



UNIVERSITAT ROVIRA I VIRGILI
DEPARTAMENT D'ENGINYERIA INFORMÀTICA I MATEMÀTIQUES
MÀSTER D'ENGINYERIA INFORMÀTICA I DE LA SEGURETAT

MASTER THESIS

ONTOLOGY-BASED EXECUTION OF CLINICAL GUIDELINES

STUDENT:
SILVIA PADRÓN

ADVISORS:
DR. ANTONIO MORENO
DR. DAVID ISERN



ITAKA

Tecnologies Intel·ligents
Associades per la
Gestió del Coneixement

2008/09

Table of contents

1 INTRODUCTION	5
1.1 CLINICAL GUIDELINES	5
1.1.1 BENEFITS AND LIMITATIONS OF CLINICAL GUIDELINES	5
1.1.2 BARRIERS FOR ADOPTING CLINICAL GUIDELINES.....	6
1.2 MANAGEMENT OF CLINICAL GUIDELINES.....	7
1.3 LANGUAGES USED TO REPRESENT CLINICAL GUIDELINES.....	8
1.4 OBJECTIVE OF THIS WORK AND ORGANIZATION OF THE MANUSCRIPT	9
 2 GUIDELINE MODELLING LANGUAGES.....	 11
2.1 PROFORMA	11
2.1.1 THE PROFORMA MODEL	11
2.1.2 CLASS ATTRIBUTES.....	12
2.1.3 EXECUTION OF PROFORMA GUIDELINES.....	14
2.1.4 EXAMPLE.....	15
2.2 SDA *	17
2.2.1 THE SDA* MODEL	17
2.2.2 SDA* TERMS.....	18
2.2.3 TIME MODEL	19
2.2.4 EXECUTION OF SDA* GUIDELINE	19
2.2.5 EXAMPLE.....	20
2.3 ASBRU	23
2.3.1 THE ASBRU MODEL	23
2.3.2 COMPONENTS	24
2.3.3 EXECUTION	25
2.3.4 EXAMPLE.....	25
2.4 GLIF – GUIDELINE INTERCHANGE FORMAT	28
2.4.1 GLIF MODEL	28
2.4.2 EXECUTION	30
2.4.3 EXAMPLE.....	30
2.5 COMPARISON	33
2.5.1 ORGANIZATION OF GUIDELINE	33
2.5.2 MODEL OF GUIDELINE ACTIONS	33
2.5.3 MODEL OF GUIDELINE DECISIONS	33
2.5.4 OTHER BASIC TASKS	34
2.6 CONCLUSIONS.....	36
 3 ONTOLOGY-BASED REPRESENTATION OF CLINICAL GUIDELINES	 37
3.1 ONTOLOGIES.....	37
3.2 DESIGN OF AN ONTOLOGY TO MODEL GUIDELINE REPRESENTATION LANGUAGES	38
3.3 CLASSES.....	40
3.4 DATA TYPE PROPERTIES	42
3.5 OBJECT PROPERTIES.....	43

3.6 INSTANCES.....	43
3.7 CONCLUSIONS.....	52
4 GENERIC GUIDELINE EXECUTOR SIMULATOR	53
4.1 ABSTRACT VIEW OF THE EXECUTOR	53
4.2 ALGORITHM FOR EXECUTING CLINICAL GUIDELINES	54
4.3 ACCESSING THE ONTOLOGY AND THE CLINICAL GUIDELINES	55
4.3.1 INTERFACE TO ACCESS THE ONTOLOGY	55
4.3.2 LANGUAGE PARSERS	56
4.5 EXECUTION	59
4.5.1 SIMULATION OF THE EXECUTION OF A GUIDELINE IN PROFORMA	59
4.5.2 SIMULATION OF THE EXECUTION OF A GUIDELINE IN SDA*	68
4.6 CONCLUSIONS.....	71
5 CONCLUSIONS AND FUTURE WORK	73
APPENDIX A GUIDELINE EXAMPLES	79
A.1 GUIDELINE OF HYPERTENSION TREATMENT WRITTEN IN PROFORMA.....	79
A.2 HOME CARE PATIENT DISCHARGE GUIDELINE WRITTEN IN SDA*	85

1 Introduction

1.1 Clinical guidelines

Clinical Guidelines (in the following, CG) provide a way to support effective clinical practice. CGs are systematically developed statements to assist practitioner decisions about appropriate health care for specific clinical circumstances. Thus, the adoption of CGs is a promising way for standardizing and improving health care practices (Field & Lohr, 1990; Mersmann & Dojat, 2004; Miller & Kearney, 2004). Nowadays, the execution of CGs is one of the most interesting topics of study within *clinical informatics*.

1.1.1 Benefits and limitations of clinical guidelines

For health professionals

The principal benefit of clinical guidelines for health professionals is to improve the quality of care they are able to deliver to patients (Woolf, *et al.*, 1999). CGs facilitate reuse of knowledge keeping abreast of new developments, through the provision of carefully evaluated summaries of available evidence. In addition, a guideline can be adapted, tailored and applied to different clinical situations.

However, poor clinical guidelines can have a negative outcome for health professionals through their recommendation of ineffective, wasteful interventions which undoubtedly compromise the quality of care received by patients (Woolf, *et al.*, 1999). The creation and dissemination of clinical guidelines should be a task of national healthcare organizations, which can promote their cooperative creation, taking into account the local circumstances of their adoption (Barahona, *et al.*, 2001).

While guidelines also have a role in supporting quality assurance, audit and research, including providing a framework against which care can be evaluated (Thomson *et al.*, 1995; Duff *et al.*, 1996), the economic implications of guideline implementation should not be overlooked as, for example, different referral recommendations or treatments could have a significant economic impact on a clinical service. However, CGs promote interventions of proved benefits and discourage those that are ineffective, in any sense, economic or practical.

A number of reviews of the impact of guidelines on practice have been undertaken across a wide range of specialities. However, the potential for a publication bias exists that would result in more publications that demonstrate improvements in practice and outcomes.

In order to enable their potential benefits, protocols must fulfil strong *quality* requirements. This is true not only for the final product, the protocol, but also for the development process. Several attempts to validate clinical guidelines are also being developed (Marcos, *et al.*, 2002; Pech-May, *et al.*, 2009).

For patients

Clinical guidelines permit to inform patients about what their clinicians should be doing and the services that they can expect, and they provide information about the benefits and treatment/care options (Woolf, *et al.*, 1999).

Care should enhance communication between patients and health professionals. Nevertheless, guidelines can empower consumers of health services, enabling them to make informed choices, so enhancing patient participation in decision-making.

However, it is important to carefully evaluate each guideline, as failure to take account of all relevant evidence has the potential to result in sub-optimal or, at worst, harmful practices that can deny patients the best care. Furthermore, guidelines that fail to contain an element of flexibility may prevent clinicians from tailoring treatment or care to the individual needs of specific patients. Guidelines can also benefit patients by influencing public policy, for example, drawing attention to underserved populations or under-recognised health problems (Woolf, *et al.*, 1999). New services may be made available or better services for those in need can be a direct outcome of newly developed guidelines.

1.1.2 Barriers for adopting clinical guidelines

Although several international organizations create and maintain repositories with guidelines in different domains (*e.g.*, oncology, cardiology, pediatrics), they are not being widely used in daily practice.

The factors limiting or restricting complete physicians adherence to clinical guidelines were identified in (Cabana, *et al.*, 1999). Such factors, named *barriers*, were organized into groups based on whether they affected physician knowledge, attitude or behaviour.

- *Knowledge* barriers include a lack of awareness with the guideline's existence, or simply a lack of familiarity with the guideline content.
- *Attitude* barriers refer to lack of agreement with specific guidelines or with guidelines in general, lack of physician self-efficacy, lack of outcome expectancy or a lack of motivation for the inertia of habits or routines.

- *Behaviour (or external)* barriers were divided into guideline-related factors (e.g., difficulty to apply CGs in daily practice, a CG changes established behaviours), patient-related factors (e.g., inability to consider patient preferences with guideline recommendations), and environmental factors (e.g., insufficient staff or resources).

These factors could be tackled (in most cases) with an automation and computerization of the daily management of clinical guidelines and patient data (Blaser, *et al.*, 2007). To deliver patient-specific advice at the time and place of a consultation is an important contribution to improve clinician performance, and the introduction of IT applications is one of the preconditions to be accomplished (Maviglia, *et al.*, 2003; Raghupathi & Tan, 2002; Zielstorff, 1998). Furthermore, IT inclusion into clinical practice is a critical task, as current processes will necessarily have to change and adapt to new circumstances.

1.2 Management of clinical guidelines

Several steps must be considered in the management of clinical guidelines: *representation*, *acquisition*, *verification* and *execution*. The first three tasks concern the authors of the guideline, whereas the later is related to practitioners. Briefly, these steps can be described as follows:

- a) **Choice of a representation language.** A CG contains several elements to be modelled, such as actions, required patient data, decisions to be taken, constraints between tasks, temporal constraints in a global plan, etc. Different researchers have defined formal languages to model computer-interpretable clinical guidelines, such as PROforma, EON, GLIF, GUIDE, SDA* or Asbru (Clercq, *et al.*, 2004; Fox & Das, 2000; Peleg, *et al.*, 2003; Wang & Shortliffe, 2002; Yu, 2007, Riaño, 2007).
- b) **Acquisition of CGs.** Medical guidelines are based on the evidence collected from clinical trials and existing literature (Davis, *et al.*, 2007; Priori, *et al.*, 2003). Some authors are also currently working in the semi-automatic construction of guidelines, by applying Machine Learning techniques from a corpus of clinical data collected in a medical centre (Riaño, 2004) or directly from textual documents (Hrabak, *et al.*, 2007).
- c) **Verification of CGs.** Verification includes two aspects: is a medical guideline well formed?, and, which of these two available medical guidelines is the best? The first question seeks to verify the formal correctness of the guideline (Pech-May, *et al.*, 2009; ten Teije, *et al.*, 2006). The second question is more difficult to answer since it is necessary to quantify how appropriate is a medical guideline. To tackle this problem, some authors proposed an evaluation procedure called AGREE which calculates a set of parameters for a given medical guideline to evaluate its *quality* (AGREE, 2003). In addition, a methodology to facilitate the whole development and evaluation of clinical guidelines can be found in (Ricci, *et al.*, 2006).

- d) **Execution aspects.** As mentioned above, a medical guideline contains a great amount of information to be considered (decisions to be taken, constraints between tasks, temporal restrictions). All these data have to be collected and monitored when enacting the guideline.

Concretely, this work is focused on the construction of a generic knowledge structure, based on some of the most representative guideline representation languages, which allows the automated execution of guidelines written in different formalisms.

1.3 Languages used to represent clinical guidelines

There are several available languages and models to represent guidelines (Peleg, *et al.*, 2003). However, none has achieved widespread use. Indeed, some are not used beyond the context of the research project or institution that gave rise to them. Because they are used for a variety of purposes, such as screening and prevention, diagnosis, judging indications for operative procedures, determining the appropriate use of specific technologies and care of specific conditions, clinical guidelines are complex entities. Beyond such clinical applications, guidelines reflect a variety of activities, such as documentation, recommendation, explanation, presentation, calculation and communication (Miller & Kearney, 2004).

Because any guideline formalism must accommodate these diverse clinical applications and tasks, such formalisms are necessarily complex. As a result, achieving consensus on a standard formalism is a complex undertaking, too. This is an important drawback because it prevents researchers from implementing tools in the same way, reusing components or comparing easily different approaches and there is not any *de facto* standard language.

Although there are different representation languages, it can be seen that, as (Peleg, *et al.*, 2003) and (Clercq, *et al.*, 2004) noticed, they share common functionalities and elements that permit to establish a set of common features. They use similar basic elements (like actions and decisions). They embed declarative knowledge using different primitives like cycling iterations, logical expressions in decisions, control structures for synchronising sequences of actions, and evaluation of pre and post conditions (in different ways) before and after the execution of an action.

Therefore, it is feasible to think that these approaches could converge into a *common language* to cover all the functionalities identified in these representations. Nowadays, the fact that there is no standardisation of representations implies that hospitals implement *ad hoc* solutions in most of the cases. There have been different attempts to create a standard to represent CGs such as GLIF or GELLO, which have had limited success, and none of those proposals was successfully adopted by researchers and widely used in guideline-based execution engines (Boxwala, *et al.*, 2004; Peleg, *et al.*, 2004; Shiffman, *et al.*, 2004).

But, although it is difficult to find a common representation of all languages, it is possible to find a core with the main elements and unify them under a generic structure which allows the guideline execution in different representation languages.

To define this structure it is possible to utilize a knowledge representation tool to model the subset of basic features and elements that CGs should have, in different representations languages. This information will be used to execute the main steps of a guideline, that is, it will allow to identify and process each component, according to its type, and finally show the appropriate information to the user, independently of the language used to write the guide. In this way, the practitioner that will use the clinical guideline for a certain medical treatment will not need to know either the details of its implementation or the language used to write it.

In this manner, this work is focused in the design of an ontology to model a generic structure that allows to guide the user in the workflow of the medical treatment, in a transparent way, and independently of the language in which the guideline is written.

1.4 Objective of this work and organization of the manuscript

Thus, the objectives of this Master Thesis are:

- 1) Study different clinical practice guidelines representation languages. Make a global analysis and a comparison of them, find their common aspects and their most important differences and obtain a core to support a subset of each representation.
- 2) Propose an ontology to model this core to allow the generic execution of CGs, independently of the language in which they are represented.
- 3) Create an application (executor) as generic as possible to simulate the execution of the main steps of CGs written in different representation languages using the information modelled in the ontology.

This thesis is organized as follows:

- **Chapter 2** presents a review of different guideline representation languages. It provides an analysis and a comparison of them in order to know their common features and their main differences.
- **Chapter 3** describes the design of the ontology to support the execution of clinical guidelines in different representation languages. The chapter explains the main classes, relationships and instances represented in this knowledge structure.

- **Chapter 4** presents the prototype built to execute guidelines using the information modelled in the ontology. Firstly it describes the algorithm used to build it and then the main classes and functions that have been created. After that, the explanation is illustrated with the test of the system using two CG examples written in *Proforma* and SDA*.
- **Chapter 5** presents the Conclusions and the Future work. The document also includes the bibliographic references and annex with the guidelines used in chapter 4 to test the application.

2 Guideline Modelling Languages

There are different representation languages to code clinical guidelines (Isern & Moreno, 2008; Peleg, *et al.*, 2003). Even though they are designed to cover different aspects of medical informatics, most of them share a common set of elements and features. Some languages are discussed in this chapter, *PROforma* (Sutton & Fox, 2003), *SDA** (Riaño, 2007), *Asbru* (Miksch, *et al.*, 1997) and *GLIF* (Boxwala, *et al.*, 2004), particularly in the most important subjects for the purpose of this study: the model, basic components and the workflow process. Finally, a global analysis and a comparison are made, with the aim of finding the common aspects and the most important differences to obtain a core with the main elements of each representation language.

