(blood-glucose) HIGH GDM-Type-II
      [[24 WEEKS, 24 WEEKS], [DELIVERY, DELIVERY], [7 DAYS,_], CONCEPTION]))
   ;; avoid, throughout the guideline, a period of high blood-glucose level lasting more than 7 days

(FILTER-PRECONDITIONS
   (one-hour-GTT (140, 200) pregnancy
        [24 WEEKS, 24 WEEKS], [26 WEEKS, 26 WEEKS], [_,_], CONCEPTION])
(SETUP-PRECONDITIONS
   (PLAN-STATE one-hour-GTT COMPLETED
      [[24 WEEKS, 24 WEEKS], [26 WEEKS, 26 WEEKS], [_,_], CONCEPTION])
      ;; The patient must have completed a glucose-tolerance test (another plan in the library)
(SUSPEND-CONDITIONS
   (STATE(blood-glucose) HIGH GDM-Type-II
      [[24 WEEKS, 24 WEEKS], [DELIVERY, DELIVERY], [4 DAYS,_], CONCEPTION]
      (SAMPLING-FREQUENCY 24 HOURS)))
      ;; suspend if high blood-glucose level exists for at least 4 DAYS
(ABORT-CONDITIONS
   (insulin-indicator-conditions TRUE GDM-Type-II *
      (SAMPLING-FREQUENCY 24 HOURS)))
(COMPLETE-CONDITIONS
   (delivery TRUE GDM-Type-II * (SAMPLING-FREQUENCY 24 HOURS)))
(REACTIVATE-CONDITIONS
   (STATE(blood-glucose) (OR NORMAL SLIGHTLY-HIGH) GDM-Type-II
      [[24 WEEKS,  24 WEEKS], [DELIVERY, DELIVERY], [_,_], CONCEPTION]
   (SAMPLING-FREQUENCY 24 HOURS)))

(DO-ALL-TOGETHER
  (glucose-monitoring)
  (nutrition-management)
  (observe-insulin-indicators)))
     ;; the plan body is a concurrent one; the three plans start together and all need to complete
```

Figure 6. A small portion of the representation of the guideline for management of non-insulin-dependent gestational diabetes mellitus (GDM) type II. Double colons are followed by comments.

4. Acquisition and maintenance of Guideline Plans

Expert physicians need not have familiarity with the syntax of the Asbru language to author clinical guidelines. Graphical **knowledge-acquisition** (**KA**) tools can be generated automatically by systems such as **PROTÉGÉ-II** [20]. The KA tools can

internally use the Asbru representation or its equivalent, but that representation need not necessarily be known to the user. In addition to creation of an internal (e.g., object-oriented) version of the plan, the KA tool should be able to generate a text-based Asbru version. The Asbru version can then be used as a sharable machine-readable version that does not depend on any particular platform and will also be useful for reading and editing by more knowledgeable designers. We have explored the option of generating an automated graphic KA tool for acquiring the set of shared knowledge roles, using the PROTÉGÉ-II suite of tools, with encouraging results.

4.1. Modeling a clinical-guideline ontology using the PROTÉGÉ II methodology

PROTÉGÉ-II is a set of tools and a methodology to develop knowledge based systems. We used **PROTÉGÉ/Win** (the Windows version of PROTÉGÉ-II) to develop the ontology and to generate a graphical KA tool automatically from the ontology. This ontology is shared by the task-specific cluster of problem-solving methods relevant to the support of skeletal-plan execution. In PROTÉGÉ-II terms, we have developed a *method ontology*, global to all our problem-solving methods. By this we mean that the ontology is in theory local (specific) to some hypothetical, all-encompassing method that performs the task cluster, but is in practice global to (shared by) all the methods that perform subtasks in that cluster. This is sensible in this case, as there is a great deal of overlap in the concepts needed for the different tasks, and the roles undertaken by these concepts are the same in all their uses by this set of tasks. The *domain ontology* in the case of clinical guidelines is also required, but not shown here. The domain ontology specifies concepts, such as drugs, diseases, patient findings, tests, and clinic visit types.

