Modeling Computer-Supported Clinical Guidelines and Protocols
A Survey

Katharina Kaiser and Silvia Miksch

Vienna University of Technology
Institute of Software Technology & Interactive Systems
Asgaard-TR-2005-2 March 2005
Abstract

Clinical Practice Guidelines are an important means for improving quality in care. They describe systematic developed statements to promulgate the most effective and efficient treatment. Due to the growing complexity of treatment processes there arose efforts to provide a computer-supported interpretation and execution of the clinical guidelines.

In this paper we present several methodologies that were developed for computer-supported clinical practice guidelines. We describe the different strengths and weaknesses of the various approaches as well as methods and tools to author guidelines in a computer-supported way.
Chapter 1

Introduction

Errors in health care are a leading cause of death and injury. Kohn et al. [18] mention that, for example, preventable adverse events are a leading cause of death in the United States. Studies have implied that at least 44,000 and perhaps as many as 98,000 Americans die in hospitals each year as a result of medical errors. Significantly, there is no data about Central Europe, which is doubtlessly a consequence of our exposure with errors. In Austria, we act on the assumption of about 3,000 to 5,000 events of death per year due to preventable errors in hospitals.

This leads us to efforts for improving the quality of care, concurrently reducing cost of care, and improving workflow for health care providers.

Clinical guidelines and protocols (CGPs) are ”systematically developed statements to assist practitioners and patient decisions about appropriate health care for specific circumstances” [7]. CGPs typically address a specific health condition and provide recommendations to the physician about issues such as who to investigate for the problem, how to investigate it, how to diagnose it, and how to treat it. Research has shown that, if properly developed, communicated, and implemented, guidelines can improve patient care [16]. CGPs provide not only decision support for the medical personal (physicians, nursing staff, etc.), patients, and relatives, but also promulgate the most effective and efficient treatment. Therefore, CGPs are an important issue in quality assurance.

Many researchers have proposed frameworks for modeling CGPs in a computer-interpretable and -executable format [13, 10, 36, 8, 48, 28, 46]. These frameworks are tailored for specific classes of guidelines, specific users, and specific organizations.

Tu and Musen [49] define five tasks that computer-supported guideline-based care must provide: (1) making decisions, (2) sequencing actions and decisions, (3) defining goals (e.g., achieve/maintain/avoid special patient states), (4) interpreting data (i.e., deriving abstract concepts from concrete data), and (5) refining actions. Therefore, CGPs are not only an important issue in quality assurance (during the therapy), but also in quality control (during and after the therapy). But to execute CGPs in a computer-supported way, the information in the guideline, which is in plain textual form, in tables, or represented in flow charts, has to be formalized. That means, a formal representation is required in order to make the information computable. Thus, several so called guideline representation languages were de-
veloped for the assistance of varied guidelines and protocols and to make different kinds of applications possible.

Most of these languages have accomplished a state of complexity where the modeling process of a CGP into such a language is a very challenging venture. Thus, research has to be directed in such a way where tools are developed for supporting the modeling process.

The next sections give an overview on various guideline representation languages as well as tools that support the modeling process of CGPs into the formal representation of these languages.
Chapter 2

Guideline Representation Languages

Various guideline representation languages have been and are developed for different purposes, with different goals and intentions, different intended users, and different applications. Shiffman et al. [38] describe the requirements to a guideline knowledge model as following:

- **Comprehensive**, i.e., capable of expressing all the knowledge contained in the guideline
- **Expressively adequate** to convey the complexities and nuances of clinical medicine while remaining informationally equivalent to the original guideline
- **Flexible**, i.e., a useful model must be able to deal with the variety and complexity of guidelines. The representation should permit modeling at high and low levels of granularity, so that guidelines can be interpreted at different levels of abstraction.
- **Comprehensible**, i.e., it should match the stakeholders’ normal problem-solving language and allow domain experts to describe their knowledge with little effort
- **Shareable** across institutions
- **Reusability** across all phases of the guideline life cycle

Figure 2.1 presents the history of various guideline modeling methods. We will describe the most significant in detail in the next sections.

2.1 Arden

The development of the Arden Syntax arose from the HELP system [23], which is an information management system, developed at the LDS Hospital in Salt Lake City. HELP has provided general decision support based on medical decision rules. The Arden Syntax for Medical Logic Systems is a language for encoding such rules [14]. In 1992 it became Standard of the American Society for Testing and Materials (ASTM) [1] and HL7-Standard in 1998.
Figure 2.1: History of guideline modeling methods. The guideline modeling methods are positioned on a time axis according to the time at which they started being developed. An arrow between two methods originates from a method that influenced the method depicted next to the arrowhead. (Adapted from [6].)
The aims for developing Arden Syntax were to create a standard knowledge representation syntax, to make medical knowledge portable and sharable between information systems and medical care providers, and to address the clinician to formalize its knowledge.

Arden encodes medical knowledge in knowledge base form as Medical Logic Modules (MLMs). An MLM is a hybrid between a production rule (i.e., an “if-then” rule) and a procedural formalism. Each MLM is invoked as if it were a single-step “if–then” rule, but then it executes serially as a sequence of instructions, including queries, calculations, logic statements, and write statements. A parallel execution of various MLMs is not possible.

The general structure of the Arden Syntax are slots within three categories: maintenance, library, and knowledge. Within each slot, different representation formalisms are taken. For example, in logic slot of knowledge category, if-then rules are used to represent the logic of the MLM invoked, while in purpose slot of library category, text is used to represent the purpose of the MLM. Following is an example of an MLM in Arden Syntax:

```
maintenance:
    title:
        CT study with contrast in patient with renal failure;
    filename:
        astm_ct_contrast;
    version:
        1.00;
    institution:
        ASTM E31.15; SMS;
    author:
        Harm Scherpbier, M.D.;
    specialist:
        ;
    date:
        1995-09-11;
    validation:
        testing;
library:
    purpose:
        Issue alert when physician orders CT study with contrast in patient with renal failure;
    explanation:
        If physician orders CT scan with contrast, this rule retrieves most recent serum creatinine. If the value is less than 1 week old, and more than 1.5, the system issues an alert to the physician to consider the possibility that his patient has renal failure, and to use other contrast dyes.;
    keywords:
        ;
    citations:
        ;
    links:
        ;
knowledge:
    type:
        data_driven;
```
data:
last_creat := read last {"Creatinine level"};
last_BUN := read last {"BUN level"};
;
evokes:
ct_contrast_order;;
logic:
if
  last_creat is null and last_BUN is null
then
  alert_text := "No recent serum creatinine available. Consider patient’s kidney function before ordering contrast studies.";
  conclude true;
elseif
  last_creat > 1.5 or last_BUN > 30
then
  alert_text := "Consider impaired kidney function when ordering contrast studies for this patient.";
  conclude true;
else
  conclude false;
endif;

action:
write alert_text || "
Last creatinine: " || last_creat || " on: " || time of last_creat || "
Last BUN: " || last_BUN || " on: " || time of last_BUN;
;
urgency: 50;
end:

Arden was developed for embedding MLMs into proprietary clinical information systems. MLMs are independent units in a health knowledge base containing maintenance information, links to other sources of knowledge (e.g., URLs, articles), and sufficient logic to make a single medical decision. Sequencing tasks can be modeled by chaining a sequence of MLMs. MLMs have been used to generate clinical alerts and reminders, interpretations, diagnoses, screening for clinical research studies, quality assurance functions, and administrative support. With an appropriate computer application (known as an event monitor), MLMs run automatically, generating advice where and when it is needed.

One intention, when developing the Arden Syntax, was that it has to be easy to read and learn (due to a language-like syntax). This is supported by the template (i.e., slots) that specifies the structure of every MLM and therefore facilitates the authoring of MLMs. Thus, the intended authors of MLMs are clinicians.