2.1 *PROforma*

PROforma is an executable process modelling language that has been successfully used to build and deploy a range of decision support systems, guidelines and other clinical applications (Sutton & Fox, 2003). It was developed at the Advanced Computation Laboratory of Cancer Research, UK.

The heart of the *PROforma* approach is a *knowledge representation* language for representing familiar clinical tasks, concepts and reasoning in a way which seems natural to medical professionals yet is sufficiently well defined to be interpreted unambiguously by a computer. It combines logic programming and object-oriented modelling (Fox & Das, 2000; Fox, *et al.*, 2006).

2.1.1 The *PROforma* model

With the aim to achieve a balance between the clarity and precision of a formal specification language, and an intuitive representation of clinical concepts and procedures, a decision process is defined using the concepts of object-oriented programming. From this point of view, a clinical guideline is modelled as a *plan* made up of one or more *tasks*. There are four basic classes of tasks:

- A **decision** that involves a choice of some kind, such as a choice of investigation or diagnosis therapy. The specification of a decision task defines the decision options, relevant information and rules by which the pros and cons of different options can be derived.
- An **action** is a clinical procedure that is to be enacted outside the computer system, such as the administration of an injection.
- An **enquiry** is an action that yields information. The specification of an enquiry includes a description of the required information (e.g. the type, range and other properties of a clinical parameter) and a method for getting it (e.g. by creating a data entry form, by querying a local patient record or a remote laboratory database).
- A **plan** is a set of tasks, which are to be carried out to achieve a clinical objective, such as a therapeutic objective. Plans are composed of any number of tasks, including sub-plans, and are usually ordered to reflect logical, temporal or other constraints.

Tasks are modelled in an *object-oriented style* and each task is seen as an instance of a more general class. This is shown in Figure 2.1. Each class also has a set of attributes.

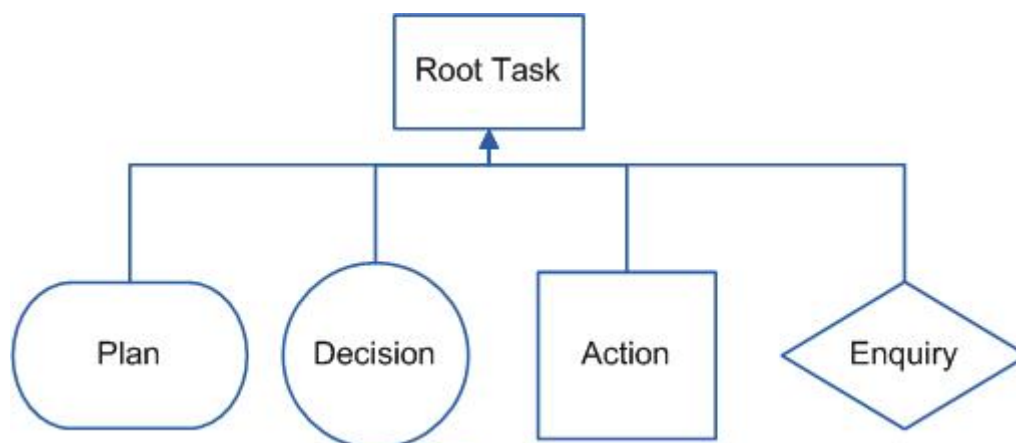


Figure 2.1 PROforma basic tasks.

2.1.2 Class attributes

As mentioned previously, plans can contain any number of tasks and sub-plans. All tasks have a set of common attributes but each subtype has its own distinguishing attributes.

2.1.2.1 General features of tasks

Each task can have a number of properties whose values determine how to interpret it. The value of a property may be a scalar value (e.g., an integer), an expression, or it may be an object, which, itself, has properties.

The main properties shared by all tasks are the following:

- *Caption and description*: anything that can have any properties at all can have a text caption and a description, which are used to provide short or long comments (e.g., explanations of the intended purpose of a task).
- *Pre-condition*: a truth-valued expression that must be true before a task is started.
- *Task scheduling constraints*: logical constraints that prevent one task from starting before another task or set of tasks have been completed.
- *Post-condition*: Situational descriptions which will be assumed to be true at completion of the task.

2.1.2.2 Specific features of tasks

Also, particular subclasses of tasks have distinctive properties:

Plan

Plans define sets of tasks which may be required in a clinical procedure, such as a protocol, and may define logical and temporal constraints on their enactment, together with any circumstances in which the plan should be aborted or terminated.

- *Components*: it is the only attribute of a plan that *must* be specified. Consequently the simplest form of a plan is just a set of tasks which must all be carried out but not in any particular order. A typical plan may be used to capture a clinical flow diagram in which there are constraints on the ordering of tasks together with some conditional branching implemented by means of decisions.

Decisions

Decisions are defined in terms of a set of options and the rules by which arguments for choosing alternative options may be generated.

- *Candidates*: objects representing the options to be considered when a decision has to be taken.

Candidates have their own properties:

-*Arguments*: An argument rule can support (+) or oppose (-) a decision option, establishing a preference order on the options as a function of the overall balance of pros and cons.

-*Recommendation*: rule used to express the conditions under which it would be appropriate to commit to that candidate.

Action

An *action* can be used to represent a wide variety of operations within a plan, from administering medication to printing a leaflet, ordering a test or carrying out a surgical procedure.

- *Procedure*: Operation to be performed as a result of executing an action. The default procedure for an action is to issue a message to the user interface, but an application designer can also include a call to an external procedure where appropriate.

Enquiry

An enquiry is a task that is intended to return information. This may be clinical, administrative or other information and may involve a simple process, such as displaying a data entry form, or a complex one as extracting information from an image or capturing physiological signals from patient monitoring equipment. As with an action, an enquiry method can be implemented with an attached procedure that is encapsulated within the enquiry task.

- *Source*: represents the required information. It is encapsulated by Data Item objects.
- *Data Item*: set of validation (type, range), metadata (units, caption) and interpretation rules (default, abstractions) for obtaining the information.

2.1.3 Execution of PROforma guidelines

The execution guideline starts in a root task, usually a plan, which is used to capture a clinical flow diagram. This plan defines sets of tasks (including another plan, like sub-plans) which may be required in a clinical procedure, such as a protocol, and they are executed in a precise order using logical and temporal constraints.

The action sequencing is governed by the satisfaction of the scheduling constraints and preconditions of individual tasks. The executor maintains the current state of the tasks, which can be executed, in progress, and waiting.

2.1.4 Example

The authors of PROforma designed a graphical tool to create, test and validate clinical guidelines written in this language. This tool is called Tallis and the Figure 2.2 shows part of the hypertension guideline. In addition, Figure 2.3 shows the syntax of this language by text-code.

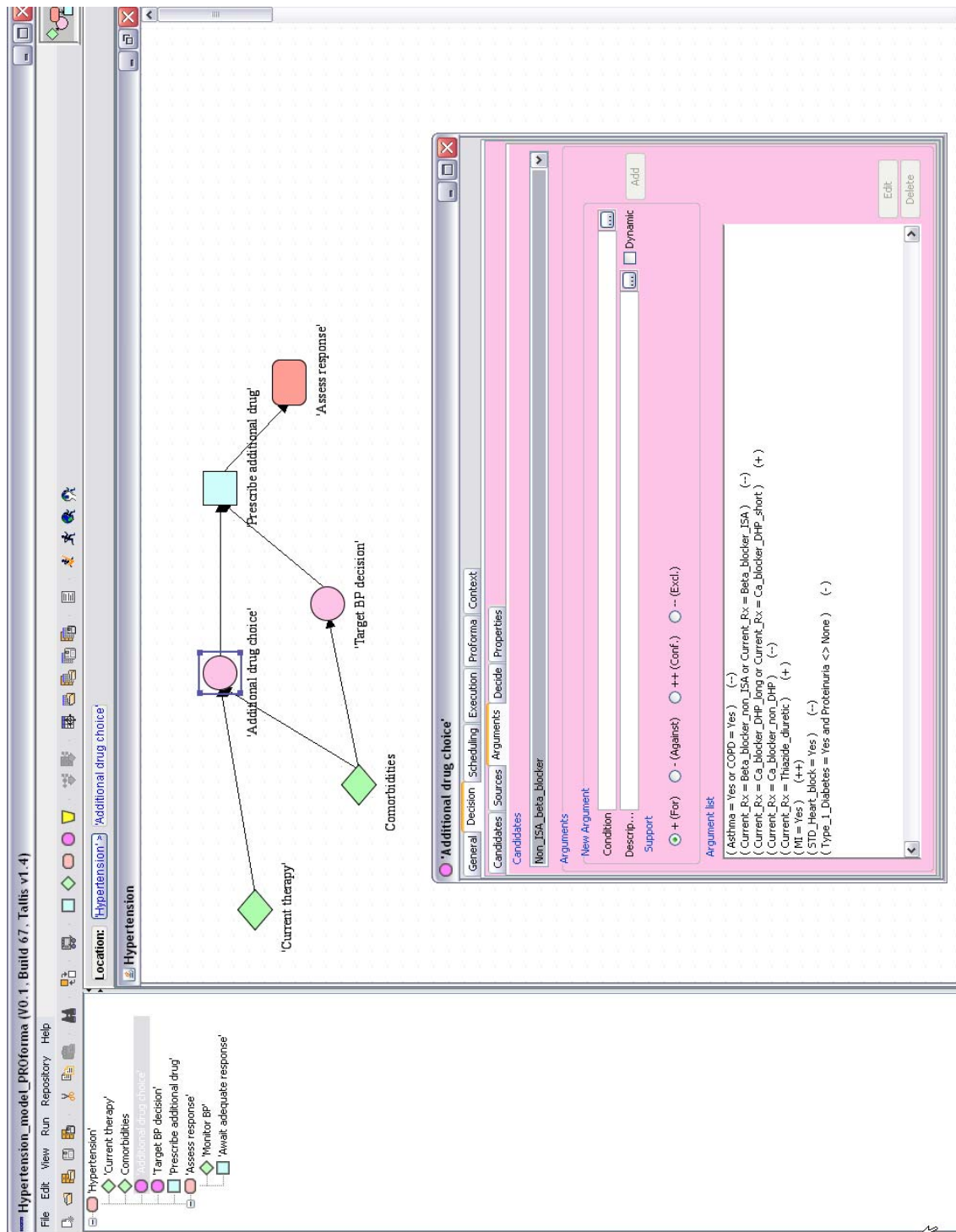


Figure 2.2 Hypertension guideline in PROforma

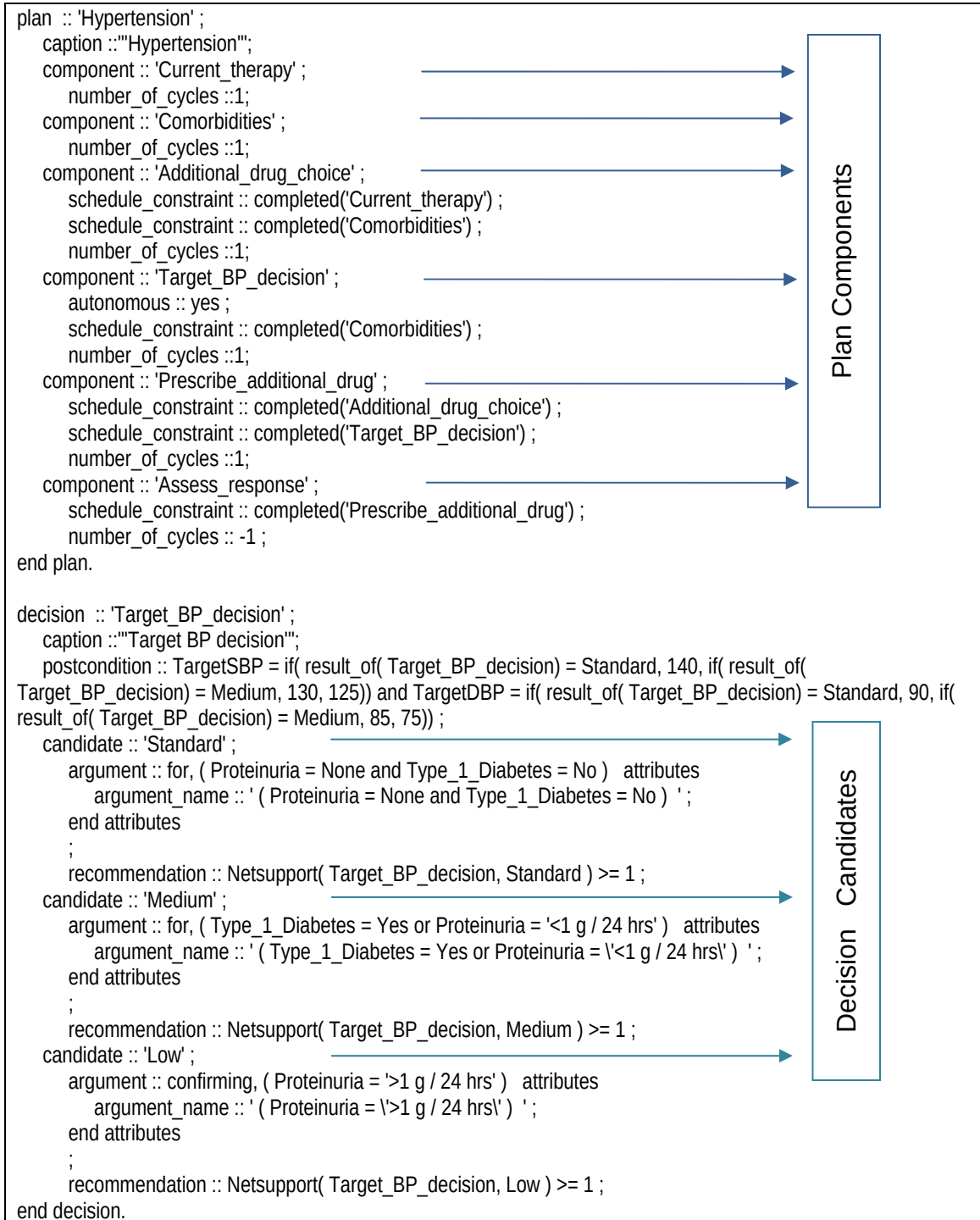


Figure 2.3 Syntax of PROforma

2.2 SDA *

SDA* (State-Decision-Action) was introduced as an alternative language that promotes *representation capability* (i.e., procedural knowledge in Medicine can be represented with this language) and *simplicity* (i.e., the understanding and management of the language does not need hard training) in such a way that not only computers but also health care professionals are able to work with it (Riaño, 2007). SDA* has been tested in the K4Care project (Knowledge-Based Homecare eServices for an Ageing Europe) as an effective framework to formalise clinical procedures (Campana, *et al.*, 2008).