Ontologies in PROTÉGÉ-II are represented as a hierarchy of classes. Each class is represented as a frame with slots. Slots may be constrained to basic data types, or to be instances of another class defined in the ontology, thus allowing the expression of relationships in the ontology. The PROTÉGÉ/Win OntologyEditor tool was used to capture the ontology of the cluster of methods supporting skeletal-plan execution. Fig. 7 shows a portion of that ontology.

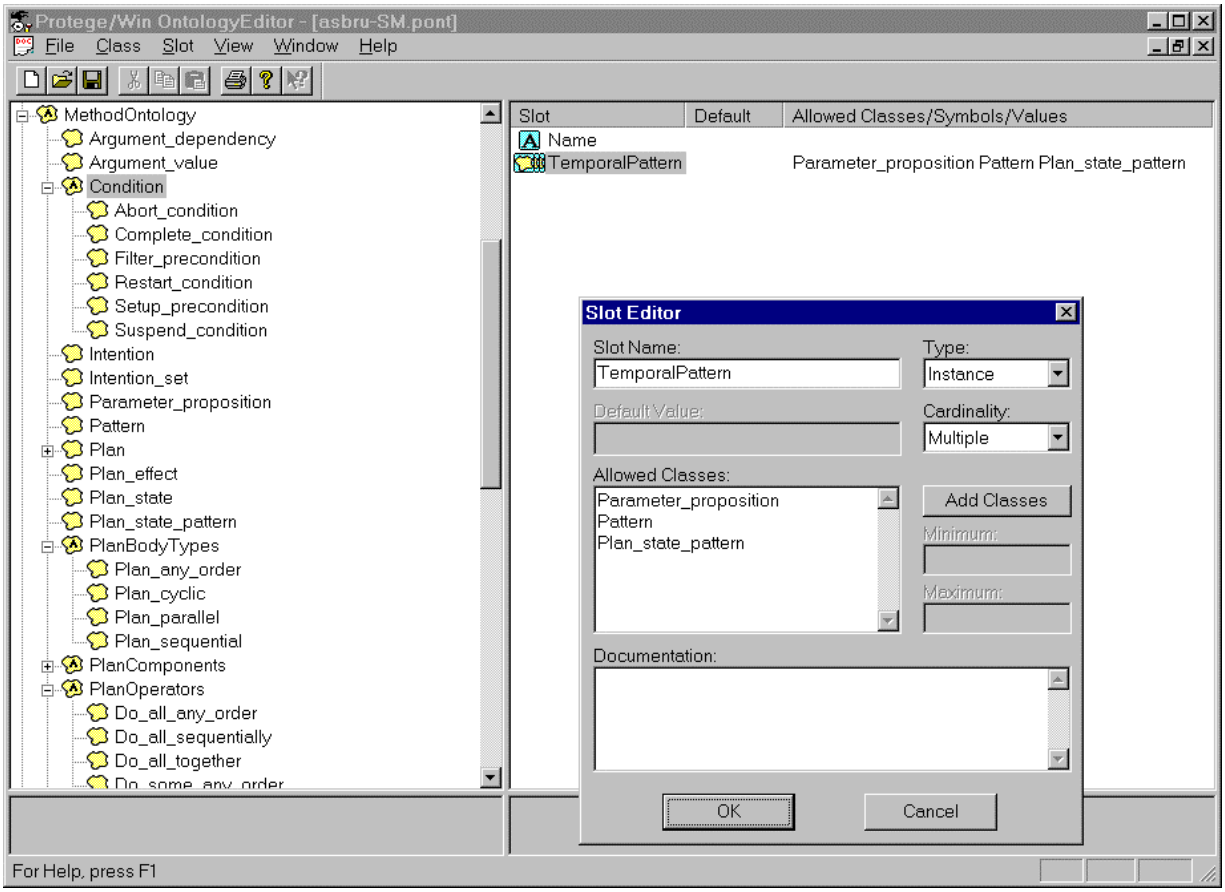

Figure 7. Part of the execution–support methods ontology, represented by the PROTÉGÉ/Win Ontology Editor

Once the ontology is defined, the PROTÉGÉ/Win LayoutEditor tool automatically generates a specification of a graphical KA tool for this ontology. The specification of the KA tool is interpreted by the PROTÉGÉ/Win LayoutInterpreter. It is possible to change the layout of the user interface to some degree in the LayoutEditor. The resultant KA tool can then be used to acquire instances of the ontology, which in this case would be guidelines in the Asbru language.