Arden brings particular support for time functions. It ensures that every data element and every event has a data/time stamp that is clinically significant. Various time functions are provided to help users specify the date and time in MLMs. FuzzyARDEN extends the Arden Syntax by concepts of fuzzy set theory, fuzzy logic, and fuzzy control to adequately represent rules with inherent linguistic uncertainty.
A problem that occurs with any form of clinical knowledge representation is the need to interact with a clinical database in order to provide alerts and reminders. Database schemata, clinical vocabulary, and data access methods vary widely so any encoding of clinical knowledge (such as a MLM) must be adapted to the local institution in order to use the local clinical repository. This hinders sharing of knowledge. Arden explicitly isolates references to the local data environment in curly braces ”{}” in a MLM, so this is sometimes called the ”curly braces problem”. EFForts are underway in HL7 to help solve this problem. Another potential limitation of Arden is that it does not explicitly define notification mechanisms for alerts and reminders. Instead, this is left to local implementation and is, like database queries, contained in curly braces in a MLM. Explicit notification mechanisms in the Syntax itself may be a part of a future edition. As well, ongoing development that will be part of a future edition is the representation of the Arden Syntax in XML-format.

Although Arden has been important and influential, it is recognized that to formalize complex decisions and care pathways or clinical workflow, a more expressive formalism will be needed. Thus, a number of languages has been developed that typically embed logical rules in higher order structures that represents tasks such as decisions, plans, and actions, which can be composed into time-oriented networks to represent guidelines and protocols. We will represent these in the next sections.

2.2 Asbru

Asbru is a task-specific and intention-based plan representation language, developed at Stanford Medical Informatics, Vienna University of Technology, University of Newcastle, and Ben Gurion University, that is used in the Asgaard project\footnote{In Norse mythology, Asgaard was the home of the gods. It was located in the heavens and was accessible only over the rainbow bridge, called Asbru (or Bifrost). For more information about the Asgaard project see http://www.asgaard.tuwien.ac.at.} to embody clinical guidelines and protocols as time-oriented skeletal plans \footnote{In Norse mythology, Asgaard was the home of the gods. It was located in the heavens and was accessible only over the rainbow bridge, called Asbru (or Bifrost). For more information about the Asgaard project see http://www.asgaard.tuwien.ac.at.}. Skeletal plans provide a powerful way to reuse existing domain-specific procedural knowledge, while leaving room for execution-time flexibility to achieve particular goals.

Asbru is designed to represent protocols rather than guidelines. Protocols are local tools that set out specifically what should happen, when and by whom in the care process. They can be seen as the local definition of a particular care process derived from a more generic guideline. Although, there is a difference between protocols and guidelines, it was proved that Asbru can represent not only protocols but also guidelines \footnote{In Norse mythology, Asgaard was the home of the gods. It was located in the heavens and was accessible only over the rainbow bridge, called Asbru (or Bifrost). For more information about the Asgaard project see http://www.asgaard.tuwien.ac.at.}.

Asbru was designed specific to the set of plan management tasks \footnote{In Norse mythology, Asgaard was the home of the gods. It was located in the heavens and was accessible only over the rainbow bridge, called Asbru (or Bifrost). For more information about the Asgaard project see http://www.asgaard.tuwien.ac.at.}. It enables the designer to represent both the prescribed actions of a skeletal plan and the knowledge roles required by the various problem-solving methods performing the intertwined supporting subtasks. The major features of Asbru are that

- Prescribed actions and states can be continuous
• Intentions, conditions, and world states are temporal patterns

• Uncertainty in both temporal scopes and parameters can flexibly be expressed by bounding intervals

• Plans might be executed in sequence, all plans or some plans in parallel, all plans or some plans in a particular order, or periodically

• Particular conditions and operators are defined to monitor the plans’ execution

• Explicit intentions and preferences can be stated for each plan separately

Basically, an Asbru plan can be seen as a template. This template gets instantiated whenever the plan gets executed. Furthermore, more than one instance might be created for a single plan. This pattern can be seen analog to the Class – Object relationship in Object Oriented Programming.

An Asbru plan consists of a plan name, arguments (e.g., time annotation), knowledge roles, and a plan body. Knowledge roles are preferences (i.e., constraints concerning the plan execution), intentions (i.e., goals that have to be achieved, maintained, or prevented), conditions (i.e., constraints concerning the transition between plan states and the switching between plans), effects (i.e., relation between plan arguments and measurable quantities), and plan layout (i.e., the order of executing subplans is defined).

The basic construct is the temporal pattern. One important part of such temporal pattern is the time annotation, which is represented in Figure 2.2.

Figure 2.2: Time interval in Asbru. The gray areas indicate the periods when the action has to start and accordingly finish.

Thereby, uncertainties exist in respect of the begin, the end, and the duration of the interval. Furthermore, no time points for the begin and the end are defined, but shifts from an arbitrary definable reference point. This point can individually be assigned for any interval.

Furthermore, we can differentiate among seven plan states in Asbru: considered, possible, ready, activated, suspended, completed, and aborted. An instantiated plan can have only one plan state at one time (see Figure 2.3).
For representing different relations among plans, Asbru provides several plan types, which describe the behavior concerning the execution of the plans and their synchronization [34]:

- **Sequential plans**: Subsequent plans can be activated if the preceding plan is finished. That means, its state must be completed or aborted.

- **Parallel plans**: Plans are activated at the same starting time. They do not need to be finished at the same time.

- **Any-order plans**: Only one plan can be in state activated at one time. Other plans must be in preselection phase, suspended, completed, or aborted state.

- **Unordered plans**: All plans are executed without any synchronization.

- **Subplans**: Plans can be nested. That means, a plan can invoke subplans, which are again subject of a particular ordering by assigning one of the above plan types. A subplan can be activated during its parent’s activated or suspended state and can last during its parent’s completed or aborted state.

Since a plan is represented in XML, it is basically human readable. But understanding such a plan needs a lot of training, semantic and syntactic knowledge about the representation language, is cumbersome, and surely not suited for physicians.

### 2.3 EON

Between 1996 and 2003 at Stanford University the EON model was developed. The EON guideline modeling and execution system forms part of the EON architecture, a component-based suite of models and software components for the creation of guideline-based applications [27].
EON includes an extensible suite of models to represent parts of a clinical practice guideline, domain ontologies, a view of patient data (virtual medical record), and other entities (e.g., those that define roles in an organization). The guideline model (called the Dharma model) defines guideline knowledge structures such as eligibility criteria, abstraction definitions, guideline algorithm, decision models, and recommended actions. The EON guideline execution system obtains patient data through a specified temporal database manager or from user input, and generates recommendations according to the contents of the specific guideline.

Encoding of EON guidelines is done in the Protégé-2000 (see Section 3.2.1) knowledge engineering environment (see Figure 2.4) [26]. The encoding process is facilitated by specialized views of the EON guideline model designed to satisfy specific requirements of different classes of guidelines. These requirements are conceptualised in terms of a set of guideline tasks (e.g., decision making, specification of work to be performed, interpretation of data, setting goals). A guideline developer using EON creates specialized views of the guideline model by selecting modeling solutions to these tasks.

![Figure 2.4: Modeling an EON guideline with Protégé.](image)

The general components used in the Dharma model to model guidelines are (1) a temporal model, (2) a concept model, (3) a patient data model, (4) an expression language, and (5) a guideline model (see Figure 2.5).
**Temporal Model.** The temporal model depicts a class hierarchy with various kinds of time stamps (e.g., time points, durations, time intervals). It admits absolute time points with fuzzy borders, relative time points (e.g., today), fuzzy durations, and time intervals.

**Patient Data Model.** This is a simplified model of patient information with focus on clinical and demographic information needed for clinical decision making. It is a flat hierarchy for easier mappings to relational databases. The data are time-stamped, interval-based, or static (see Figure 2.7).