The model is based on the concept of *term* or vocabulary item in the medical domain where procedural knowledge is being generated. Terms are employed to construct the *elements* that once interconnected will describe the medical procedure. The SDA* model uses a flowchart structure and it is extended with several elements to ease health care procedural knowledge representation as, for example, the concept of state as starting point that allows the execution of the chart from different points, and the introduction of time constraints to introduce time restrictions in medical procedures.

2.2.1 The SDA* Model

SDA stands for State-Decision-Action, SDA* represents the repetition of states, decisions and actions in order to describe health care procedural knowledge. Figure 2.4 shows these three basic components.

- **States:** Represent patient conditions, situations, or statuses that deserve a particular course of action which is totally or partially different from the actions followed when the patient is in other state.
- **Decisions:** Capture the need of procedural knowledge to represent alternative options whose selection depends on the available information about the patient. In this sense, decisions unify in a single point of the procedural knowledge alternative courses of actions that have to be applied to patients that meet different conditions. They allow the integration of the variability that a treatment has depending on the available information about the patient.
- **Actions:** Constitute the proper treatment steps in the SDA* procedural knowledge that are performed according to the preceding decisions.



Figure 2.4 Basic elements of the SDA* Model.

The correct combination of states, decisions, and actions allows the construction of explicit health care procedural knowledge within the SDA* model. This combination of elements is made by means of connectors. In the SDA* model, a connector is defined as an arrow that goes from one element in the input of the connector (or in-element) to another element in the output of the connector (or out-element).

From the point of view of the SDA* elements, any state is an in-element of one connector, but it may be an out-element of any number of connectors. Decisions are in-elements of as many connectors as the number of branches on the decision, and out-elements of one or several connectors in the SDA* procedure. Finally, actions are in-elements of one only connector.

2.2.2 SDA* Terms

Each of the classes that make up the SDA* is defined by a set of terms which represent and describe the procedural health care knowledge.

State

From the semantic point of view, a *state* (or *SDA* entry point*) describes an abstract patient condition in which all the terms in the state hold. From a logical point of view, a state is a conjunction of state terms. From a functional point of view, the states of a SDA* procedure are the entry points to that procedure or, in other words, the points where the treatment can start.

A patient state contains a set of logical conditions that should be evaluated before proceeding. Some of the values required in these conditions can be retrieved from the patient's record, but there could be findings to be entered by the practitioner during the medical visit.

Decision

A *decision* (or *SDA* branching point*) describes a point of the SDA* where the treatment can follow alternative courses of action depending on which are the decision terms the treated patient meets.

From a logical point of view, a decision represents a disjunction of conjunctions on a set of decision terms and, from a functional point of view, decisions allow the represented SDA* to be general and flexible enough as to combine several variations on the treatment of a disease, and to make the application of these variations depend on the particularities of the patient.

A decision term is defined as:

- *Branch List*: Represents the decision alternatives.
- *Otherwise Branch*: Selected alternative when none of the other alternatives is appropriate.

Action

An *action* (or *SDA* action block*) describes a group of actions in the SDA* procedure. These elements represent action proposals whose application must be made out of the SDA* model.

- *Action blocks*: In SDA* the actions are contained in action blocks. They are independent of the patient condition; therefore they are usually preceded either by a state that describes what the state of a patient should be in order to deserve that action, or by a decision that determines whether the patient meets the properties required for the action to be applied.

Each action term in an action element has two constraints:

- *Petitioner*: Actors that are allowed to request the action.
- *Performer*: Actors that are allowed to perform the action in the real world.

2.2.3 Time model

The time model establishes two sorts of temporal constraints: those which are related to the terms in a SDA* element and those related to the connectors. Each term and connector may optionally have one constraint.

The time constraints of the terms are of the sort *[start, end, frequency]* and they mean that the term is observed from the start time to the end time with the indicated frequency. The second sort of time constraints is related to the SDA* connectors and it has the form *[min, max]*. They are optional and represent delays (or durations). A connector with such a time constraint indicates that the evolution from the in-element to the out-element of the connector takes between *min* and *max* times.

2.2.4 Execution of SDA* guideline

The execution of a SDA* guideline, given a patient condition, starts at any of the states of the SDA* that are feasible entry points. Each feasible entry point in the selected set starts an alternative feasible treatment of the patient.

A treatment consists of all the action (temporal) terms found in a SDA* path that starts in a feasible entry point and finishes either in a state that is not a feasible entry point or in a connector with a temporal range [min, max] with min>0. This path is a sequence of SDA* elements, each one being an out-element of a connector with in-element the previous element in the sequence. If the in-element is a decision, all the branches that the patient condition meets (or the branch otherwise if none meets), i.e. the feasible branch of the decision, can be followed.

The basic structure of the SDA* model is the SDA sequence that connects one state with a decision and each branch of that decision with an action.

2.2.5 Example

Below is shown the graphical representation of an example of a clinical guideline for blood pressure coded in SDA*. Also, the SDA* model is stored in a XML-based syntax shown in Figure 2.5 in an example of anemia treatment.

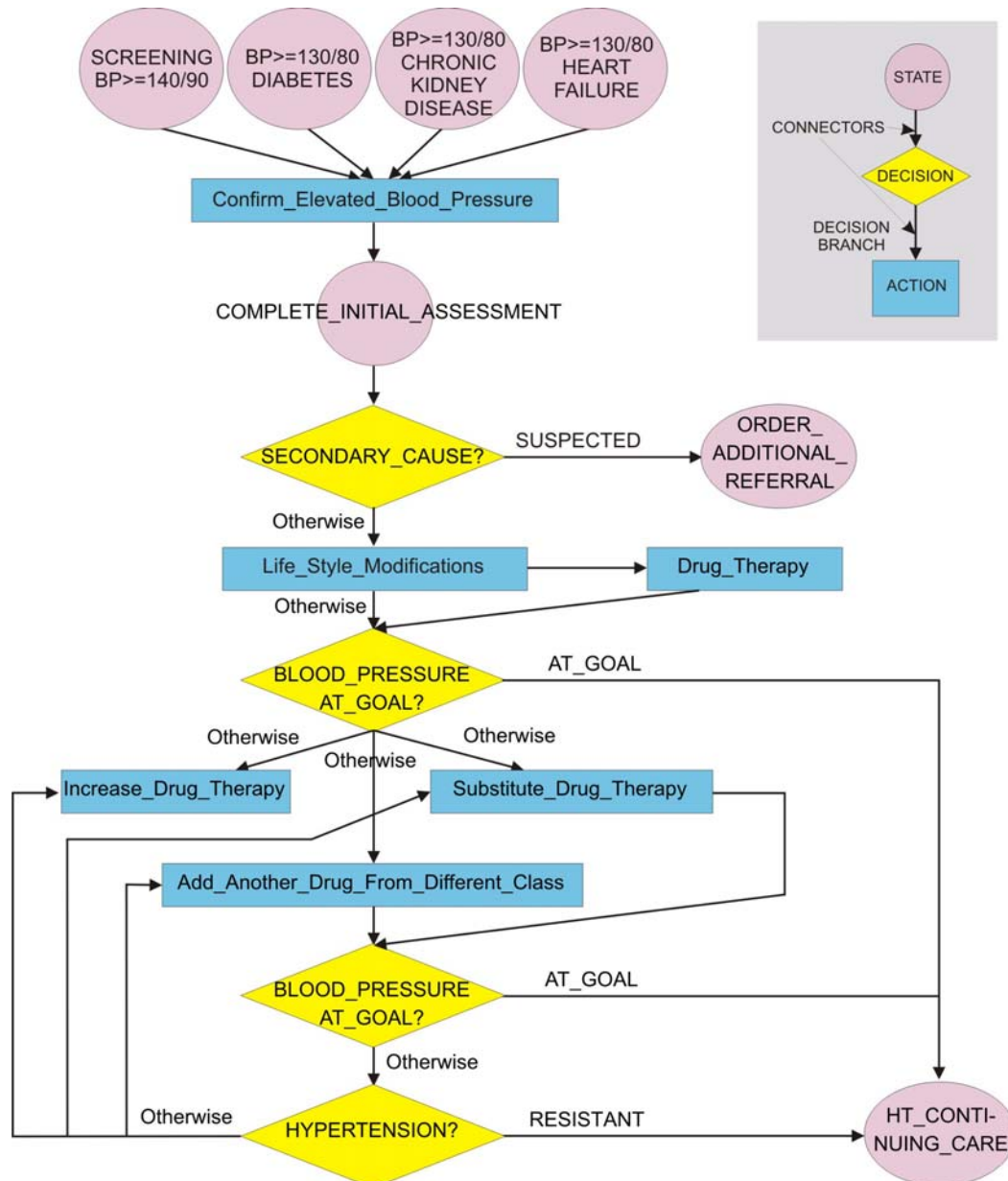


Figure 2.5 Graphical representation of a clinical guideline in SDA*

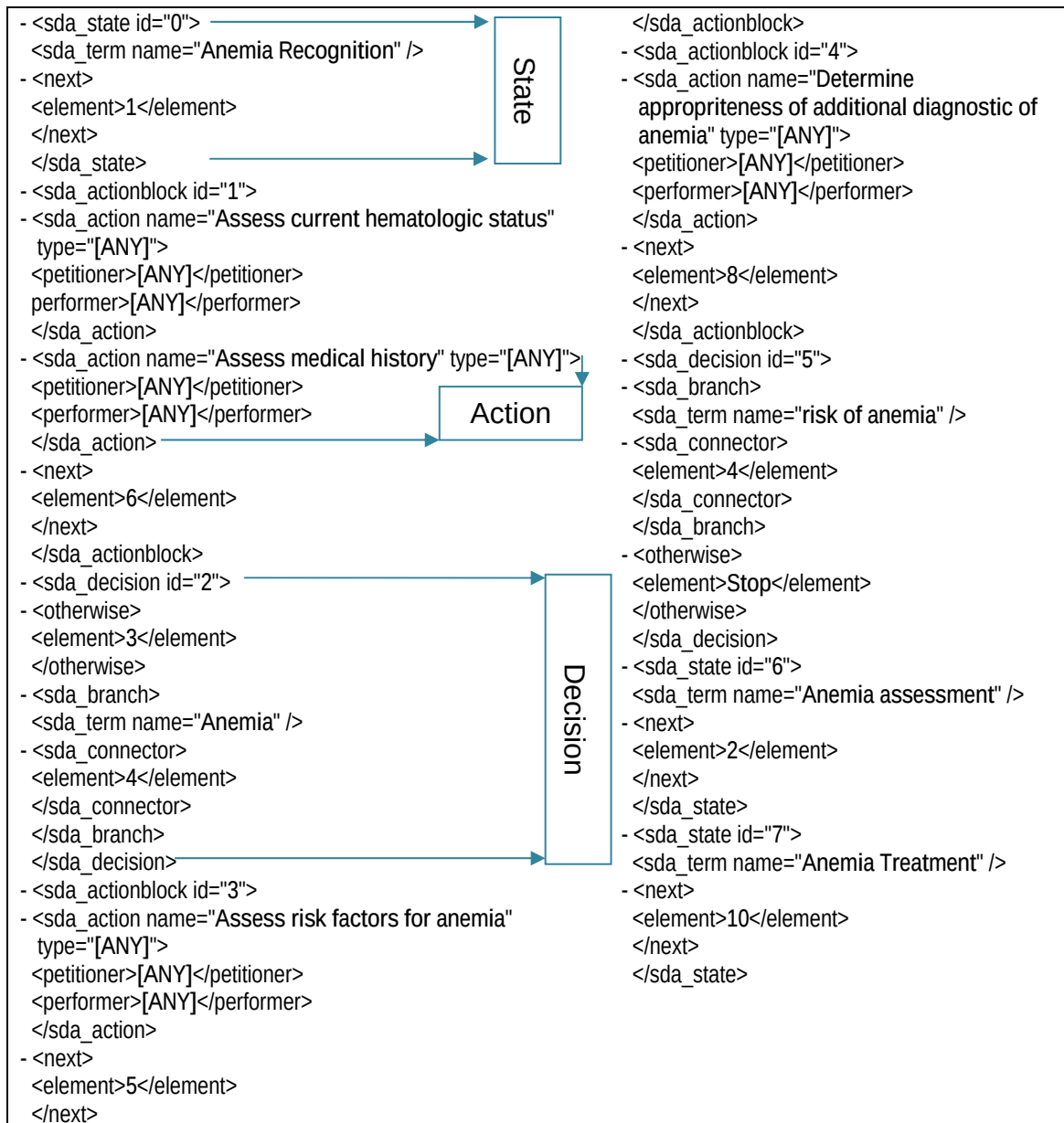


Figure 2.6 SDA* Syntax

2.3 Asbru

Asbru is the time-oriented, intention-based, skeletal plan-specification representation language defined in the Asgaard Project to represent clinical guidelines and protocols in XML (Miksch, *et al.*, 1997; Shahar, *et al.*, 1998). Asbru can be used to express clinical protocols as skeletal plans that can be instantiated for every patient. It was specifically designed for plan-management tasks. Asbru enables the designer to represent both the prescribed actions of a skeletal plan and the knowledge roles required by the various problem-solving methods for performing the intertwined supporting subtasks.

2.3.1 The Asbru Model

In Asbru, protocols are expressed as plan schemata defined at various levels of detail, precise enough to capture the essence of clinical procedures but leaving space for flexibility during their application.

An Asbru specification includes *conditions*, such as eligibility criteria; control structures for the guideline's *body* (e.g., sequential, concurrent, and periodic combinations of actions or sub-guidelines), *preferences* (utility functions), expected *effects*, and process and outcome *intentions*. A feature unique to Asbru is the use of explicit intentions, represented as temporal patterns, which support intelligent quality assessment by representing the designer's *intentions* regarding care-provider actions and patient outcomes.

The major features of Asbru are:

- Explicit intentions and preferences can be stated for plans.
- Intentions, conditions, and states are temporal patterns.
- Uncertainty in temporal patterns and parameters can be flexibly expressed.
- Plans can be executed in different compositions, e.g. in parallel (TOGETHER), in sequence (SEQUENCE), in any order (ANYORDER), or every certain time (CYCLICAL); it can also be defined whether all the steps should be executed (ALL) or not (SOME).
- Conditions can be defined to control plan execution, e.g. to set applicability conditions (FILTER) or to determine when execution should be interrupted (ABORT).

Intentions are high-level goals that support special tasks such as critiquing and modification. They are patterns of states or actions to be achieved, maintained, or avoided (ACHIEVE, MAINTAIN or AVOID), during or after the execution of the plan (INTERMEDIATE or OVERALL).