The knowledge roles in the Asbru syntax can be viewed as a set of slots in a frame-based ontology of skeletal plans. The ontology that we have developed mirrors the BNF Syntax of the Asbru language. As this language is not object oriented, the ontology is fairly flat, in that inheritance is not used significantly. The concepts of inheritance and polymorphism can be usefully applied to this domain, indeed it seems more natural to express the ontology in this form. As an example, all plans in the current ontology share the same state transition criteria, which has been chosen as one which applies to most actions. However, in a hierarchical ontology it is easy to create special subclasses of the plan class which have variants of the state transition criteria. Such an ontology maps well as an object-oriented language.

4.2. A knowledge-acquisition tool for support of clinical-guideline execution

We have generated an automated graphical KA tool for the object-oriented version of the Asbru language, using the PROTÉGÉ-II suite of tools, with encouraging results. Fig. 8 shows an example of a domain expert using the KA tool to acquire a part of the GDM type II guideline. The expert has defined the body of the guideline as being composed of three parallel plans, all three to be started together; one of the plans is being highlighted and examined, as wel as the overall state intention of the top-level (GDM type II) plan.

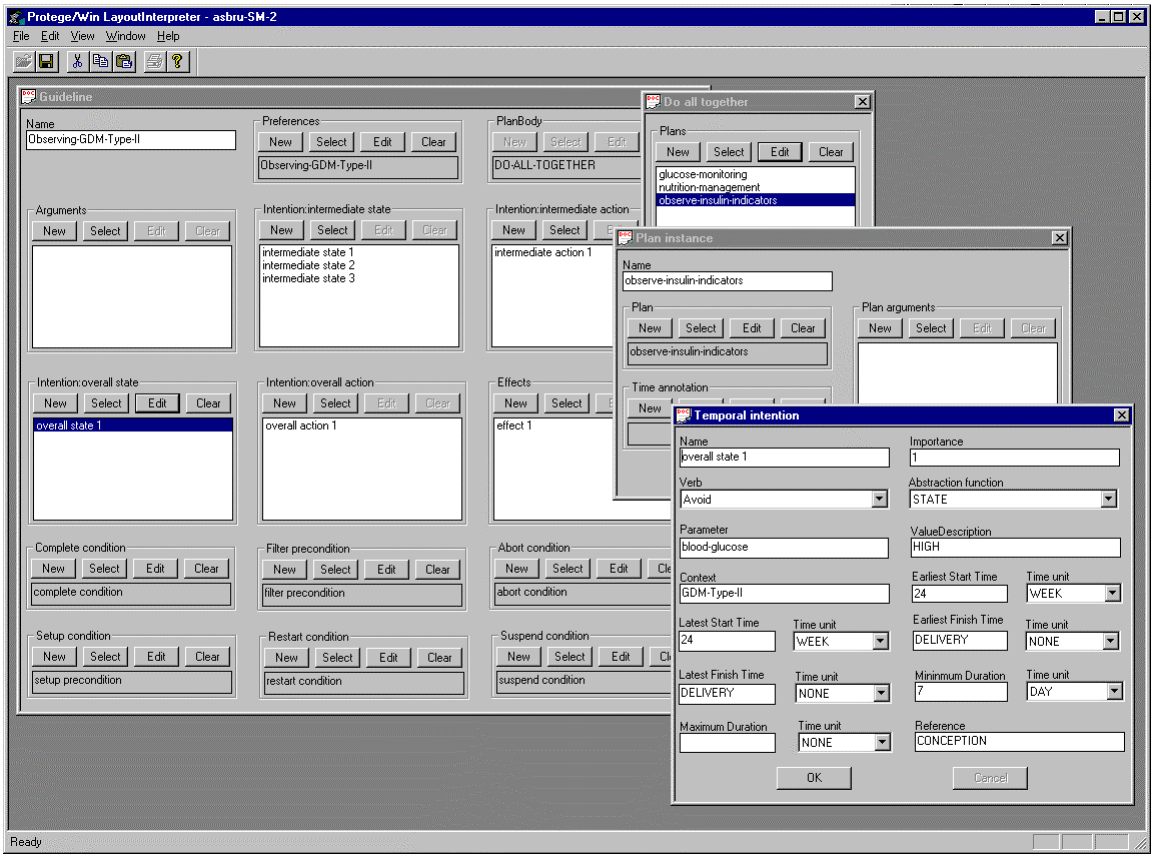

Figure 8. The Asbru knowledge-acquisition tool. The screen shot demonstrates acquisition of part of the gestational diabetes mellitus (GDM) type II guideline from a domain expert. The main window shows the guideline's components and is generated automatically by the Protégé/Win tools from an Asbru ontology. Two of the three subwindows show acquisition of one of three parallel (do-all-together) plans that the GDM guideline is composed of; the third shows a definition of the GDM guideline's overall state intention.