**Medical Concept Model.** This model defines a taxonomic hierarchy of concepts (e.g., ACE inhibitor) used in guidelines, relationships among concepts (e.g., indications for ACE inhibitor), and supplies a controlled terminology for guideline encoding.

**Expression Language.** EON provides three models to specify expressions.

1. **Simple templates:** Simple expressions can be specified by predefined templates. Thereby, it is possible to apply boolean operators and combine various templates.

2. **Protégé Axiom Language (PAL):** By means of PAL it is possible to specify expressions directly in the language of Protégé. PAL uses the predicate language KIF (Knowledge Interchange Format).

3. **Temporal query and criterion language:** This is an SQL-related specification of expressions.
Figure 2.6: Structure of the EON temporal model.

Figure 2.7: Structure of the EON patient data model.
Guideline Model. In EON guidelines can be modeled by two different kinds of models. The first one is the Consultation Guideline, which is used to specify actions and decisions whose consequences are independent from temporal processes. The second one is the Management Guideline, which is used to model actions and decisions that change the patient state and are therefore critical in time.

Within the Guideline model clinical algorithms are described by scenarios, decisions, and action sequencing and synchronization.

In the EON guideline model, conditional goals are associated with guidelines and subguidelines. The guideline algorithm is represented as a set of scenarios (cf. Prodigy), action steps, decisions, branches, synchronisation nodes connected by a "followed-by" relation.

Furthermore, EON supports the reusability of medical domain knowledge, temporal queries, and abstractions.

Some of the work of EON is carried forward by the SAGE project.

2.4 GASTON

GASTON [5] is a generic architecture for the design, development, validation, and implementation of guideline-based medical decision support systems. It is developed by a joint effort of the Signal Processing Systems group of the Eindhoven University of Technology, the Department of Medical Informatics of the Maastricht University, and Medecs BV, Eindhoven. It was introduced in 1997 and is under continued development.

GASTON is a methodology and a framework that facilitates the development and implementation of computer-interpretable guidelines and guideline-based decision support systems. The overall goal of this approach is to improve the acceptance of computer-interpretable guidelines and decision support systems in daily care by facilitating all phases in the guideline development process.

The GASTON framework consists of (1) a guideline representation formalism that uses the concepts of primitives, Problem-Solving Methods (PSMs), and ontologies to represent guidelines of various complexity and granularity and different application domains; (2) a guideline authoring environment that enables guideline authors to define guidelines; and (3) a guideline execution environment that translates defined guidelines into a more efficient representation, which can be read in and processed by an execution-time engine.

The guideline representation formalism uses a frame-based model as an underlying mechanism. The formalism is non-monolithic, meaning that it can be extended with additional classes to capture new guideline characteristics.

Similar to the GLIF and EON approaches, the GASTON guideline authoring environment represents and visualizes guidelines by temporally sequenced graphs (flowcharts) of frame instances from the guideline model. Guidelines in GASTON consists of various layers (depending on the guideline’s complexity or application domain) that describe the control structure of a guideline (flow), its contents (e.g., actual decisions or actions), possible local adaptations, and communica-
tion/implementation details (e.g., the method of acquiring patient data or the form of decision support).

The GASTON approach defines various methods for the detection of various logical and procedural errors in guidelines. In addition, the framework also contains a simulation environment where developed guidelines and decision support systems can be tested and evaluated.

Finally, the framework contains a guideline execution environment that is able to execute guidelines and interface with external patient information systems. The execution environment consists of a core guideline execution engine, which can be extended with additional components (plugins) to communicate with patient information systems, medical databases and patient monitors.

GASTON has been used to develop, implement, and evaluate guidelines and guideline-based decision support systems in the areas of critical care, family practice, psychiatry, oncology, cardiology, and chronic disease management.

2.5 GEM

GEM (the Guideline Elements Model) [38] is an XML-based guideline document model that can store and organize the heterogeneous information contained in practice guideline documents.

GEM is under continued development at Yale University since 2000. It is an international ASTM standard for the representation of practice guidelines in XML format since 2002 [2].

The GEM project consists of the Guideline Elements Model itself, the editing tool Gem Cutter (see Section 3.1.3), and the quality evaluation method GEM-Q.

GEM is intended to facilitate the translation of natural language guideline documents into a standard computer interpretable format. It encodes considerable information about guideline recommendations in addition to the recommendations themselves, including the reason for each recommendation, the quality of evidence that supports it, and the recommendation strength assigned by the developers. For encoding guideline knowledge no programming knowledge is required, but a markup process is applied.

The authoring process for GEM guidelines takes place in three steps.

1. The GEM document, which has an XML-based syntax, is created based on the original guideline using the GEM Cutter. The elements of the GEM document are then stored in a relational design database.

2. Knowledge Customization: meta-information is added, the guideline can be locally adapted, and abstract concepts of the guideline can be implemented. This step is guided by the knowledge customization wizard.

3. Knowledge Integration into the clinical workflow depending on local circumstances.

GEM is intended to be used throughout the entire guideline lifecycle to model information pertaining to guideline development, dissemination, implementation,
and maintenance. Information at both high and low levels of abstraction can be accommodated. It preserves the intent of guideline developers by marking up the actual guideline language.

GEM is constructed as a hierarchy with more than 100 discrete tags and more than nine major branches (see Figure 2.8).

— Figure 2.8: Top level branches in GEM.

GEM proposes a document model for practice guidelines that can store and organize the heterogeneous information they contain. Although the elements identified could be added to most existing health services and informatics models, GEM describes concepts and knowledge more comprehensively than do other models. Three groups of elements are used to describe the content of a guideline in detail: recommendations, definitions, and algorithm (see Figure 2.9 for a detailed outline).

— Figure 2.9: Model of the knowledge components hierarchy.
**Recommendations.** They are the unique components that distinguish guidelines from other clinical publications and are intended to influence the practitioners’ behavior. When recommendations are analyzed into atomic concepts, they can be executed by a computer’s logic.

Recommendations can be categorized as conditional or imperative. Conditional concepts clearly delineate the situations in which they apply. Imperative statements are broadly applicable to the target population and do not force constraints on their relevance (e.g., *A major aspect of initial treatment should consist of lifestyle modifications, such as weight loss, reduction of salt and alcohol intake, ...*). They may parallel the actions in a conditional recommendation.

**Definitions.** They store important guideline terminology as well as the meaning of the terms and are both free text.

**Algorithm.** It can be graphically represented in flowcharts. This describes a temporal sequence of activities and the branching decision logic that implement the guideline’s recommendations. The GEM algorithm hierarchy includes four different elements related to GLIF. The *action step* specifies a clinical action that is to be performed in the patient-care process. The *conditional step* directs flow from one guideline step to another. The *branch step* directs flow in alternate directions. The *synchronization step* marks the point, where the different branch steps meet again.

The GEM guideline model tries to be comprehensive, expressively adequate, flexible, comprehensible, and shareable across institutions by the use of XML for knowledge representation and markup to provide unparalleled cross-platform compatibility. Furthermore, GEM supports the reuse across all phases of the guideline life cycle.

Currently, the development goes towards reusable methods (and creating tools) to facilitate guideline development and implementation using GEM. GEM II is also under development: the goal is to improve the comprehensiveness and usability of the model while maintaining backward compatibility.

The GEM format does not allow to execute guidelines in a computerized way by a execution engine, but it is possible to use GEM for generating tailored patient education materials [17]. Thereby, the information content is created suitable for tailored messages, along with a document structure needed to present them. A rules engine processes the converted GEM-based guideline and creates the tailored documents based on the patient data entered.

### 2.6 GLARE

GLARE stands for GuideLine Acquisition, Representation, and Execution and is a domain-independent system developed by the Dipartimento di Informatica, Università del Piemonte Orientale ”Amedeo Avogadro”, Alessandria, Italy.