Temporal patterns are crucial in Asbru. The time annotations used in Asbru allow the representation of uncertainty in starting time, ending time, and duration with intervals, as well as the use of multiple reference points. A time

annotation is written in the form ([EarliestStarting,LatestStarting] [EarliestFinishing,LatestFinishing] [Min-Duration,MaxDuration] REFERENCE). Thus, the temporal pattern (TSB-decrease=yes any [4 hours,] [,6 hours] [,] *self*) is a proposition becoming true if there is a decrease of TSB (total serum bilirubin), in any context, between 4 and 6 hours after the activation of the current plan, *self*.

2.3.2 Components

A plan consists of a name, a set of arguments, including a time annotation (representing the temporal scope of the plan), and five components: **preferences**, **intentions**, **conditions**, **effects**, and a **plan body** which describes the actions to be executed (Seyfang, *et al.*, 2005). The general arguments, the time annotation, and all components are optional. A subplan has the same structure.

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

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

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

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

Plan Body (Actions). The plan body contains plans or actions that are to be performed if the preconditions hold. A plan is decomposed into sub-plans until a nondecomposable plan (called an action or a user-performed plan) is found. All the sub-plans consist of the same components as the plan.

Time Annotations. An important part in specifying the complex temporal aspects of plans are *Time Annotations*. They are used to allow a representation of uncertainty in starting time, ending time, and duration.

2.3.3 Execution

During the execution phase, an applicable plan is instantiated. A set of mutually exclusive *plan states* describes the actual status of the plan during execution.

Particular *state-transition criteria* which are defined using *conditions* specify transitions between neighboring plan states. There are different plan states (*start*, *possible*, *rejected*, etcetera) and their corresponding transition criteria (*filter-precondition*, *wrong filter-precondition*, *setup-precondition*). The states are *considered*, *possible*, *rejected*, and *ready*, and they determine if a plan is applicable and whether a plan instance can be created. The user-performed plans are displayed to the user, which can react or agree when the action is finished and if it was successful.

2.3.4 Example

In the following, a complex example of Jaundice written in Asbru is shown (Marcos, *et al.*, 2001). Figure 2.7 shows a screenshot of the Asbru viewer, called AsbruView. Figure 2.8 shows the syntax of this language by text-code.

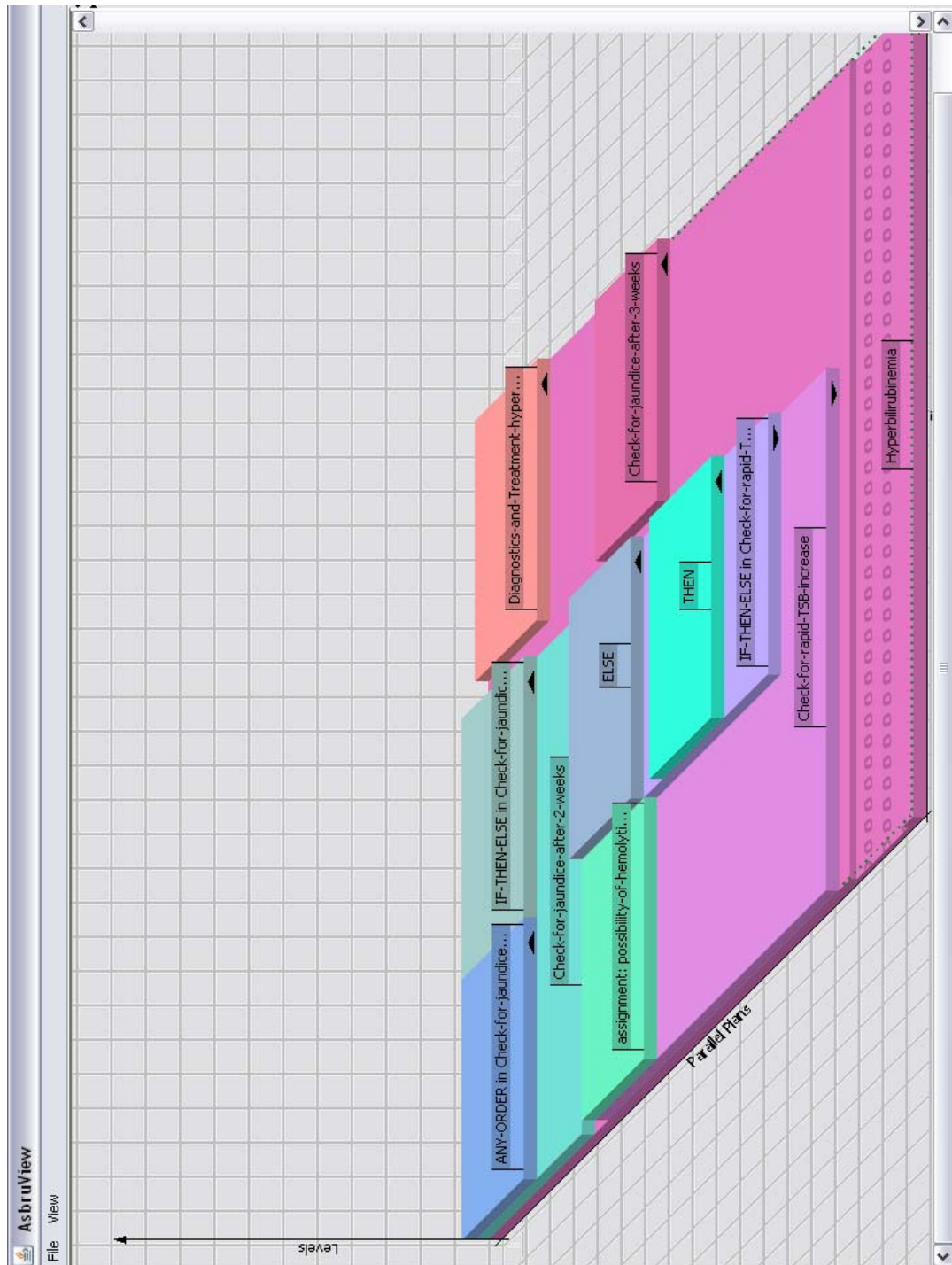


Figure 2.7 Jaundice example written in Asbru

```

<parameter-def name="bilirubin" type="bilirubin-scale">
  <qualitative-parameter-def>
    <limits unit="mgdl">
      <context>
        <one-of name="age">
          <value-ref name="day2"/>
        </one-of>
      </context>
      <limit-entry value="0"/>
      <limit-entry value="12"/>
      <limit-entry value="15"/>
      <limit-entry value="20"/>
      <limit-entry value="25"/>
      <positive-infinite/>
    </limits>
    <limits unit="mgdl">
      <context>
        <one-of name="age">
          <value-ref name="day3"/>
        </one-of>
      </context>
      <limit-entry value="0"/>
      <limit-entry value="15"/>
      <limit-entry value="18"/>
      <limit-entry value="25"/>
      <limit-entry value="30"/>
      <positive-infinite/>
    </limits>
    <limits unit="mgdl">
      <context>
        <one-of name="age">
          <value-ref name="later"/>
        </one-of>
      </context>
      <limit-entry value="0"/>
      <limit-entry value="17"/>
      <limit-entry value="20"/>
      <limit-entry value="25"/>
      <limit-entry value="30"/>
      <positive-infinite/>
    </limits>
  </qualitative-parameter-def>
</parameter-def>
<parameter-def name="TSB-change"
type="amount">
  <change-def>
    <interval>
      <numerical-constant scale="h"
value="6"/>
    </interval>
    <parameter-ref name="TSB-value"/>
  </change-def>
</parameter-def>
<parameter-def name="TSB-decrease"
type="boolean">
  <qualitative-parameter-def>
    <limits>
      <context>
        <any/>
      </context>
      <negative-infinite/>
      <limit-entry include-limit-value="yes"
value="0"/>
      <positive-infinite/>
    </limits>
    <parameter-ref name="TSB-
change"/>
  </qualitative-parameter-def>
</parameter-def>

```

Figure 2.8 Portion of the Jaundice Asbru guideline written in XML

2.4 GLIF – Guideline Interchange Format

The Guideline Interchange Format version 3 (GLIF) has been collaboratively developed by groups at Columbia, Stanford, and Harvard universities (working together as the InterMed Collaboratory) (Boxwala, *et al.*, 2004).

GLIF stresses the importance of sharing guidelines among different institutions and software systems. It tries to build on the most useful features of other guideline models, and to incorporate standards that are used in health care (Choi, *et al.*, 2007; Peleg, *et al.*, 2001).

The authoring process. Supports representing guidelines in three levels of abstraction, marked by letters A to C. First, the guideline is authored at a conceptual level (A). This level enables the guideline author to concentrate on seeing the guideline as a flowchart without full details of decisions. The latter, together with patient data and iteration information are added to reach level B. Mapping actions to institutional procedures at a certain site and patient data references to a certain electronic patient data record leads to level C.

Representation of overall guideline features. *Author information* comprises authors, authoring date, guideline version, guideline status (published or not, obsolete), and developing institution of the free text guideline, plus a similar set of encoder, encoding last modification data, encoded guideline version, GLIF version used, representation status (of the syntax used), and adapting institution for the encoding of the free text version in GLIF.

Logic-oriented aspects describe intention, eligibility criteria, exceptions from the guideline, data items used by the guideline, and parameters passed to the guideline. Guidelines can be grouped into guideline collections within which one guideline can call another and exchange data with it. The data items are specified through ontologies which can be mapped to standard controlled vocabularies. Also, the activities described in a GLIF guideline can be mapped to a Reference Information Model such as that of HL7, which is also called Unified Service Action Model (USAM) (Schadow, *et al.*, 2001).

In addition, a list of supplemental didactic material can be given for each guideline.

2.4.1 GLIF Model

The representation of guideline content is shown in Figure 2.9. The guideline's algorithm is composed of guideline steps. These can be *action*, *decision*, *branch*, *synchronization*, and *patient state*.

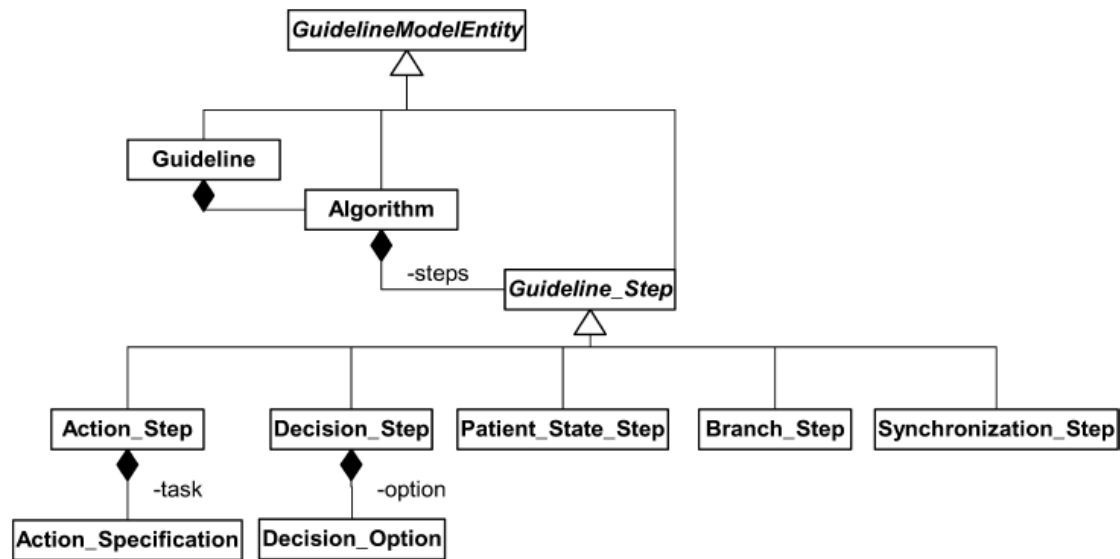


Figure 2.9 A high-level view of the major classes in GLIF (Boxwala, *et al.*, 2004). The lines between classes denote relationships: a diamond-shape arrowhead indicates an aggregation or containment relationship, and a triangle shape indicates a generalization relationship

Action An action step specifies a set of tasks to be performed. Attributes are iteration information, duration, triggering events, and associated exceptions. Actions can be nested. There are two types of actions: medically-oriented and programming-oriented. *Medically-oriented actions* specify a medical task as defined in the Reference Information Model (RIM) layer of GLIF's data model. Examples include diagnostic test order, drug prescription, referral, and scheduling. *Programming-oriented actions* include sending messages, generating events, accessing patient data, assigning values to variables, and calling a subguideline.

Decision Decision steps direct the control flow between alternative steps. There are two varieties: case steps model deterministic decisions while choice steps model non-deterministic decisions. In the first case, the condition is (automatically) evaluated and the appropriate option is taken. If none of the options fits the data or if data are missing, a default action is chosen for continuation. In the case of choice steps, more than one option can be chosen. In this case, the user has to choose the option which is actually performed. The degree of preference can be modeled differently for different types of choices:

- A rule-in choice defines two criteria: a rule-in and a strict rule-in. The first is a condition which may hold for the choice to be applicable. The second is a condition which must hold.
- An array of criteria where each criterion is associated with a weight. The sum of all fulfilled criteria of a choice determines how it is ranked on the list of choices.
- Utility choices use either a decision analysis tree or an influence diagram to compute the ranking of each option.

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

Synchronization Synchronization 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 A patient state 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.

2.4.2 Execution

The guideline execution is encoded in the form of a flowchart. The flowchart is an instance of the *Algorithm* class shown in Figure 2.9. It contains a collection of steps that are instances of the *Guideline_Step* class. The flow of the steps in the algorithm is specified by linking explicitly a step to its subsequent step(s) using appropriate attributes in each subclass of *Guideline_Step*, e.g., the *next_step* attribute of an *Action_Step*. The *first_step* attribute of the *Algorithm* class indicates the starting point of the algorithm. However, alternative starting points in the algorithm can be specified using patient state steps.

2.4.3 Example

Figure 2.9 shows the graphical representation of a clinical guideline example for Hypertension treatment in GLIF. In addition, Figure 2.10 shows the guideline code in the XML-based language called RDF.