If the user is conversant with the syntax of the Asbru language, it may be quicker to design guidelines by writing them in the language, using 'copy and paste' functions or editor macros. If the user, in particular a domain expert, is unfamiliar with the syntax, then it is easier to use the KA tool. The complexity of the ontology enforces the automatic generator of the KA tool to produce a user interface with many cascading, small dialogs. Thus, for

providing an optimal dialog, more control of the layout of the automatically generated user interface was needed than was possible in early versions of PROTÉGÉ/Win. Once this additional feature became available, we managed to generate and customize significantly better KA tools, although more improvements can take place.

Another significant benefit of the KA tool approach is that it detects incorrect syntax while authoring a guideline. Thus, implicit syntactic support is provided at no cost.

One of the interesting areas for acquisition of knowledge is the temporal annotations. We are examining several alternatives to our ontology-based graphic interface, including a specialized *string editor* that accepts as input the BNF syntax of temporal annotations, and creates automatically a graphical KA tool that acquires legal strings from the user (Fig. 9). The string editor is part of the PROTÉGÉ/Win framework. We are also exploring other representations to facilitate acquisition from domain experts, such as assuming that complex temporal patterns are composed of a small number of temporal intervals, amongst which the user only needs to specify temporal and value relations, while each is acquired as a separate instance.