GLARE technology consists of a representation formalism, a knowledge authoring tool, and a guideline execution tool [44]. The GLARE representation formal-
ism aims to balance expressiveness and complexity. GLARE provides consistency checking facilities during the guideline acquisition phase, supports the implementation of temporal reasoning [43], and provides a hypothetical reasoning facility.

The system is based on a modular architecture, which includes an acquisition tool and an execution tool.

The GLARE representation language is designed to achieve a balance between expressiveness and complexity. The formalism consists of a limited, but very focused and clearly understandable set of primitives. It is made up of different types of actions: plans (i.e., composite actions, hierarchically decomposable in their sub-actions) and atomic actions. Atomic actions can be query actions, decisions, work actions, and conclusions. Work actions refer to activities of care personal, similar to user-performed plans in Asbru or actions in PROforma. Query actions request information as does the ask-parameter in Asbru or enquiries in PROforma. Conclusions are the different outcomes of a decision process and resemble choices in GLIF or PROforma. Decisions are similarly modeled as in GLIF and PROforma. All actions are linked by control relations, which define the order in which they are performed: sequential, concurrent, alternative, or repetition [44].

GLARE provides expert physicians with an “intelligent” guideline acquisition interface. This provides different types of checks to help developing a consistent guideline: syntactic and semantic tests verify the “well-formedness” of a guideline. Further, extended Artificial Intelligence (AI) temporal reasoning techniques are used to check the consistency of temporal constraints imposed between actions [44].

**Decision support.** During guideline execution, a physician is often faced with the choice between alternative procedures. Clearly, a tool for collecting the pieces of information that are relevant to the current choice can play a fundamental role in the semi-automatic execution of a guideline. In many cases, decisions should not solely be taken on the basis of ”local information”, the criteria associated with the current specific decision. The choice may also need to take into account information associated with the alternatives on offer. GLARE allows for use of this ”global information” through a hypothetical reasoning facility [45] that enables users to gather relevant decision parameters (e.g., costs, resources, times) from selected parts of the guideline in a semi-automatic way. This hypothetical reasoning facility provides a way of simulating the consequences of choosing different alternative paths through a guideline. This can be particularly useful when a physician has to choose among different therapeutic procedures and the best one for a specific patient is not obvious. A GLARE-based guideline can take a clinician through the different implications of each treatment choice.

**Temporal reasoning.** The GLARE formalism is able of specifying the temporal issues involved in developing and executing a clinical guideline. In most therapies, actions have to be performed according to a set of temporal constraints determining the order in which they are carried out, their duration, and the intervals (cyclical or non-cyclical) between them. The GLARE representation formalism is designed to
cope with different types of temporal constraints needed to manage clinical guidelines, and specialized temporal reasoning algorithms operating on them. Temporal reasoning can be useful both when the guideline is being developed, and when it is being executed [43].

GLARE technology has been successfully tested in different clinical domains (bladder cancer, reflux esophagitis and heart failure), at the Laboratorio di Informatica Clinica, Azienda Ospedaliera S. Giovanni Battista, Torino, Italy.

2.7 GLIF

GLIF (the Guideline Interchange Format) [28] is a computer-interpretable language for modeling and executing clinical practice guidelines. GLIF was developed by the InterMed Collaboratory at Stanford Medical Informatics, Harvard University, McGill University, and Columbia University, and supports sharing of computer-interpretable clinical guidelines across different medical institutions and system platforms. GLIF has a formal representation. It defines an ontology for representing guidelines, as well as a medical ontology for representing medical data and concepts. Tools are under development to support guideline authoring and execution.

GLIF2 supported guideline modeling as a flowchart of structured steps, which represented clinical actions and decisions. However, the attributes of these constructs were defined as text strings that could not be parsed, preventing the resulting guidelines from being able to make inferences during computerized execution.

GLIF3 is a new version of GLIF designed to support computer-based execution. GLIF3 includes a formal expression languages for specifying decision criteria and patient states named GELLO [39]. GELLO, an object-oriented expression language, is extensible and allows implementation of expressions that are not supported by the Arden Syntax. Besides this, a layered patient data model to refer to patient data items defined by a controlled terminology that includes standard medical vocabularies (such as UMLS), as well as standard data models for medical data is implemented.

GLIF3 enables guideline encoding at three levels of abstraction. In a conceptual level the guideline is presented as a flowchart without full details of decisions. The second level, a computable specification that can be verified for logical consistency and completeness, is reached by adding decisions, patient data, and iteration information. The third level, an implementable specification that can be incorporated into particular institutional information systems, is reached by mapping actions to institutional procedures at a certain site and patient data references to a certain electronic patient data record.

GLIF3 has been designed to support computer-based guideline execution: it has a computable level of specification, which formally defines logical criteria, definitions of patient data items, clinical actions and the flow of the guidelines. The computable level of the specification may be regarded as coming between the abstract flowchart level and the implementation level. The abstract flowchart level helps authors and users view and understand a guideline. The implementation level
includes non-shareable, institution-specific details, which enable guidelines to be incorporated into operational clinical information systems. Shareable components of a guideline are therefore explicitly separated from institution-specific or vendor platform-specific (non-shareable) components.

For representing the guideline’s content the guideline’s algorithm is composed of guideline steps. These can be either action, decision, branch, synchronization, or patient state.

**Action.** This step specifies a set of tasks to be performed and may contain several attributes (e.g., iteration information, duration, etc.). Actions can be nested. There are two types of actions: medically-oriented actions, which specify a medical task as defined in the Reference Information Model (RIM) layer of GLIF’s data model, and programming-oriented actions.

**Decisions.** These steps direct the control flow between alternative steps. There are two types of decision steps: case steps, which model deterministic decisions, and choice steps, which model non-deterministic decisions.

**Branch.** This step models concurrency of multiple guideline steps, which may be performed in parallel or in any order.

**Synchronization.** These steps are used in conjunction with branch steps. They mark the place where the different branches of execution meet again and specify the conditions to proceed (i.e., whether all, some, or one of the preceding steps must have been completed before continuing).

**Patient State.** This step labels its position in the guideline for two purposes. On the one hand, it shows the progress of the patient state. On the other hand, it serves as an entry point of the guideline. This means that guidelines can be started at any place, which contains a patient state.

Current work focuses on two topics: (1) development of GLEE, an execution engine for GLIF [53], and the implementation of GLIF-encoded guidelines in hospital settings, as well as (2) management of versioning of computer-interpretable guidelines [29].

### 2.8 GUIDE

Guide [4] is a component-based multi-level architecture designed to integrate a formalized model of the medical knowledge contained in clinical guidelines and protocols with both workflow management systems and Electronic Patient Record technologies. It is developed by the Laboratory for Medical Informatics, Department of Computer and System Science, University of Pavia, Italy.
The *Guide* environment (see Figure 2.10) integrates three main modules:

1. Guideline Management System (GlMS), which provides clinical decision support
2. Electronic Patient Record (EPR), which provides access to patient data
3. Workflow Management System (WfMS) or Careflow Management System (CfMS), provides organizational support

![Diagram of Guide environment](image)

**Figure 2.10: Guide environment – high level architecture.**

Communication among the three modules uses a loosely coupled, message-based system, which reduces crosstalk and supports parallelization.

*Guide* aims to provide an integrated medical knowledge management infrastructure, providing support for:

- Computerized knowledge representation
- Different views of the formalized knowledge to allow different people with different roles (e.g., clinicians, patients, administrators) to have their own context-specific interactions with the system (e.g., if a guideline for a chronic condition suggests taking a blood sample every fifteen days, the physician view would incorporate the interpretation of the examination results, while the patient view would provide a reminder and a facility to book the blood examination)
- The use of formalized knowledge for generating health care processes able to respond to environment stimuli and patient condition mutations (connection to the electronic patient record and to the workflow system)
- The generation of new knowledge (elicitation of tacit knowledge) through continuous feedback on guideline acceptance, usability and compliance. Since this is a crucial aspect of the system, Guide is able to manage flexible health care pathways and user interaction.
- Reuse and sharing of statically and dynamically generated knowledge components
Guide can use standard terminologies like LOINC\(^2\) (the Logical Observation Identifiers Names and Codes) and ICD9 (the International Classification of Diseases).