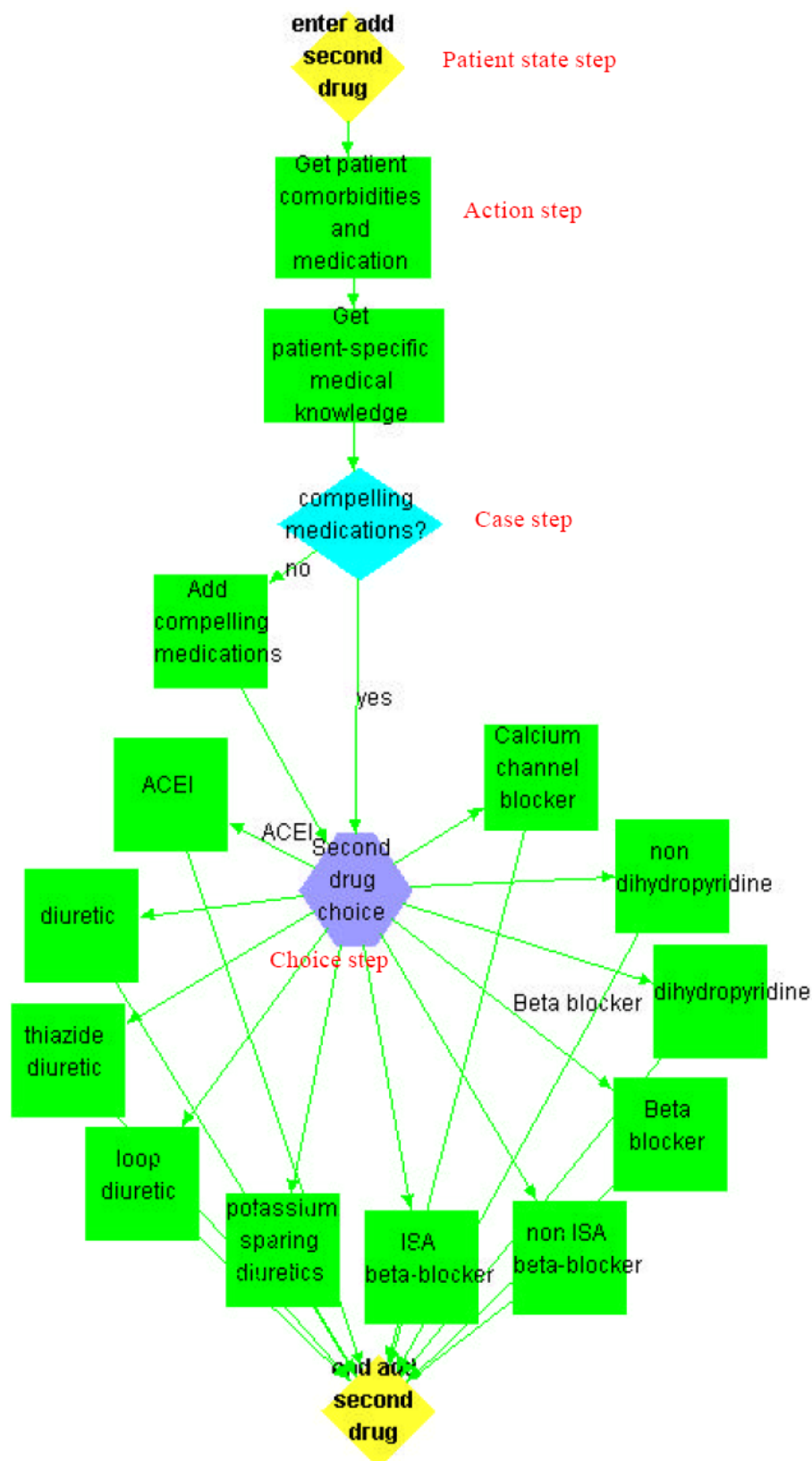


Figure 2.9 Graphical representation of a clinical guideline in GLIF

```

<!DOCTYPE rdf:RDF [
  <!ENTITY a 'http://www.rdfschema.org/mynamespace.rdf#'>
  <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
]>
<rdf:RDF xmlns:a="&a;"
  xmlns:rdf="&rdf;">
  <a:Guideline rdf:about="&a;hypertension_00001">
    <a:name>Add a second drug</a:name>
    <a:intention xml:space='preserve'><![CDATA[Sustain goal BP of < 140/90 mm Hg.
      In patients with Diabetes Mellitus, the BP should be controlled to < 130/85 mm Hg.
      In patients with proteinuria in excess of 1 gram per 24 hours the BP should be < 125/75 mm Hg.
      In patients with proteinuria with at most 1 gram per 24 hours the BP should be < 130/85]]>
    </a:intention>
    <a:eligibility_criteria rdf:resource="&a;hypertension_00002"/>
    <a:maintenance_info rdf:resource="&a;hypertension_00003"/>
    <a:didactics rdf:resource="&a;hypertension_00004"/>
    <a:algorithm rdf:resource="&a;hypertension_00006"/>
    <a:data_items rdf:resource="&a;hypertension_00041"/>
    <a:parameters_passed rdf:resource="&a;hypertension_00547"/>
    <a:parameters_passed rdf:resource="&a;hypertension_00548"/>
  </a:Guideline>
  <a:Three_Valued_Criterion rdf:about="&a;hypertension_00002">
    <a:name>the patient is not at the goal blood pressure, the response to the initial drug choice for
      hypertension is inadequate after reaching the full dose, and the patient is tolerating the first choice
      well
    </a:name>
  </a:Three_Valued_Criterion>
  <a:Maintenance_Info rdf:about="&a;hypertension_00003">
    <a:GLIF_version>GLIF3</a:GLIF_version>
    <a:authoring_date>November 1997</a:authoring_date>
    <a:developing_institution>NIH</a:developing_institution>
    <a:developing_institution>National Heart, Lung and Blood Institute</a:developing_institution>
    <a:encoded_guideline_version>1.0</a:encoded_guideline_version>
    <a:encoded_last_modification_date>Decmeber 7, 2000</a:encoded_last_modification_date>
    <a:encoder>Mor Peleg</a:encoder>
    <a:guideline_status>published</a:guideline_status>
    <a:guideline_version>No. 98-4080</a:guideline_version>
    <a:representation_status>production</a:representation_status>
    <a:title>hypertension guideline</a:title>
  </a:Maintenance_Info>

```

Figure 2.10 Portion of a GLIF guideline for Hypertension treatment coded in RDF

2.5 Comparison

In previous sections different examples of guideline representation languages were analyzed: *PROforma*, *SDA**, *Asbru* and *GLIF*. In the following, a comparison of all these languages is made in order to find a core with the main elements of each representation language. This study, analysis and comparison tasks are timely because there is understandable interest in developing a standardised consensus guideline representation language (Peleg, *et al.*, 2003).

2.5.1 Organization of guideline

A unifying feature of languages is the decomposition of guidelines into basic elements, like actions and decisions, and interrelationships between them to form a workflow. To describe this workflow *PROforma* and *Asbru* use plans. In *SDA** there are not explicit plans to control guideline flow, which is generated from the same tasks (actions, decisions or states), through a connector element, which points to the next task to execute. Plans are implicitly coded in *GLIF* through *Guideline_steps*, in a way similar to *SDA** connectors.

2.5.2 Model of guideline actions

All the guidelines model actions, used to represent the actual tasks described by a clinical guideline.

In *PROforma*, post-conditions and pre-conditions can be used to affect control flow, although task selection on the basis of such post-conditions has not yet been implemented.

2.5.3 Model of guideline decisions

Decision-making is central to most clinical guidelines. Although a variety of decision models are used, one can find a common element: a decision has a set of alternatives.

In *PROforma* a decision can have pre-conditions and post-conditions. Moreover, each alternative has arguments and recommendation rules. In *SDA** and *GLIF* the decision model is based in a set of branches which describe the mutually exclusive options of guideline control flow. Each branch represents an alternative and has a connector to the next task, which will continue the workflow when the alternative expression matches. In *SDA** a decision has also an otherwise branch that is used when none of the other alternatives is feasible.

It is important to note the different relationships between decision-making and control flow. In all the languages, except in *Proforma* decision-making is explicitly coupled to commitment to a decision alternative and the alternative selected in a decision affects flow. In *PROforma* sequencing is governed by the satisfaction of the scheduling constraints and preconditions of individual tasks, rather than through the use of explicit *go to* relationships between tasks.

2.5.4 Other basic tasks

All the languages contain actions and decisions, although in each case they have specific characteristics. In addition, each of them also models other tasks: the states in SDA* and GLIF, the action blocks in SDA*, and enquiries in *PROforma*, with other attributes: pre-conditions and post-conditions, procedure or performer and petitioner.

Although these elements appear particular and specific to each language, in many cases similarities between them can be found and, in other cases, they are defined implicitly in other languages.

Action and Block-Plan

Both are containers of other tasks and coordinate their execution: in the case of the plans, of several types, in the case of the action block, only actions. Also, in both cases the temporal restrictions are considered in the task relationship. Therefore, it can be said that an action block simulates a plan that contains only actions.

Procedure

In *PROforma* the operation to be performed as a result of executing an action is named procedure and it is a static definition. It means that the allowed values for this attribute are texts. Therefore it is similar to the definition of an action of other languages, in which a procedure is not defined explicitly, but in such a way that the name of the action denotes the operation to execute.

Pre-condition and State

In *PROforma*, all tasks are associated with a pre-condition, i.e. a condition that must be met for executing the task.

In other languages, although the pre-conditions are not explicitly defined, they are implicit. In SDA* and GLIF, each task defines the next task to execute, and it depends on a set of conditions to be met at execution time.

Pre-conditions can also be represented using other elements. For example, in SDA*, it is possible to simulate the pre-conditions creating a state with the conditions that must be satisfied before the execution of the task.

The most important common and particular elements of all languages are compared in Table 2.1 and everything that has been explained previously in the case of the particular elements is also summarised.

	Proforma	SDA*	Asbru	GLIF
Action	X	X	X	X
o Procedure	(explicit)	(implicit)	(implicit)	(implicit)
o Performer		X		
o Petitioner		X		
o Pre-condition	X	(implicit)		
o Post-condition	X			
Action Block	(implicit)	X	(implicit)	(implicit)
Decision	X	X	(implicit)	X
o Candidate	X	X		X
o Argument	X			
o Recommendation	X			(implicit)
o Otherwise		X		
o Pre-condition		X		
o Post-condition	X			
State	(implicit)	X	(implicit)	X
Enquiry	X	(implicit)	(implicit)	
Plan	X	(implicit)	X	(implicit)

Table 2.1 Comparison of main elements in SDA* , PROforma, Asbru, and GLIF

2.6 Conclusions

There are different languages to represent clinical guidelines. This chapter analyzed and compared different important languages: SDA*, PROforma, Asbru and GLIF.

- These languages have basic tasks or elements, which can be treated as classes with their attributes. There are common elements such as the decisions and actions and different ones like enquiries and states.
- Classes relate to one another and define the workflow process guideline, but this relationship is defined differently in each language.
- Although some elements are particular of a given language, in more cases they can be modelled or simulated with elements of other languages. For example, a *Proforma* plan can be simulated with a SDA* Action Block.

Once the common aspects and the most important differences among the languages are known, that information can be used to define a structure to allow the guideline execution in different representation languages. In the next chapter we attempt to define this structure.

3 Ontology-based representation of clinical guidelines

As seen in previous chapters there exist different languages to represent CGs, and none has achieved widespread use. This fact that there is not a standard language implies the implementation *ad hoc* of guideline execution environments. There have been different attempts to create a standard to represent CGs but they have not been successful.

Based on the principle that it is hard to find a common representation, but it is possible to obtain a generic structure to support a subset of each representation, in the previous chapter different languages for guideline representation were discussed.

Considering this information, this chapter describes how an ontology has been designed to support the execution of clinical guidelines in different representation languages.

3.1 Ontologies

An *ontology* can be defined as a formal, explicit specification of a shared conceptualization (Studer, *et al.*, 1998). *Conceptualization* refers to an abstract model of some phenomenon in the world by having identified the relevant concepts of that phenomenon. *Explicit* means that the type of concepts used, and the constraints of their use, are explicitly defined. *Formal* refers to the fact that the ontology should be machine-readable. *Shared* reflects the notion that an ontology captures consensual knowledge, that is, it is not private of some individual, but accepted by a group.

In (Neches, *et al.*, 1991), a definition focused on the form of an ontology is given. An ontology defines terms and relations comprising the vocabulary of a topic area as well as the rules for combining terms and relations to define extensions to the vocabulary.

Different knowledge representation formalisms exist for the definition of ontologies. They share the following minimal set of components:

- a) *Classes*: represent concepts. Classes in an ontology are usually organized in taxonomies through which inheritance mechanisms can be applied.
- b) *Individuals*: are used to represent real world entities.
- c) *Properties*: they represent binary associations between ontological entities. On the one hand, *object properties* establish relationships between pairs of individuals. On the other hand, *data type properties* relate an individual to a data value (integer, string, float, etc.); they can be considered attributes (Isern, *et al.*, 2007).

3.2 Design of an ontology to model guideline representation languages

The main elements of each language determined in the previous chapter were taken into consideration to define the ontology (see table 2.1) and the following aspects were considered:

- The ontology is divided in two fundamental branches. The semantic aspects of the representation languages are modeled in the first one. The second is used to describe the aspects related with the execution of the guideline.
- The classes were defined in the most generic possible way, gathering all the similar elements in a concept but, at the same time, considering the most important differences. The ontology was defined to support a subset of the studied representation languages, with the elements allowing the modeling of the most important characteristics of each one. It can happen that some languages do not contain some component or some attribute modeled in the ontology; in this case, this concept will not be defined or the attribute would be null for this language respectively.
- The instances of a class represent each guideline modeling language, *i.e.*, each instance describes how this class is represented in a given language. The idea is that, when a new language has to be modeled, a new instance for each class will be added whenever necessary. This representation is thought for the third objective of this work: the execution of a guideline using the concepts modeled in the ontology, topic that will be described in the next chapter.
- As examples to design the ontology two representation languages were selected: *PROforma* and *SDA**.
- Finally, to build the ontology, the tool Protégé (Gennari, *et al.*, 2003) was used as well as the Web Ontology Language (OWL), one of the most used languages for representing ontologies (McGuinness & van Harmelen, 2004).

The defined ontology will be presented in the next sections. Next, a graphical representation (Figure 3.1) is shown with the objective to illustrate the ontology as a whole to provide a better understanding of the built structure to the readers. Classes are represented by ovals and the instances by rectangles. Classes also list the defined properties (*object* and *data type* properties). The relationships among the classes and the subclasses, as well as among the classes and the instances are shown with an arrow with solid lines, while the relationships defined by the *object properties* are represented with dashed lines.