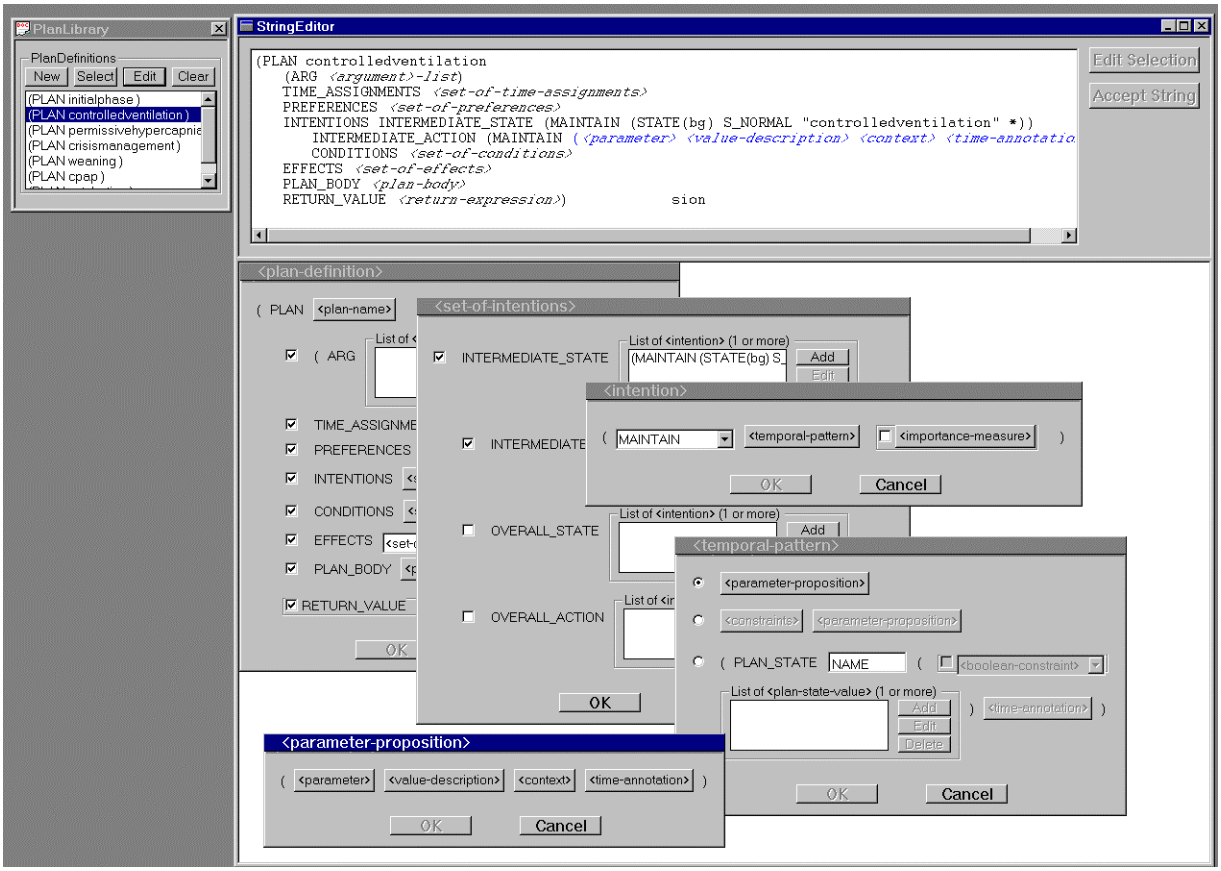

Figure 9. The string editor. The screen shot is showing acquisition of part of a treatment guideline for newborn infants who are being mechanically ventilated from a domain expert. In the string editor, the acquisition process is guided directly by the Asbru langauge's BNF syntax.

5. The Asbru Interpreter

We have implemented an interpreter for the Asbru language (i.e., a computational module that performs the execution task). The Asbru interpreter is implemented in the CLIPS language [6] and assumes as input a library of generic guideline plans represented as CLIPS objects, and a top-level plan (with appropriate arguments, if relevant) to be considered for execution. Guideline plans are acquired through the graphic KA tool generated through the PROTÉGÉ/Win system, and can also be created directly from an Asbru pure-text version by a simple 1:1 mapping. Thus, the Asbru interpreter assumes a generic, objected-oriented intermediate representation for plans that can be acquired directly from an expert physician using the graphic KA tool or that can be created by a preprocessor (a parser) from an existing (text-based) Asbru guideline written by a domain expert who knows the language. The Asbru interpreter creates as output *plan objects* that refer to generic or instantiated plans, and communicates with the user by sending messages involving plan objects (e.g., regarding execution of atomic plan instances that cannot be decomposed further, or regarding the need to make certain set-up conditions true to make a possible generic plan ready to be instantiated).