The Guide model is based on Petri Nets. The strength of the formalism, when applied to healthcare, is its ability to support the modeling of complex concurrent processes (sequential, parallel, and iterative logic flows). The formalism has been extended to support improved modeling of time, data, and hierarchies.

Guideline authoring for Guide is done with the Guide Editor.

### 2.9 Prestige

Prestige’s approach, derived from the DILEMMA project and other earlier work in clinical knowledge-bases systems, uses an explicit, declarative representation format in knowledge bases, which facilitates the adaptation, flexible use and maintenance of clinical knowledge. This approach to knowledge representation requires an explicit knowledge model defining common and standard formats for computerized representation and use of items of knowledge. The Prestige models incorporate and refine the DILEMMA Generic Protocol Model, which offers a common electronic format for representing all clinical guidelines across specialties, user sites and application software platforms.

Prestige was a project for applying telematics to assist the dissemination and application of clinical practice guidelines and protocols.

The Prestige Conceptual Guideline Model describes the concepts and the relationships between them needed to support the use of guidelines in the planning and provision of personal healthcare. This implies a wide-ranging ability to represent knowledge about medicine, patients and carers, and the enterprises and personnel that care for them. It has proved possible to produce such a model that is portable, that means, is applicable in any organization using any technology.

It provides an essential part of a meta-language for those responsible for writing the generic clinical scripts for particular scenarios, the protocol authors.

The model has two major subdivisions. The first describes healthcare in general, and the second focuses on clinical protocols. The model has been built and stored using a proprietary object-oriented CASE tool, which can produce skeleton software if required.

### 2.10 PRODIGY

PRODIGY is a computer-based decision support system (for prescribing in particular) that integrates with commercial primary care information systems in England. PRODIGY phase 3 incorporated support for chronic disease management. It is developed by the Sowerby Centre for Health informatics in Newcastle (SCHIN). It was introduced in 1996 and is under continued development.

PRODIGY is a guideline-based decision support system in use by a large number of general practitioners in the UK. The PRODIGY 3 model was created to model

\(^2\)http://www.loinc.org
guidelines for the management of chronic diseases, such as asthma, hypertension, and angina, in primary care. Essentially, it supports guideline modeling a series of decisions that a general practitioner may have to make in different patient encounters. The model enables a guideline to be organized as a network of patient scenarios, management decisions, and action step, which produce further scenarios [15]. Scenarios are patient states defined by the patient’s condition and current treatment. Scenarios are associated with:

- A consultation template that describes the best-practice workup for a patient in that scenario;
- A choice between alternative courses of action.

The PRODIGY3 decision model uses rule-in and rule-out conditions associated with each available alternative to determine the preferred course of action. Decisions in PRODIGY always require confirmation. This reflects the philosophy that the autonomy of clinicians should be maintained.

The management over time of a patient according to a guideline specification can be viewed as the traversal of a number of selected scenarios and associated actions and further decision points along a single path. Sequencing of actions is achieved by defined followed-by relations.

Guideline authoring for PRODIGY is done with the Protégé knowledge-acquisition tool (see Section 3.2.1).

### 2.11 PROforma

PROforma [41] is a formal knowledge representation language capable of capturing the structure and content of a clinical guideline in a form that can be interpreted by a computer. It is developed by the Advanced Computation Laboratory, Cancer Research UK since 1992. It is the basis of both a method and a technology for developing and publishing executable clinical guidelines. Applications built using PROforma software are designed to support the management of medical procedures and clinical decision making at the point of care. One aim of the PROforma project is to explore the expressiveness of a deliberately minimal set of modeling constructs.

In PROforma, a guideline application is modeled as a set of tasks and data items. The PROforma task model (Figure 2.11) divides from the keystone (generic task) into four types: plans, decisions, actions, and enquiries.

**Plans** are the basic building blocks of a guideline and may contain any number of tasks of any type, including other plans. **Decisions** are taken at points where options are presented, e.g., whether to treat a patient or carry out further investigations. **Actions** represent some procedure that needs to be executed in the external environment (such as the administration of an injection). **Enquiries** are typically requests for further information or data, required before the guideline can proceed.

All tasks share attributes describing goals, control flow, pre-, and post-conditions. The simple task ontology should make it easier to demonstrate soundness and to teach the language to encoders.
Figure 2.11: The PROforma task model.

Figure 2.12: Task states and state transitions.
PROforma processes may be represented diagrammatically as directed graphs in which nodes represent tasks and arcs represent scheduling constraints. All tasks have a state property, which can take four different values (cp. Figure 2.12). The initial state is dormant. The usual interpretation of the task states is that a task is dormant if it has not been started, and it is not yet possible to say whether it will be started, in progress if it has been started, discarded if the logic of the guideline implies either that it should not be started or that it should not be completed, and completed if it has been done.

PROforma software consists of a graphical editor to support the authoring process, and an engine to execute the guideline specification. The engine can also be used as a tester during the application development phase.

A PROforma editor supports the construction of a guideline in terms of the four task types. Using the icons shown in the diagram, networks of tasks can be composed that represent plans or procedures carried out over time. In the editor, logical and temporal relationships between tasks are captured naturally by linking them as required with arrows. Any procedural and medical knowledge required by the guideline as a whole or by an individual task is entered using templates attached to each task. The resulting populated graphical structure is automatically converted into a database ready for execution.

2.12 SAGE

The Standards-based Shareable Active Guideline Environment (SAGE) was introduced in 2002. It is developed by IDX Systems Corporation, Apelon Inc., Intermountain Health Care, Mayo Clinic, Stanford Medical Informatics, and University of Nebraska Medical Center.

Three considerations led to the decision to start development of the SAGE model. Past efforts have gone into developing shared models for representing medical decisions and clinical guidelines. However, it takes more than a formalism for medical logic to accomplish sharing of computable medical knowledge. Reuse of a guideline knowledge base also requires that an infrastructure that includes medical record query interface, terminology mediation, and act interface is in place. The clearing of the complex relationship between existing standards (e.g., HL7, SNOMED) and requirements of guideline modeling and deployment is one of the themes of the SAGE project.

The second consideration is the SAGE’s approach to integrating guideline-based decision support with the workflow of care process. The success of clinical decision-support systems (DSSs) depends heavily on how the system is integrated into the care process. Therefore, SAGE will not be in control of a host systems’ workflow management. Thus, the SAGE modeling approach does not require detailed workflow to be modeled. Instead, the system will respond to opportunities for decision support in the care process.

The third consideration leading to the decision to develop SAGE is that, in recent years, much interchange and cross-fertilization have taken place in the guideline modeling community. The SAGE project gives the opportunity to synthesize prior
work and, wherever possible, to establish mappings between the SAGE model and other models.

In summary, the SAGE project seeks to create a guideline model that

- Uses standardized components that allow interoperability of guideline execution elements with the standard services provided within vendor clinical information systems
- Includes organizational knowledge to capture workflow information and resources needed to provide decision-support in enterprise setting
- Synthesizes prior guideline modeling work for encoding guideline knowledge needed to provide situation-specific decision support and to maintain linked explanatory resource information for the end-user.

The SAGE model organizes guideline recommendations as recommendation sets consisting of either Activity Graphs that represent guideline-directed processes or Decision Maps that represent recommendations involving decisions at a time point. Within a particular context, a recommendation may either describe the preferred choice in a management decision or it may recommend a series or actions to be carried out.

SAGE uses a suite of data models and services as interfaces to clinical information systems to achieve interoperability of guideline decision support systems (DSS). It uses standard terminologies and a deployment-driven guideline modeling methodology.