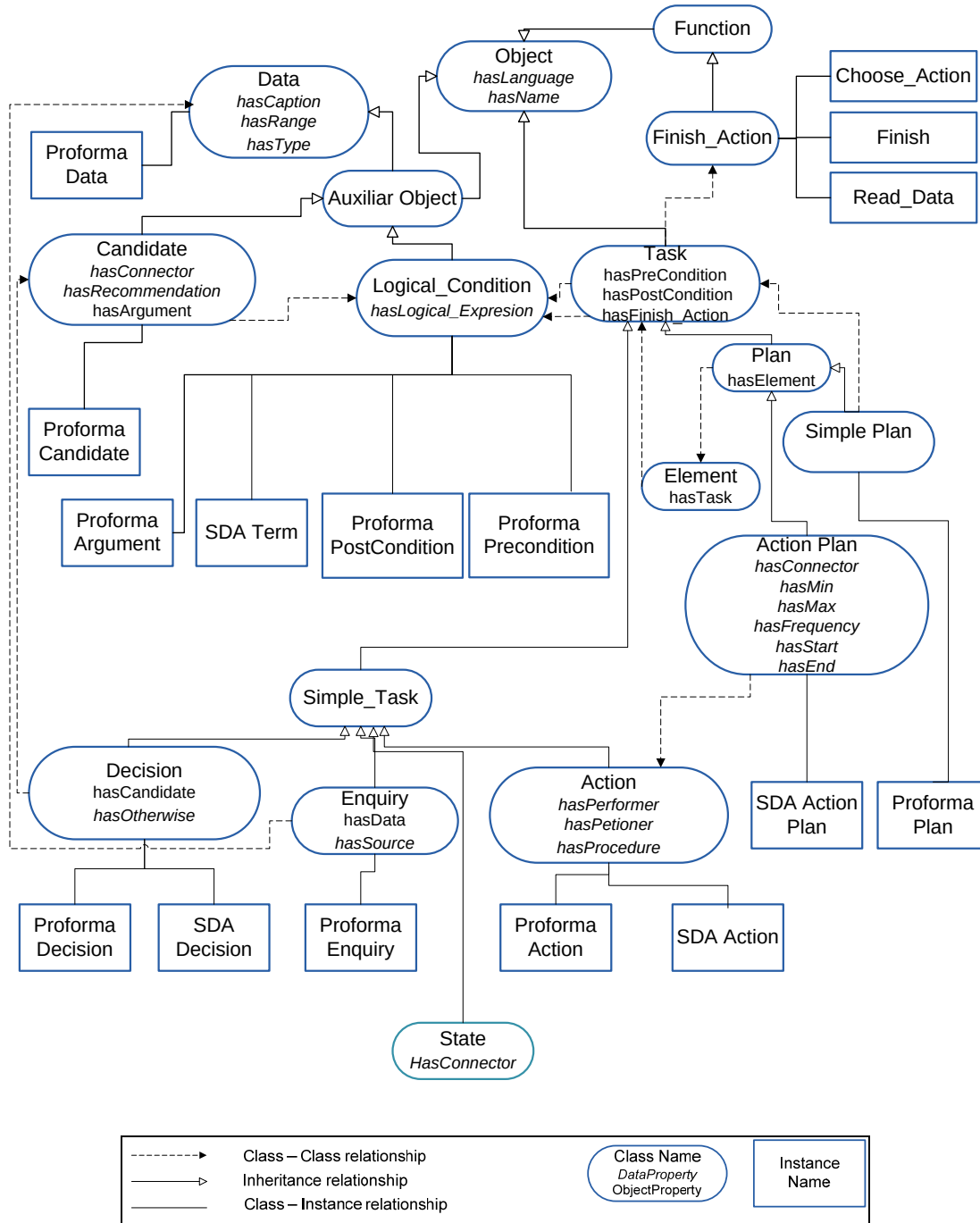


Figure 3.1 Graphical representation of the ontology.

3.3 Classes

The class hierarchy that has been defined is divided into two main parts (see Figure 3.2). The former, named *Object* class, stores the information about all language components. The latter, named *Function* class, stores all dynamic aspects related with the execution of clinical guidelines.

Task and *Auxiliar_Object* inherit from the *Object* class. The *Task* class represents the language basic elements, which are decisions, actions, states and enquiries (modeled in *Decision* class, *Action* class, *State* class and *Enquiry* class, respectively), that were grouped in the *Simple_Task* class. A *Decision* will be a thread in a guideline with different alternatives. An *Action* will be a process that contains some medical operation to perform (as a message to the user interface, or a call to an external application) and generally it is performed according to the preceding decisions. A *State* will be a condition that must be true to continue a sequence and an *Enquiry* will model steps where some user's information is required. The *Plan* class describes a task that contains another tasks (including itself) and leads their execution. This class was divided into two. The *Action_Plan* class describes a *Plan* that only contains actions (a SDA* action block) and the *Simple_Plan* will be a general plan (Proforma plan).

The *Auxiliar_Object* class was defined to represent the particularities of each basic element. The *Candidate* class models the set of alternatives of a *Decision*; the *Element* class models the *Plan* components; and the *Data* class, the information required in the *Enquiry*. Finally, the *Logical_Condition* class models the logical expressions used in *Task* or *Auxiliar_Object*.

Furthermore, the *Function* class models aspects needed to implement the guideline. The *Finish_Action* class describes the way of concluding a task, that is, the action to be taken to complete the execution of a certain component. Three different finish actions were defined. *Finish*, which means that the execution of the sequence is ended and the next element can be processed; *Choose_Action* that implies that the user chooses an alternative, and *Read_Data*, that is used when it is needed that the user provides some data.

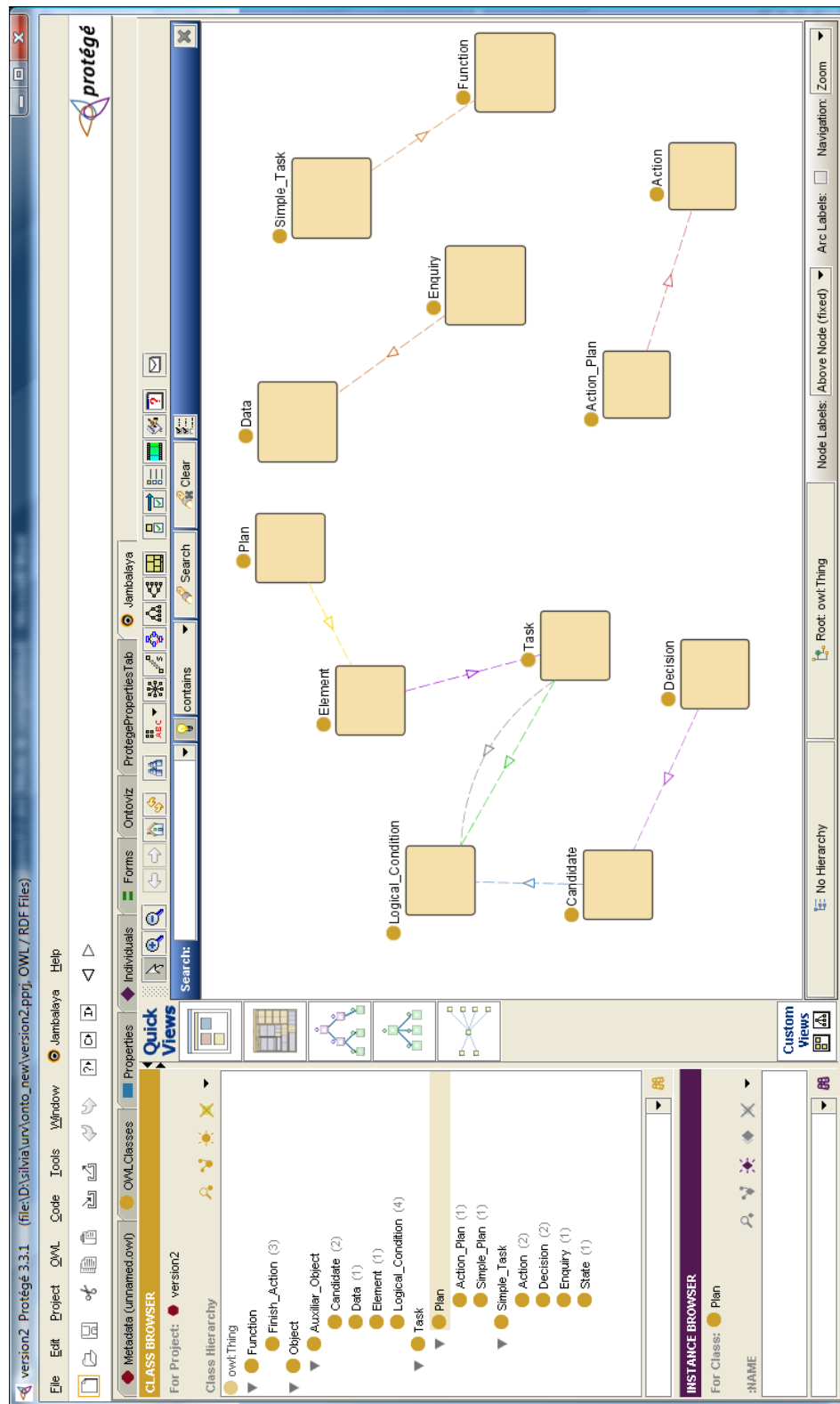


Figure 3.2 Hierarchy of classes defined in the guideline-execution ontology using Protégé. The left hand of the screenshot shows the tree-like structure of classes, and the right hand shows the main object relationships

3.4 Data type properties

Each class can contain information about attributes called *Data type properties*. The attributes defined for each class and a description of their use are shown in Table 3.1.

Class	Data type property	Description
<i>Object</i>	<i>hasLanguage</i>	Guideline representation language
	<i>hasName</i>	Regular expression that allows to find in the guideline code the component modeled by the class
<i>Action</i>	<i>hasPetitioner</i>	Actors that are allowed to request the action
	<i>hasPerformer</i>	Actors that are allowed to perform the action in the real world.
	<i>hasProcedure</i>	Operation to be performed as a result of executing an action
<i>Decision</i>	<i>hasOtherwise</i>	Selected alternative when none of the “normal” alternatives is appropriate
<i>Candidate</i>	<i>hasConnector</i>	Represents the next task to execute when the execution of this component is over
	<i>hasRecommendation</i>	Rule used to express the conditions under which it would be appropriate to commit to that candidate
<i>Logical_Condition</i>	<i>hasLogical_Expression</i>	Logical expression to be evaluated in a given situation (e.g., required conditions to be accomplished before performing an action)
<i>Enquiry</i>	<i>Source</i>	Describes the required user's information. It is encapsulated by the <i>Data</i> class
<i>Data</i>	<i>hasCaption</i>	Represents the question to be asked to the user
	<i>hasType</i>	The data type of the user's answer
	<i>hasRange</i>	List of possible answers to the question. The user's answer should match with one of these values
<i>State</i>	<i>hasConnector</i>	It represents the next task to execute
<i>Action_Plan</i>	<i>hasConnector</i>	It represents the next task to execute
	<i>hasMin</i>	It represents the minimum duration or delay of the execution connector
	<i>hasMax</i>	It represents the maximum duration or delay of the execution connector
	<i>hasStart</i>	It represents the initial time from the observed task
	<i>hasFinal</i>	It represents the final time from the observed task
	<i>hasFrequency</i>	It stores the frequency in which this task should be observed
<i>Function</i>	<i>Finish_Action</i>	Describes all the possible ways to finish or complete the execution of a task

Table 3.1 Defined data properties

3.5 Object properties

The relationship between two or more classes is represented through the use of *Object Properties*. Table 3.2 lists each of the defined relationships and provides a brief description. For example as shown in Figure 3.1, the class *Candidate* has the relationship *hasArgument*. In this case, all instances of that class can contain the information about any argument, which is embedded in the classes *Decision* and *Candidate*.

Object property	Related classes	Description
<i>hasElement</i>	<i>Plan-Element</i>	A plan has a set of elements
<i>hasTask</i>	<i>Element-Task</i>	An element has a set of tasks
<i>hasCandidate</i>	<i>Decision-Candidate</i>	A decision has a set of candidates
<i>hasArgument</i>	<i>Candidate-Logical_Condition</i>	A candidate has a set of arguments
<i>hasPreCondition</i>	<i>Task-Logical_Condition</i>	A task has a pre-condition which is a logical condition
<i>hasPostCondition</i>	<i>Task-Logical_Condition</i>	A task has a post-condition which is a logical condition
<i>hasData</i>	<i>Enquiry-Data</i>	An enquiry has a set of data
<i>hasAction</i>	<i>Action_Plan-Action</i>	An action block has a set of actions
<i>hasFunction</i>	<i>Task - Function</i>	A plan has a finish action function

Table 3.2 List of defined *object properties*

3.6 Instances

In this case, the instances represent how each class is coded in each language, i.e, the class attributes take values depending on the language that represents the instance. Each class has one instance for each language modelled in the ontology and it's described how this class is represented in a given language. This representation is needed because the information of the ontology will be used to allow the execution of guidelines independently of the language in which they have been written. In order to use the ontology for this purpose, it had to be described in such a way that each represented component is associated with the respective sequence in the guideline.

That was the reason for including in the ontology how the different components were coded in each language and then, at the moment of execution, to use this information for reading the corresponding sequence in the guideline, without the necessity of using the parser. That is, regular expressions were used to represent how to find each attribute in the code of the guideline.

The name of each instance was chosen for convenience: LanguageName_ClassName. For example, *Proforma_Decision* is the instance that represents how a decision is coded in PROforma.

The instances defined in each class are shown in Tables 3.3 – 3.9. They are grouped by concept. The values of the attributes are also described as well as the relationships (*Object Properties*) with other classes or more precisely with instances of those classes. Note that if a class does not have instances for PROforma or SDA* or some attribute is null is because this component or this attribute are not modelled in these languages.

<i>Proforma_Simple_Plan</i> : Describes a <i>Simple_Plan</i> in PROforma		
	Name	Value
Attributes	<i>hasLanguage</i> <i>hasName</i>	Proforma plan ::
Relationships	<i>hasElement</i> <i>hasPreCondition</i> <i>hasPostCondition</i> <i>hasFinish_Action</i>	<i>Proforma_Element</i> <i>Proforma_PreCondition</i> <i>Proforma_PostCondition</i> <i>Finish</i> ⁽¹⁾

<i>SDA_Action_Plan</i> : Describes an <i>Action_Plan</i> in SDA*		
	Name	Value
Attributes	<i>hasLanguage</i> <i>hasName</i> <i>hasConnector</i> <i>hasMin</i> <i>hasMax</i>	SDA <sda_actionblock>*</sda_actionblock> <next>*</next> <min>*</min> <max>*</max>
Relationships	<i>hasElement</i> <i>hasFinish_Action</i> <i>hasPreCondition</i> <i>hasPostCondition</i>	<i>SDA_Action</i> <i>Finish</i> <i>null</i> <i>null</i>

⁽¹⁾ *Finish*: It means that the current task has been finished and it is possible to continue to the next process.

Table 3.3 Definition of the instances *Simple_Plan* and *Action_Plan* respectively

Firstly, it presents in Table 3.3 the instances of the class *Simple_Plan* and *Action_Plan*. As it can be seen, the attributes of the instances take value depending on the languages they represent. In this case, the classes are different; therefore they do not share most of the attributes, except those of the classes *Object* (*hasLanguage* and *hasName*), *hasElement*, and *Finish_Action*, because they are derived from *Plan* and *Task* respectively. The attribute *hasLanguage* will always have the language to describe the instance and

hasName the regular expression for reading the corresponding sequence in the code of that language.