The Asbru interpreter goes through several distinct *phases*. In each phase, all plan objects (i.e., both those that refer to generic plan types and those that refer to plan instances) that refer to (generic or instantiated) plans in a particular state are processed. Therefore, the phases correspond roughly to potential states of both generic plans and plan instances, and include *consider*, *check-possible*, *check-ready*, and *execute* phases. The *execute* phase includes several subphases in which the *abort*, *suspend*, and *complete* conditions of all plan-objects are being checked. In each phase, an appropriate list of plan objects is processed. Thus, for instance, in the *consider* phase, the interpreter examines the filter condition for all considered plans, changing their states to *possible* or *rejected* until the list of considered plans is empty.

The Asbru interpreter implements several aspects of the pragmatic semantics underlying the Asbru syntax. For instance, aborting any type of plan instance (e.g., an instance of a parallel plan) aborts all its component plans (e.g., the plan instances executing in parallel as part of a Do-All-Together plan type). The inverse is not necessarily true, and the effect of aborting a component plan on the parent plan depends on the type of the plan.

We are implementing the plan-recognition and critiquing task by building on this model of interpretation with respect to the execution model, and on our temporal-abstraction model [18] and the Asbru temporal annotations with respect to the representation and querying of patient data.

6. Summary and Discussion

Representing clinical guidelines and the intentions underlying them in a standard, machine-readable, and machine-interpretable form is a prerequisite for sharing clinical guidelines and for useful, flexible automated assistance in the

execution of these guidelines. The task-specific representation we suggest supports several different *knowledge roles* that can be used by multiple reasoning modules, for direct execution of a guideline and for related tasks such as recognition of care providers' intentions and critiquing their actions. In addition, the Asbru language places a particular emphasis on an expressive representation for time-oriented provider actions and patient states. Temporal reasoning is important for many clinical domains and needs to be supported.

Expert physicians need not have familiarity with the Asbru syntax to author clinical guidelines. We used the PROTÉGÉ-II framework to develop an Asbru-like object-oriented guideline ontology and to generate a graphical knowledge-acquisition tool automatically from the ontology. The tool acquires instances of guidelines modeled in an object-oriented Asbru version.

We are continuing to develop the computational modules, focusing on the execution, plan recognition, and critiquing modules.

Acknowledgments

Yuval Shahar has been supported by grants LM05708 and LM06245 from the National Library of Medicine, and IRI-9528444 from the National Science Foundation. We thank Kinkoi Lo for implementation of the prototype of the Asbru interpreter.

References

(1) M. Barnes and G. O. Barnett, An Architecture for a Distributed Guideline Server, in: R. M. Gardner, ed., Proceedings of the Annual Symposium on Computer Applications in Medical Care (SCAMC-95) (New Orleans, LA, 1995) (Hanley & Belfus, 1995) 233–237.

(2) M.E. Bratman, Intention, Plans and Practical Reason, (Harvard University Press, Cambridge, MA, 1987).

(3) R. Dechter, L. Meiri, and J. Pearl, Temporal Constraint Networks, Artificial Intelligence, Special Volume on Knowledge Representation 49(1-3) (1991) 61–95.

(4) H. Eriksson, Y. Shahar, S. W. Tu, A. R., Puerta, and M. A. Musen, Task Modeling with Reusable Problem-Solving Methods, Artificial Intelligence 79(2) (1995) 293–326.

(5) P. Friedland and Y. Iwasaki, The Concept and Implementation of Skeletal Plans, Journal of Automated Reasoning 1(2) (1985) 161–208.

(6) J. Giarratano and G. Riley, Expert Systems: Principles and Programming, (PWS Publishing Company, Boston, 1994).

(7) J. M. Grimshaw and I. T. Russel, Effects of Clinical Guidelines an Medical Practice: A Systematic Review of Rigorous Evaluation, Lancet, 342 (1993) 1317–1322.