![Figure 2.13: Steps in modeling clinical practice guidelines for integration into workflow. The arrows represent information flow.](image)

The SAGE guideline knowledge base development methodology consists of six main steps (cp. Figure 2.13).

1. Clinicians must create clinical scenarios that are detailed enough to support integration of executable guideline content into real clinical workflow. For
each scenario, user-interface screens are created to simulate the interactions between care providers and the clinical information system.

2. Clinicians analyze the information content of the desired guideline recommendations and extract the knowledge and logic needed to generate these recommendations from guideline texts, medical literature, and clinical expertise. The extraction process requires clinicians to select, interpret, augment, and operationalize guideline statements to disambiguate concepts.

3. Clinical concepts used in the extracted guideline logic are identified.

4. Concepts identified as part of the required guideline logic are instantiated as detailed data models that correspond to constraints on classes of ”virtual medical record” (vMR). The vMR supports a structured data model for representing information related to individual patients, domains for values of attributes in the data model, and queries through which guideline DSS can test the states of the patient.

5. Guideline concepts in terms of standard terminologies are specified. To implement a computerized guideline in a particular institution, terms used in a guideline knowledge base to describe patient states must be mapped to terms in that institution’s electronic patient record. Standard terminologies, such as SNOMED CT (the Systematized Nomenclature of Medicine Clinical Terms) and LOINC, provide the necessary shared semantics for such mappings.

6. Clinical scenarios and guideline logic are translated into a computer-interpretable model of guidelines. The SAGE methodology calls for explicit modeling of guideline usage as part of the executable guideline specification. Thereby, it assumes that a guideline does not dictate the workflow in a clinic, but the guideline knowledge base specifies how a DSS reacts to events in the care process.

When encoding a guideline for SAGE, clinical experts must interpret the guideline statements and create one or more plans that will support the guideline goals in the specific work environment of their health care organization. To achieve this, the recommendation set employs four ”nodes”: Context, Decision, Action, and Route.

1. **Context Node.** Specifies and declares the assumptions made about the health care enterprise work model that are otherwise implicit in every instance of a guideline implementation; their defining attributes specify their trigger events, clinical setting, and patient state.

2. **Action Node.** Models one or more information system activities employed in support of a recommendation set.

3. **Decision Node.** Describes the acquisition of some data (directly from the patient EMR or interactively by asking the clinician) and the employment of a decision model to evaluate branching logic.

---

3[http://www.snomed.org](http://www.snomed.org)

The Activity Graph can be modeled by means of the Protégé-2000 environment (see Section 3.2.1).

### 2.13 Discussion

We have described various guideline modeling languages and frameworks that have been developed for different purposes and with different aims and intentions. This leads us to comparing some of the languages to discuss the differences and the main focuses of the various approaches. Peleg et al. [30] showed in their comparison study of various guideline modeling methodologies the different attempts. Table 2.1 shows the terms that are used for concepts of plans and actions.

The various languages and frameworks also have different approaches in other concepts (see Table 2.2).

- The specification of goals or intentions is done as text strings by Prodigy, GLARE, and GLIF. This allows clinicians to browse goals of a guideline, but impedes machine reasoning about these goals. Asbru, EON, Guide, and PROforma represent them as formal expressions and allow reasoning about goals.

- Asbru and PROforma are the only modeling languages that support the expression of the effects of a plan and thus allow reasoning about plans based on these effects.

- All methodologies recognize that user intervention or confirmation may be required during the decision-making process. All except Prodigy allow some decisions to be automatically made by the guideline execution system. Decisions in Prodigy always require confirmation.

- Asbru and EON can use systems that perform temporal abstractions to abstract clinical conditions that hold over an interval of time, based on raw, time-stamped values.

- Prodigy, EON, and GLIF can create hierarchies of medical concepts and reason about them by writing expressions that utilize the concept hierarchies. In Guide, each guideline task can be associated with a single SNOMED code that represents a clinical task. Also, SAGE has the ability to use standard terminologies like SNOMED CT and LOINC. In GLARE basic attributes of the entities in a given domain, such as medicine, and at some of the basic and most frequently recurring entities in the domain (e.g., diagnosis) can be described by an ontology.

Asbru and PROforma have deliberately not attempted to include methods for the representation of static knowledge such as medical concept models and ontologies of actions, for example. Instead, they emphasize the provision of clean interfaces to access such information held externally.
Table 2.1: Terms used by guideline modeling methodologies to refer to plans and actions (adapted from [30]).

<table>
<thead>
<tr>
<th>Model</th>
<th>Plan</th>
<th>Plan Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asbru</td>
<td>Plan</td>
<td>Branching: Plan type, Plan precondi-tion</td>
</tr>
<tr>
<td>EON</td>
<td>Management guidelne</td>
<td>Branch synchronization, Action, Decision, Scenario</td>
</tr>
<tr>
<td></td>
<td>Consultation guideline</td>
<td>Consultation action, Consultation branch, Consultation guideline part of scenario</td>
</tr>
<tr>
<td>GEM</td>
<td>Guideline</td>
<td>Branch step; Synchronization step, Action; Directive; Action step, Conditional recommendation; Conditional step</td>
</tr>
<tr>
<td>GLARE</td>
<td>Plan</td>
<td>Work action; Query action, Work action; Query action</td>
</tr>
<tr>
<td>GLIF</td>
<td>Guideline, Macro</td>
<td>Branch synchronization, Action, Decision</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Patient state, Guideline or Macro called in Action or Decision steps</td>
</tr>
<tr>
<td>GUIDE</td>
<td>Guideline</td>
<td>Synch-AND, Synch-OR, Task</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Deterministic decision, non-deterministic decision, Wait monitor</td>
</tr>
<tr>
<td>Prodigy</td>
<td>Decision/Manage-ment map</td>
<td>Action</td>
</tr>
<tr>
<td></td>
<td>Consultation Template</td>
<td>Consultation branch, Consultation action</td>
</tr>
<tr>
<td>PROforma</td>
<td>Plan</td>
<td>Action, Enquiry, Decision</td>
</tr>
<tr>
<td>SAGE</td>
<td>Recommendation</td>
<td>Decision, Routing</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Special</th>
<th>Subplan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asbru</td>
<td>Subplan step</td>
<td></td>
</tr>
<tr>
<td>EON</td>
<td>Subguideline step</td>
<td></td>
</tr>
<tr>
<td>GEM</td>
<td>Consultation guideline part of scenario</td>
<td></td>
</tr>
<tr>
<td>GLARE</td>
<td>Subaction step</td>
<td></td>
</tr>
<tr>
<td>GLIF</td>
<td>Guideline or Macro called in Action or Decision steps</td>
<td></td>
</tr>
<tr>
<td>GUIDE</td>
<td>Any task can be decomposed</td>
<td></td>
</tr>
<tr>
<td>Prodigy</td>
<td>Subguideline step or called in action step</td>
<td></td>
</tr>
<tr>
<td>SAGE</td>
<td>Subguideline step</td>
<td></td>
</tr>
</tbody>
</table>

29
### Table 2.2: Different approaches of the guideline modeling methodologies.

<table>
<thead>
<tr>
<th>Concept</th>
<th>Asbru</th>
<th>EON</th>
<th>GLARE</th>
<th>GLIF</th>
<th>Guide</th>
<th>Prodigy</th>
<th>PROforma</th>
<th>SAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Representation of static knowledge</td>
<td></td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td></td>
<td>•</td>
</tr>
<tr>
<td>Formal representation of clinical goals</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td></td>
<td>•</td>
</tr>
<tr>
<td>Confirmation of decisions</td>
<td></td>
<td></td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Representation and reasoning with effects of actions</td>
<td></td>
<td></td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Temporal abstractions</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chapter 3

Tools for Formalizing Computer-Supported Clinical Guidelines and Protocols

Due to the complexity of most of the guideline representation languages methods and tools were developed to assist in the formalization process of CGPs (see Table 3.1 for details).