In the attribute *hasElement* are defined their components, in the case of *SDA_Action_Plan* is an *Action* (*SDA_Action*), and in *Proforma_Simple_Plan* is a *Proforma_Element*, which are described in Tables 3.5 and Table 3.4 respectively. The attribute *hasFinish_Action* will be used for ending the execution of the component and it will be also modelled in all the tasks derived from *Task*.

In addition in *Proforma_Simple_Plan* the pre-conditions and post-conditions were modelled with logic expressions described by *Proforma_PreCondition* and *Proforma_Postcondition*, which are defined for all the instances of *Proforma*, whose classes derive from *Task*. Notice that when an attribute or a relation is not modelled in a language, its value is null.

<i>Proforma_Element</i> : Describes a plan component in a <i>Proforma</i>		
	Name	Value
Attributes	<i>hasLanguage</i> <i>hasName</i>	<i>Proforma</i> component ::
Relationships	<i>hasTask</i> <i>hasPreCondition</i> <i>hasPostCondition</i>	<i>Proforma_Enquiry</i> <i>Proforma_Action</i> <i>Proforma_Decision</i> <i>Proforma_Simple_Plan</i> <i>Proforma_PreCondition</i> <i>Proforma_PostCondition</i>

Table 3.4 Definition of the information contained in the *Element* instance

The Table 3.4 defines the instances of the class *Element*. In this case only the instance corresponding to *Proforma* is modelled, because the components that constitute the *Proforma_Plan* can be of several types. This relationship is modelled using the *hasTask* relationship.

The instances of the *Action* class are presented in Table 3.5. In this case, different attributes exist for each instance: *hasProcedure* for describing the operation to be carried out in an *Action* in *Proforma* (*Proforma_Action*), and *hasPerformer*, *hasPetitioner*, *hasFrequency*, *hasStart* and *hasEnd* in *SDA_Action*, which are used for defining the actors that will request and execute the action and the temporal constraints.

<i>Proforma_Action</i> : Describes an <i>Action</i> in <i>Proforma</i>		
	Name	Value
Attributes	<i>hasLanguage</i> <i>hasName</i> <i>hasProcedure</i> <i>hasPerformer</i> <i>hasPetitioner</i>	<i>Proforma</i> action :: procedure :: null null
Relationships	<i>hasFinish_Action</i> <i>hasPreCondition</i> <i>hasPostCondition</i>	<i>Finish</i> ⁽¹⁾ <i>Proforma_PreCondition</i> <i>Proforma_PostCondition</i>

<i>SDA_Action</i> : Describes an <i>Action</i> in <i>SDA</i> *		
	Name	Value
Attributes	<i>hasLanguage</i> <i>hasName</i> <i>hasProcedure</i> <i>hasPerformer</i> <i>hasPetitioner</i> <i>hasFrequency</i> <i>hasStart</i> <i>hasEnd</i>	<i>SDA</i> <sda_action>*</sda_action> null <performer>*</performer> <petitioner>*</petitioner> <frequency>*</frequency> <start>*</start> <end>*</end>
Relationships	<i>hasFinish_Action</i> <i>hasPreCondition</i> <i>hasPostCondition</i>	<i>Finish</i> null null

⁽¹⁾ *Finish*: It means that the current task has been finished and it is possible to continue to the next process.

Table 3.5 Definition of the *Action* instances

<i>Proforma_Decision</i> : Describes a <i>Decision</i> in Proforma		
	Name	Value
Attributes	<i>hasLanguage</i> <i>hasName</i> <i>hasOtherwise</i>	Proforma decision :: null
Relationships	<i>hasFinish_Action</i> <i>hasCandidate</i> <i>hasPreCondition</i> <i>hasPostCondition</i>	<i>Choose_Option</i> <i>Proforma_Candidate</i> <i>Proforma_PreCondition</i> <i>Proforma_PostCondition</i>

<i>SDA_Decision</i> : Describes a <i>Decision</i> in SDA*		
	Name	Value
Attributes	<i>hasLanguage</i> <i>hasName</i> <i>hasOtherwise</i>	SDA <sda_decision>*</sda_decision> <otherwise>*</otherwise>
Relationships	<i>hasFinish_Action</i> <i>hasCandidate</i> <i>hasPreCondition</i> <i>hasPostCondition</i>	<i>Choose_Option</i> <i>SDA_Candidate</i> null null

Table 3.6 Definition of the *Decision* instances

The instances of the class *Decision* are shown in Table 3.6, and in both, the candidates were modelled by means of the relationship *hasCandidate*. The instance *SDA_Action* models the alternative otherwise that is used when none of the "normal" alternatives are appropriate.

<i>Proforma_Candidate</i> : Describes a <i>Candidate</i> in Proforma		
	Name	Value
Attributes	<i>hasLanguage</i> <i>hasName</i> <i>hasConnector</i> <i>hasRecommendation</i>	Proforma candidate :: null recommendation ::
Relationships	<i>hasArgument</i>	<i>Proforma_Argument</i>

<i>SDA_Candidate</i> : Describes a <i>Candidate</i> in SDA*		
	Name	Value
Attributes	<i>hasLanguage</i> <i>hasName</i> <i>hasConnector</i> <i>hasRecommendation</i>	SDA <sda_branch>*</sda_branch> <sda_connector>*</sda_connect or> null

Table 3.7 Definition of the *Candidate* instances

The instances of the class *Candidate* are shown in Table 3.7. In the case of *Proforma_Candidate*, it is defined the feature of Proforma which for each alternative there are a group of arguments supporting or opposing the selection, which are presented in table 3.9, through the relationship *hasArgument* and the

instance *Proforma_Argument*. In addition, the attribute *hasRecommendation* was also defined to represent the recommendation rule that *Proforma* offers to choose a candidate.

In the case of *SDA_Candidate*, the attributes previously described are not defined, therefore their value is null. Likewise, the attribute *hasConnector* is defined to represent the next element to be executed if the user chooses this alternative. As this attribute is not defined in *Proforma*, its value is null.

<i>Proforma_Enquiry</i> : Describes an <i>Enquiry</i> in PROforma		
	Name	Value
Attributes	<i>hasLanguage</i> <i>hasName</i> <i>hasSource</i>	<i>Proforma</i> enquiry :: source ::
Relationships	<i>hasFinish_Action</i> <i>hasData</i> <i>hasPreCondition</i> <i>hasPostCondition</i>	<i>Read_Data</i> ⁽¹⁾ <i>Proforma_Data</i> <i>Proforma_PreCondition</i> <i>Proforma_PostCondition</i>

<i>Proforma_Data</i> : Describes a required user's <i>Data</i> in PROforma		
	Name	Value
Attributes	<i>hasLanguage</i> <i>hasName</i> <i>hasCaption</i> <i>hasRange</i> <i>hasType</i>	<i>Proforma</i> enquiry :: caption :: range :: type ::

<i>SDA_State</i> : Describes a <i>State</i> in SDA*		
	Name	Value
Attributes	<i>hasLanguage</i> <i>hasName</i> <i>hasConnector</i>	SDA <sda_state></sda_state> <next>* </next>
Relationships	<i>hasFinish_Action</i> <i>hasPreCondition</i> <i>hasPostCondition</i>	<i>Finish</i> ⁽²⁾ <i>SDA_Term</i> null

⁽¹⁾ *Read_data*: It means that that certain user's information should be requested and obtained.

⁽²⁾ *Finish*: It means that the current task has been finished and it is possible to continue to the next process.

Table 3.8 Definition of all entities that model the input of data

The instances of the classes that are related in with the request of information to the user are grouped in table 3.8. In the case of the classes *Enquiry* and *Data*, the instances are defined only for *PROforma* because it is in this language where specific components are modelled for data entry. In the case of the class *State* happens something similar but with *SDA**, that is, an instance for *Proforma* was not defined, because although similar functions to those of *State* are modelled in this language, this component is not explicitly defined.

The information required by the user is searched through the attribute *hasSource* of the instance *Proforma_Enquiry*, and the data that will be

requested to the user are described exactly in *Proforma_Data*. The attributes *hasCaption*, *hasRange* and *hasType* are used for this purpose and they allow the finding of the precise question that will be asked to the user, as well as the rules to validate the answer. *Proforma_Enquiry* and *Proforma_Data* are related by means of *hasData*.

On the other hand the instance *SDA_State* defines a group of conditions that should be fulfilled so that the patient is in a certain state. They were defined using the relationship *hasPrecondition*, because they are equally logical expressions and a *State* can be seen like a pre-condition of a task, and by means of the instance *SDA_Term*.

<i>Proforma_Argument</i> : Describes an argument <i>Logical_Condition</i> expression in <i>Proforma</i>		
	Name	Value
Attributes	<i>hasLanguage</i> <i>hasName</i> <i>hasExpression</i>	<i>Proforma</i> argument_name :: "string"

<i>Proforma_PreCondition</i> : Describes a pre-condition <i>Logical_Condition</i> expression in <i>Proforma</i>		
	Name	Value
Attributes	<i>hasLanguage</i> <i>hasName</i> <i>hasExpression</i>	<i>Proforma</i> precondition :: "string"

<i>Proforma_PostCondition</i> : Describes a post-condition <i>Logical_Condition</i> expression in <i>Proforma</i>		
	Name	Value
Attributes	<i>hasLanguage</i> <i>hasName</i> <i>hasExpression</i>	<i>Proforma</i> postcondition :: "string"

<i>SDA_Term</i> : Describes a simple <i>Logical_Condition</i> expression in <i>SDA</i>		
	Name	Value
Attributes	<i>hasLanguage</i> <i>hasName</i> <i>hasExpression</i>	<i>SDA</i> <sda_term name>*</term> "string"

Table 3.9 Definition of all instances that express some logical condition to be accomplished

Finally, the instances of the class *Logical_Expression* are grouped in table 3.9. In *Proforma_Argument*, it is defined how the arguments of the candidates are coded in *Proforma*. *Proforma_Precondition* and *Proforma_postCondition* define the pre-conditions and post-conditions respectively in *Proforma* and finally, *SDA_Term* is used to describe the logical conditions in *SDA**.

We also present two screenshots examples to show how the information about instances is modelled in Protégé. Figure 3.3 shows the *Proforma_Action* and Figure 3.4, the *SDA_Action*. The left hand of the screenshot shows the tree-like structure of classes and the number of defined instances, and the right hand shows the values of each attribute of the class (*Action*).