(8) S. I. Herbert, C. J. Gordon, A. Jackson-Smale, and J-L. Renaud Salis, Protocols for Clinical Care, Computer Methods and Programs in Biomedicine 48 (1995) 21–26.

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

(10) S. Miksch, W. Horn, C. Popow, and F. Paky, Utilizing Temporal Data Abstraction for Data Validation and Therapy Planning for Artificially Ventilated Newborn Infants, Artificial Intelligence in Medicine 8(6) (1996) 543-576.

(11) S. Miksch, Y. Shahar, and P. Johnson, Asbru: A task-specific, intention-based, and time-oriented language for representing skeletal plans, in: Proceedings of the Seventh Workshop on Knowledge Engineering Methods and Languages (KEML-97) (Milton Keynes, UK, 1997) 9-1 – 9-20,.

(12) M. A. Musen, C. W. Carlson, L. M. Fagan, S. C. Deresinski, and E. H. Shortliffe, T-HELPER: Automated Support for Community-Based Clinical Research, in: M. E Frisse,. ed., Proceedings of the Sixteenth Annual Symposium on Computer Applications in Medical Care (SCAMC-92) (McGraw Hill, New York, 1992) 719–723.

(13) M. A. Musen, S. W .Tu, A. K. Das, and Y. Shahar, EON: A Component-Based Approach to Automation of Protocol-Directed Therapy, Journal of the American Medical Information Association 3(6) (1996) 367–388.

(14) M. Pollack, The Use of Plans, Artificial Intelligence 57(1) (1992) 43-68.

(15) J. F. Rit, Propagating Temporal Constraints for Scheduling, in: Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86) (Morgan Kaufmann, Los Altos, CA, 1986) 383–388.

(16) Y. Shahar and M. A. Musen, Plan Recognition and Revision in Support of Guideline-Based Care, in: Notes of the AAAI Spring Symposium on Representation Mental States and Mechanisms (Stanford, CA, 1995) 118–126.

(17) Y. Shahar and M. A. Musen, Knowledge-Based Temporal Abstraction in Clinical Domains, Artificial Intelligence in Medicine 8(3) (1996) 267–298.

(18) Y. Shahar, A Framework for Knowledge-Based Temporal Abstraction, Artificial Intelligence 90 (1)(1997) 79–133.

(19) E. H.Sherman, G. Hripcsak, J. Starren, R. A. Jender, and P. Clayton, Using Intermediate States to Improve the Ability of the Arden Syntax to Implement Care Plans and Reuse Knowledge, in: R. M. Gardner, ed., Proceedings of the Annual Symposium on Computer Applications in Medical Care (SCAMC-95) (New Orleans, LA, 1995) (Hanley & Belfus, 1995) 238–242.

(20) S. W. Tu, H. Eriksson, J. H. Gennari, Y. Shahar, and M. A. Musen, Ontology-Based Configuration of Problem-Solving Methods and Generation of Knowledge-Acquisition Tools: Application of PROTÉGÉ-II to Protocol-Based Decision Support, Artificial Intelligence in Medicine 7(3) (1995) 257–289.

(21) S. W. Tu, M. G. Kahn, M. A. Musen, J. C. Ferguson, E. H. Shortliffe, and L. M. Fagan, Episodic Skeletal-Plan Refinement on Temporal Data, Communications of ACM 32 (1989) 1439–1455.

(22) J. Nguyen, Y. Shahar, S.W.Tu, A.K. Das, and M.A. Musen, A temporal database mediator for protocol-based decision support, Proceedings of the 1997 AMIA Annual Fall Symposium (formerly the Symposium on Computer Applications in Medical Care) (Nashville, TN) (Hanley and Belfus, 1997) 298–302.

(23) A.K. Das and M.A. Musen, A temporal query system for protocol-directed decision support, Methods of Information in Medicine 33 (1994) 358-370.