Table 3.1: Different approaches of the guideline modeling methodologies.

<table>
<thead>
<tr>
<th>Model</th>
<th>Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asbru</td>
<td>AsbruView; DELT/A; Stepper; URUZ (Degel); PBW (Degel)</td>
</tr>
<tr>
<td>EON</td>
<td>Protégé</td>
</tr>
<tr>
<td>GASTON</td>
<td>Weaning Guideline Editor</td>
</tr>
<tr>
<td>GEM</td>
<td>GEM Cutter</td>
</tr>
<tr>
<td>GLARE</td>
<td>Guidelines Development Environment</td>
</tr>
<tr>
<td>GLIF</td>
<td>Protégé</td>
</tr>
<tr>
<td>Guide</td>
<td>Guide Editor</td>
</tr>
<tr>
<td>Prestige</td>
<td>GAUDI; GLEAM</td>
</tr>
<tr>
<td>Prodigy</td>
<td>Protégé</td>
</tr>
<tr>
<td>PROforma</td>
<td>Arezzo; Tallis; Protégé</td>
</tr>
<tr>
<td>SAGE</td>
<td>Protégé</td>
</tr>
</tbody>
</table>

3.1 Markup-based Tools

3.1.1 Document Exploration and Linking Tool / Addons (DELT/A)

The Institute of Software Technology and Interactive Systems at the Vienna University of Technology is developing a tool to provide a relatively easy way to translate
free text into various (semi-)formal, XML-based representations. It achieves this by displaying both the original text and the translation, and showing the user which parts of the formal code correspond to which elements of the original text. This not only makes it easier to author plans, but also to understand the resulting constructs in terms of the original guideline.

DELT/A (formerly known as Guideline Markup Tool – GMT) [21, 51, 52] provides two main features: (1) linking between a textual guideline and its formal representation, and (2) applying design patterns in the form of macros.

DELT/A allows the definition of links between the original guideline and the target representation, which gives the user the possibility to find out where a certain value in the XML-language notation comes from. Therefore, if someone wants to know the origin of a specific value in the XML file DELT/A can be used to jump to the correlating point in the text file where the value is defined and the other way round.

The second feature of DELT/A is the usage of macros. A macro combines several XML elements, which are usually used together. Thus, using macros allows creating and extending specific XML files more easily through the usage of common design patterns.

Figure 3.1: The Document Exploration and Linking Tool / Addons (DELT/A).

DELT/A supports the following tasks:
**Authoring and augmenting guidelines.** We want to be able to take a new guideline in plain text and create an (XML-based) representation of it, and to add links to the corresponding parts of a guideline to an already existing XML file.

**Understanding the (semi-)formal representation of guidelines.** For a guideline in a (semi-)formal representation, we want to be able to see where values in the different parts of the representation’s code come from, and how parts of the original text were translated into it. This is important not just for knowledge engineers, but also for physicians wanting to get an understanding of the language.

**Structuring the syntax of the (semi-)formal representation.** DELT/A provides a structured list of elements of the target language – the macros – that need to be done in a way that best supports the authoring of plans. This list will also provide a good starting point for teaching material and possible subsets of the language for special purposes.

By means of these features the original text parts need not be stored as part of the target representation elements. The links clearly show the source of each element in the target representation. Additionally, there is no need to produce a guideline in natural language from the target representation, since the original text remains unaltered.

The DELT/A user interface (see Figure 3.1) consists of various panes. The top left and right panes provide equivalent views to either edit XML files or HTML files. The Macros pane provides either a structure view, search view, or insertable macros view, as well as a preview of the current macro.

### 3.1.2 Stepper

*Stepper* [42][33] is a mark-up tool for narrative guidelines, developed by the EuroMISE centrum – Kardio and the University of Economics, Prague, Czech Republic. Its development started in 2001; a first beta version introduced in 2003. The Stepper project has two main goals:

1. To develop a stepwise method for formalization (in this context, XML transformation) of text documents of clinical guidelines

2. To develop the Stepper tool, an XML editor enhanced with features to support the above method

Stepper has been designed as a document-centric tool, which takes a guideline text as its starting point and splits the formalization process into multiple user-definable steps, each of which corresponds to an interactive XML transformation. The result of each step is an increasingly formalized version of the source document. An embedded XSLT processor carries out non-interactive transformation. Both the mark-up and the iterative transformation process are carried out by rules expressed in a new transformation language based on XML, the so-called XKBT.
XML Knowledge Block Transformation). This resulted, as the well-known standard for transformation XSLT did not solve all problems when explicitly expressing transformations of knowledge in each step. Hence, a tailored transformation language was developed.

The transformation process with Stepper consists of six steps:

1. **Input text format.** The format of the original guideline text is XHTML, the XML version of HTML.

2. **Coarse-grained semantic mark-up.** Basic blocks of the text are marked (e.g., headings, sentences) and parts without operation semantics are removed.

3. **Fine-grained semantic mark-up.** Complex sentences are rearranged into simpler ones and background knowledge is added. In addition, a data dictionary is created, which describes the clinical parameters involved.

4. **Universal knowledge base.** The original document is transformed into a universal knowledge base. This involves changing the structure of the document to achieve modularity, which is assumed to involve medical experts in part.

5. **Export-specific knowledge base.** The representation is adapted to ease the export to the target representation. Therefore, an export-specific knowledge base is produced from the universal one.

6. **Target computational representation.** The ultimate format is produced by the knowledge engineer. This step is assumed to be performed fully automatically using XSL style sheets.

By using the Stepper method and tool it is possible to transform CGPs into fragments of operational code (e.g., Java) or into parts of a guideline representation language (e.g., Asbru).

Stepper’s main advantage is the documentation of all activities. So other users can easily review the transformation process. Stepper also provides an interface showing the interconnection between the source text and the model.

### 3.1.3 GEM Cutter

The *GEM Cutter* [31] is a tool with the aim to facilitate the transformation of CGPs into the GEM format. It is developed by Yale Center for Medical Informatics at Yale University School of Medicine. GEM (the Guideline Elements Model – see Section 2.5) is an XML-based guideline document model.

GEM Cutter’s Main Screen consists of three vertical segments (cp. Figure 3.2). In the left pane – the *Guideline Text Segment* – you see the original text of the guideline, in the middle pane – the *GEM Tree Segment* – a tree view of the developing GEM file is displayed, and in the right pane – the *Element Segment* – there are additional, important information about the GEM file.

From the Guideline Text Segment text is copied into elements in the GEM Tree Segment. Text that has been copied is marked underlined. The GEM Tree Segment
displays the contents of the GEM document in tree structure format. Each item on the tree represents a GEM Element, which can contain text and can be edited. The Element Segment consists of various parts: the Element Name Bar, the Element Source Bar, the Element Text Box, and the Element Definitions Box. These contain additional information of the element selected in the GEM document.

3.1.4 Degel – Digital Electronic Guideline Library

Degel [37] is a generic framework with tools to support guideline classification, semantic markup, context-sensitive search, browsing, run-time application, and retrospective quality assessment. It is applicable for any XML-based guideline representation, currently supporting Asbru [36] and GLIF [28]. It supports the gradual migration of free text guidelines to formal representations.

Semantic markup is performed using the Uruz web-based guideline markup tool, which resembles the DELT/A but does not maintain links between different representations of the guideline. Uruz can also be used to create a guideline document de-novo (i.e., without using any source) by directly writing into the knowledge roles of a selected target ontology. The editor can modify the contents or add new content. This enables turning implicit knowledge into more explicit, further facilitating the task of the knowledge engineer who fully formalizes the guideline.

Several features are especially tailored to Asbru, such as the plan-body wizard (PBW), which is used for defining the guideline’s control structure. The PBW enables a user to decompose the actions embodied in the guideline into atomic actions and other sub-guidelines, and to define the control structure relating them (e.g., sequential, parallel, repeated application). The PBW, used by medical experts, significantly facilitates the final formal specification by the knowledge engineer.
To be truly sharable, guidelines need to be represented in a standardized fashion. Thus, Uruz enables the user to embed in the guideline document terms originating from standard vocabularies, such as ICD-9-CM for diagnosis codes, CPT-4 for procedure codes, and LOINC-3 for observations and laboratory tests. In each case, the user selects a term when needed, through a uniform, hierarchical search interface to our Web-based vocabulary server.

### 3.2 Graphical Tools

#### 3.2.1 Protégé

Protégé[^1] is an open source ontology development and knowledge acquisition environment, developed by Stanford Medical Informatics. It was introduced in 1989 and is under continued development.

Protégé is a Java tool, which provides an extensible architecture for the creation of customized knowledge-based tools. It assists users in the construction of large electronic knowledge bases. It has an intuitive user interface that enables developers to create and edit domain ontologies and supports customized user-interface extensions, incorporates the Open Knowledge Base Connectivity (OKBC) knowledge model, interacts with standard storage formats such as relational databases, XML, and RDF. Protégé is

- A ‘meta-tool’ that helps users construct domain-specific knowledge acquisition systems that application experts can use to enter and browse the content knowledge of electronic knowledge bases
- A knowledge-base editing tool, which supports: constructing a domain ontology; designing customized knowledge-acquisition forms; entering domain knowledge
- A platform, which can be extended with graphical widgets for tables, diagrams, animation components to access other knowledge-based systems embedded applications
- A library, which other applications can use to access and display knowledge bases

Protégé is also used to author guidelines in various models (e.g., EON, GLIF, Prodigy, Proforma). See Figure 3.3 for an example.

#### 3.2.2 AsbruView

Information Visualization is the use of computer-supported interactive visual representations of abstract data to facilitate cognition: The Asgaard/Asbru project

[^1]: http://protege.stanford.edu
Figure 3.3: View of Protégé being used to author a guideline for managing chronic cough. The guideline model being used in this application is Dharma, part of the EON framework.

focuses on data and plan visualization during the design and execution of guideline and protocol applications. They developed an graphical user interface to Asbru, which support the development of guidelines and protocols, called AsbruView [25, 22, 19, 20]. Different methods are in development, which are dealing with the visualization of data and plans during the execution phase and are utilizing Focus+Context techniques (e.g., Semantic Depth of Fields (SDOF)).

Asbru is a complex language, which cannot be understood by physicians, who have no or hardly any training in formal methods. AsbruView is a tool to make Asbru accessible to physicians, and to give any user an overview over a plan hierarchy. AsbruView is based on visual metaphors to make the underlying concepts easier to grasp. This was done because not only is the notation foreign to physicians, but also the underlying concepts.

AsbruView consists of four views: Topological View, Temporal View, SOPO-View, and XML View.

1. The Topological View mainly displays the relationships between plans, without a precise time scale. The basic metaphor in this view is the running track (see Figure 3.4).

2. The Temporal View concentrates on the temporal dimension of plans and conditions. In addition to the topological information, physicians need to be able to see the details of the temporal extensions of plans. For this purpose, the temporal view is used. It consists of a display that represents each plan with a so-called glyph, i.e., a graphical object whose features change with the values they depict (see Figure 3.5).

3. The SOPOView is a different view to capture the temporal dimensions of plans. Originally, sets of possible occurrences (SOPOs) were designed for
the easy graphical propagation of temporal constraints and not for making
complicated notion of time easy to understand. Essentially, a diagram with
two time axes is used that represent the begin and end times of an interval,
respectively (cp. Figure 3.6).

4. The **XML View** shows the Asbru 7.3 code of the plans. It describes not only the
temporal aspects of the plans, but also the conditions, preferences, intentions,
and effects.

The metaphors and graphical representation of AsbruView have proved to be useful
in communicating Asbru’s concepts to physicians. Users get a better overview of
the therapy steps than from tables, while at the same time being able to see the
precise temporal constraints of plans (which they do not with flow-charts).

### 3.2.3 Arezzo

The first implementation of software to create, visualize, and enact PROforma (see
Section 2.11) guidelines was Arezzo. Arezzo consists of two main modules: (1) the
Composer and (2) the Performer (see Figure 3.7).

The **Composer** is a graphical editor or knowledge authoring tool, which uses
PROforma notation to capture the structure of a guideline and generate an exe-
cutable specification (see Figure 3.8 for an overview). The Composer CASE tool is
the developer’s GUI. The building blocks used to construct a guideline of any level
Figure 3.6: Part of an example screenshot of SOPOView.

Figure 3.7: Overview of the Arezzo application.
of complexity are the four PROforma task types. Data items and their properties to be collected during protocol enactment are also defined using the Composer (see Figures 3.9 and 3.10). The Composer also functions as a tester: guideline applications can be run, debugged, and validated within this module.

The Performer tests and executes guidelines defined in the PROforma language. The Performer interprets the guideline specification and during guideline enactment it prompts the user to perform actions, collect data, carry out procedures, and make decisions as required. During enactment, the Performer also maintains a local patient database, which is queried by the engine to evaluate specified conditions.

### 3.2.4 Tallis

Tallis [40] is a new Java implementation of PROforma-based authoring and execution tools developed by Cancer Research UK. Tallis is based on a later version of the PROforma language model. It consists of a Composer (to support creation, editing, and graphical visualization of guidelines), Tester, and Engine (to enact guidelines and allow them to be manipulated by other applications). Tallis is also designed for delivering web-based services; applications will run on any platform and integrate with other components, including 3rd party applications.
Figure 3.9: AREZZO Composer. The Decision Editor is used to create decision options, known as candidates, the arguments for and against each option and also to specify other decision attributes.

The Tallis Publisher (based on Java Servlets) forms part of the Tallis software suite. This has also been built to allow guidelines to be published and enacted over the WWW.
Figure 3.10: AREZZO Composer. The Expression Editor is a powerful tool that enables an author to create and amend AREZZO conditions and expressions.
Chapter 4

Conclusion

The various guideline modeling methodologies demonstrate the diversity, the different strengths and weaknesses, and the various intentions of the approaches.

In the early 1990s the Arden syntax became a de facto standard for rule-based medical logic representations. Subsequently developed methodologies show the diversity emanating from this approach. One intention was to be able to model more complex decisions. Therefore, more expressive formalisms were needed, which often were developed in parallel, or often combined already existing attempts to incorporate the best out of these.

Soon, there started efforts for establishing a standardization, which could bring benefits regarding the support of the creation of public repositories of executable guidelines and reusable guideline components. Standards can also facilitate the introduction of an open source content model for publication and sharing of evidence-based practice. It promotes consistency and less incompatibility, as well as compliance with other relevant standards. A standard would also facilitate the integration of decision support with different clinical information systems. Especially, industry is more likely to support standards-based products.

But also diversity can bring advantages, because different representations have different strengths and weaknesses. As requirements and goals change over time a premature standardization should be avoided. Diversity promotes new ideas and directions. Standards are most effective if the field is mature enough, consensus has been achieved, and if users and industry are pushing and are involved.

For formalizing these methodologies also various different approaches exist. All of them need strong user interaction, most demand for high knowledge about the medical domain, the authoring process, and the formal language. These various methods and tools use pure graphical approaches (e.g., AsbruView), markup-based approaches (e.g., DELT/A), or combinations of sophisticated user interfaces and graphical approaches (e.g., Arezzo).

Also in this area it remains to be seen, whether the existing methodologies are sufficient for the industrial deployment, or whether there are completely new (knowledge-based) approaches in demand that facilitate the user in a better way.
Acknowledgements

This project is supported by "Fonds zur Förderung der wissenschaftlichen For- schung FWF" (Austrian Science Fund), grant P15467-INF.
Bibliography