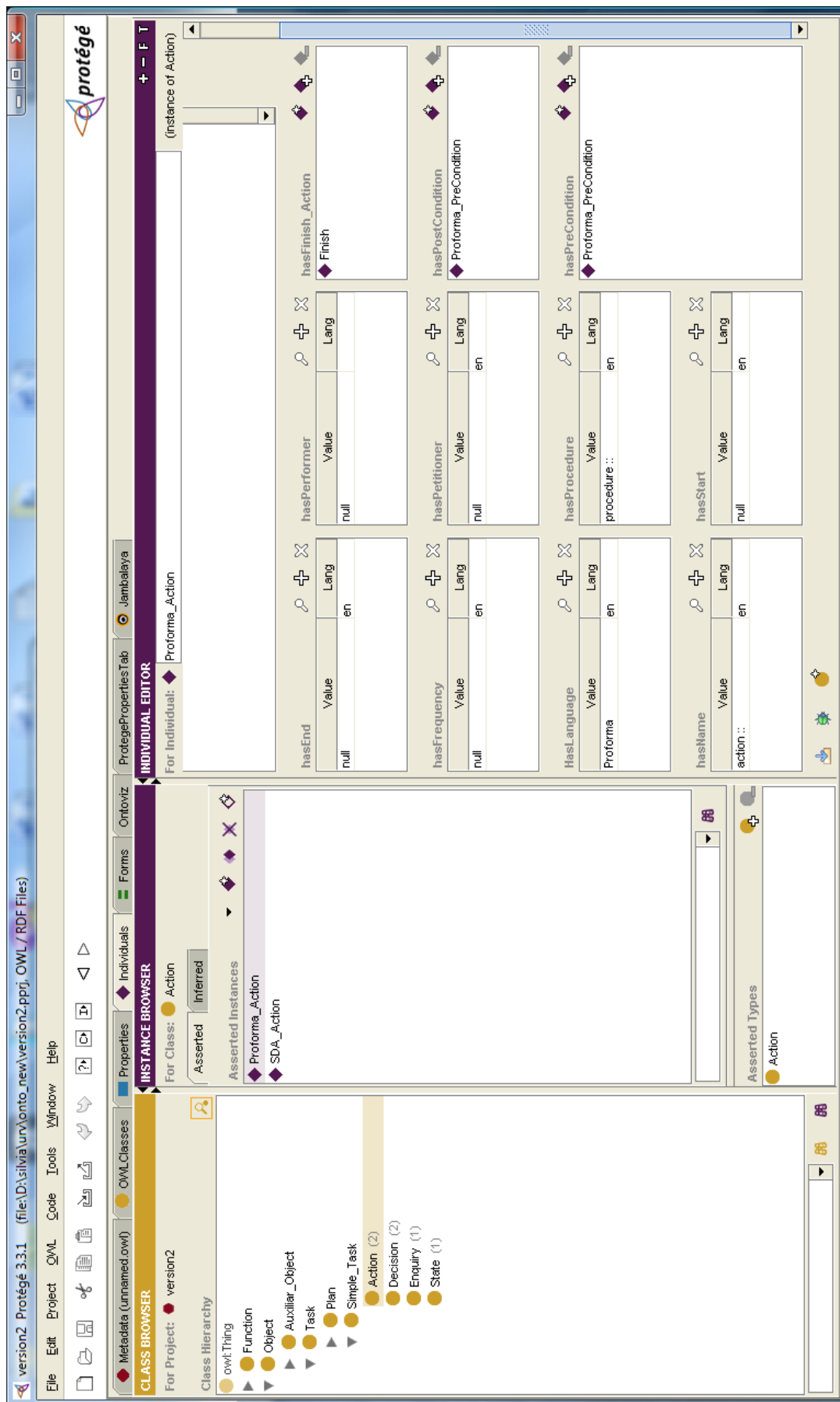


Figure 3.3 Example of the *Protéma_Action* instance modelled using Protégé

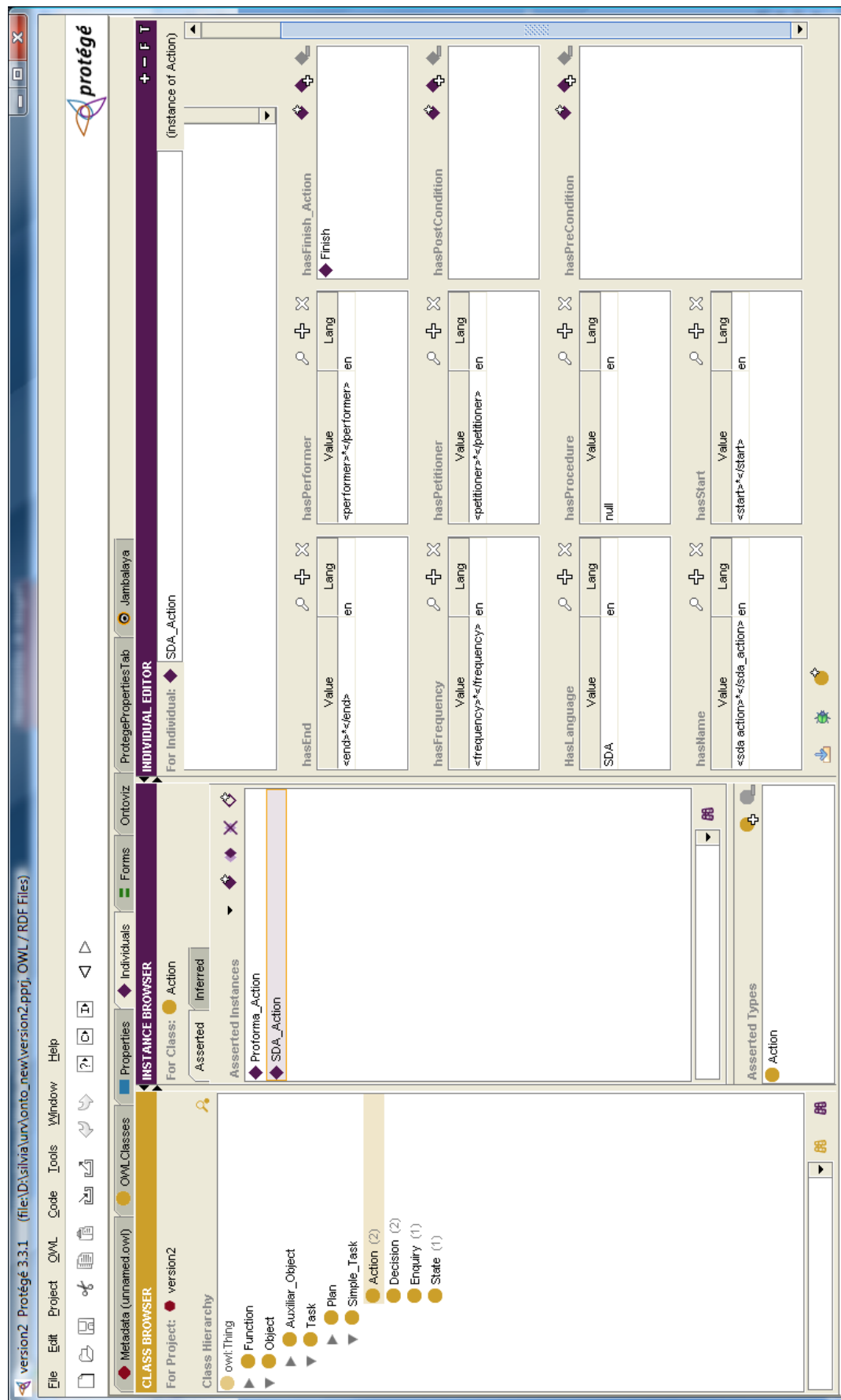


Figure 3.4 Example of the Actions instances in Protégé

3.7 Conclusions

This chapter presented an ontology for representing guideline modeling languages.

- The ontology was built from a common core with the main elements of the studied representation languages and the most important features of each one.
- Each class of the ontology represents a concept of the representation languages, and each instance how this concept is coded in a certain language. Therefore, in order to use the ontology for guideline execution, it was necessary to make explicit in each instance how each component is coded in each specific language. In this way, it is possible to read the sequence corresponding to the component in the guideline code independently of the language used to write it using the information given in the ontology.
- Although the ontology was designed taking into account two representation languages, *PROforma* and *SDA**, it was thought as generic as possible, and it is feasible to include other languages in a relatively simple way, by adding the appropriate instances representing the information related to the new language.
- The values of the attributes of each class contain the necessary data for obtaining the information of that component in the guideline, in different languages. If an attribute is not explicitly defined in a language, its value will be null in the corresponding instance.
- In order to use the information represented in the ontology for the execution of the guideline, it was also necessary to model how to end the execution of each component, according to its type, that is, the action to be taken to complete the execution of an action, a data entry, a decision, etc.

In the next chapter we show how the generic executor uses the information of the ontology to be able to simulate the execution of CGs written in different formalisms.

4 Generic guideline executor simulator

In the previous chapter, an ontology was designed for modelling the main components of different guideline representation languages. The main goal of this work is building a system that allows executing clinical guidelines independently of the used language. The ontology fills the gap between the executor and the guideline representation.

With this objective, an application was created (executor) to perform the main processes of a clinical guideline in different representation languages based on the information represented in the ontology.

4.1 Abstract view of the executor

In order to verify the usefulness of the ontology defined in the previous chapter, an application was built (executor) whose objective is to use the information modelled in this structure to execute the main steps of a clinical guideline, independently of the language used to write it.

The practitioner that will use the clinical guideline for a certain medical treatment will not need to know either the details of its implementation or the language used to write it. The application will be in charge of guiding the user step by step, in a transparent way, along the sequence of activities according to the patient's situation.

For example, if data entries are required in a certain execution stage, they will be requested by the application, or if a decision has to be taken on a certain treatment, the different alternatives will be shown to the user according to the current conditions of the patient.

That is, the executor can access the defined ontology where the characteristics of different languages for guideline representation have been modelled, and it uses this information to execute the guideline and to guide the user in the workflow of the medical treatment. This relationship is shown schematically in figure 4.1.

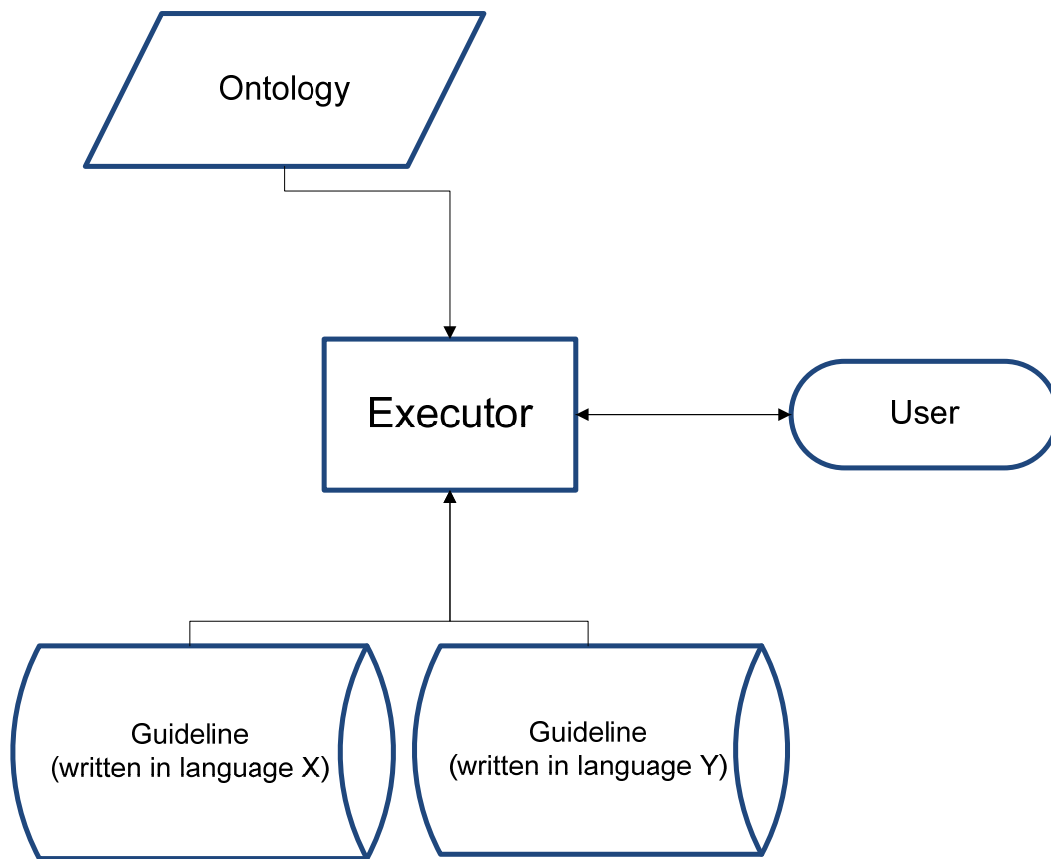


Figure 4.1 Graphical representation of the relationship among the executor, the ontology, the clinical guidelines and the user

4.2 Algorithm for executing clinical guidelines

The main objective of the executor is to run and to process each element of the clinical practical guideline according to the information contained in the ontology.

Initially, the first element to be executed is obtained from the guideline. From this point, until the end of the guideline, each one of the steps is processed according to the information obtained from the ontology. That is, if the execution is in the component T, the ontology is searched for the component type to which it belongs, to discover how it is represented in the clinical guideline according to the language and how its execution is carried out in general (e.g. how its execution is finished). Starting from this information, the object corresponding to that component is searched for in the clinical guideline, and its associated information is shown to the user, finishing the execution of this step. Therefore, processing a component T implies obtaining information about the component, showing it to the user and completing the execution depending on the component type. Afterwards, the guideline is queried again in order to know the next element to execute, and the described process is carried out again until an end point of the guideline is reached.

For example, let's imagine that the next element to be executed in the guide is a decision. The component to model it and how it is represented in the clinical guide are searched for in the ontology. The information of the decision is shown to the user and the sequence is finished, with the function that is also obtained from the ontology; in this case, a decision implies the selection of an alternative by the user. Then it is searched in the ontology how these alternatives are represented in the guideline, and they are read and shown to the user for selection.

This is the execution algorithm, which is shown in figure 4.2.

Main sequence (guideline G, language L, ontology O) T:= Get_first_element_from_guideline (G,L) WHILE NOT End _guideline(G) DO RUN (T, O); T= Get_following (G,L); END WHILE
RUN sequence RUN (T,O) Cl:= Get_information_task (T,O) Show_by_screen (Cl) Finish_Action (T,O)

Figure 4.2 Algorithm used in the development of the executor

4.3 Accessing the ontology and the clinical guidelines

To carry out the algorithm described in the previous section, the executor interacts with the ontology and the guidelines written in different languages, relationship that was represented in figure 4.1. In order to achieve the communication with each component (ontology and guidelines), it is necessary a group of elements that allows the interaction among them.

4.3.1 Interface to access the ontology

In the previous chapter was described the designed ontology where the main elements of different representation languages were modelled, in such a way that the execution of a guideline could be possible, independently of the language used. With this objective, a group of classes, attributes, relationships

among classes and instances were defined where the necessary information will be obtained to carry out the execution.

To access this information, it is necessary to use a component for reading the content of the ontology according to the language in which it is written, in this case OWL (McGuinness & van Harmelen, 2004).

Then, using this parser or interface, different functions were created (which are explained in section 4.4) to recover the information of the represented components according to the executor's necessities. For example, a method was created that, given a concept and a representation language, searches for the class associated to this concept in the ontology, and returns the instance of this class which represents the concept searched in the given language.

4.3.2 Language parsers

Besides the information modelled in the ontology, for the executor to work properly it is necessary to have a component for reading the guideline according to the used language. That is, it is necessary to have the parser of each representation language. As it has been mentioned in the previous chapter, the ontology was designed for this work taking as example two languages, *PROforma* and *SDA**, and these two languages were used to test the execution, therefore, it was necessary to have the parser of both ⁽¹⁾. One of the goals of the work was to build a prototype as independent as possible of the final language representation. In this sense, only a few features of these parsers were used, such as loading a guideline and getting the main elements contained in it. More particular functions such as getting a particular value for an attribute, or the connections between the elements were written ad-hoc for our prototype.

The development of the executor was based on the principle to use the parser as less as possible, to achieve the highest possible independence, giving more importance to the ontology and, in this way, contributing to the generality of the application. That was the reason of including in the ontology how the different components were coded in each language and then, in the moment of the execution, to use this information for reading the corresponding sequence in the guideline, without the necessity of using the parser. Finally, the decision was taken of using the parser of each language only in two specific points: to know the first element to execute and to obtain the next one.

⁽¹⁾ The parser of *PROforma* is embedded into the Tallis executor tool, which can be requested for evaluation at <http://www.acl.icnet.uk/lab/tallis/index.html>. The parser of *SDA** was developed under the K4Care project. It is not freely available.

4.4 Class Diagram

Finally, the developed application is formed by four basic classes:

HighAccessOntoOWL: It allows the retrieval of the necessary information of the ontology which will be used by the application to execute the guidelines. It contains several methods; the most important are the following:

- `getConcept (nameClass)`: given a class name, returns the associated class.
- `getRelationship (class, nameRelationship)`: given a relationship name it returns the associated relationship.
- `getInstance (class, language)`: returns the instance that corresponds to the given language.
- `String getObjectPropertyValue (property, language)`: given an attribute (*object property*), a class and a language, returns the value of this attribute in the instance corresponding to the language.
- `List getObjectPropertyValues (task, language)`: similar to above, but in this case it returns a list of *object properties*.
- `String getDataPropertyValue (property, language)`: given an attribute (*data property*), a class and a language, returns the value of this attribute in the instance corresponding to the language.
- `getFinishTask (task, language)`: given a task and a language, returns the finishing action to complete the execution of the task (*Finish, Read_Data, Choose_Option*).

Parser_SDA or Parser_Proforma: it allows the access to the parser of SDA* or Proforma respectively, both have similar functions.

- `getFirst()`: to return the first element to execute.
- `getNext()`: to return the next element to execute.

Main: It is the controlling class and it contains the main functions to invoke the methods of the rest of the classes. It includes functions such as:

- `getFirstComponent(task, language)`: requests the first element to execute to the *Parser_SDA* or *Parser_Proforma*.
- `getNextComponents(task, language)`: requests the next element to execute to the *Parser_SDA* or *Parser_Proforma*.
- `getInfoComponent (task, language)`: returns the information of a component, that is, it is in charge of invoking the functions to obtain the necessary information from the ontology, and then to read the corresponding sequence in the guideline.

- `showInfoComponent (task, language)`: shows the component information to the user.
- `runComponent (task, language)`: It includes all the necessary functions to execute a task. First, it invokes the method `getInfoComponent ()` which returns the information associated to this task in the guideline, and it is shown to the user by means of `showInfoComponent ()`. Afterwards `getFinishTask ()` is invoked in order to know how to end the execution of the component. Depending on this answer, if it is necessary, the previous procedure is carried out for searching the necessary information, and finally it is shown to the user. For example, if the component is a decision, the ending action is *Choose_Option* and it implies that the information associated to the alternatives of this decision should be searched and shown to the user for choosing an option.

The class diagram of the developed application is shown in Figure 4.3.

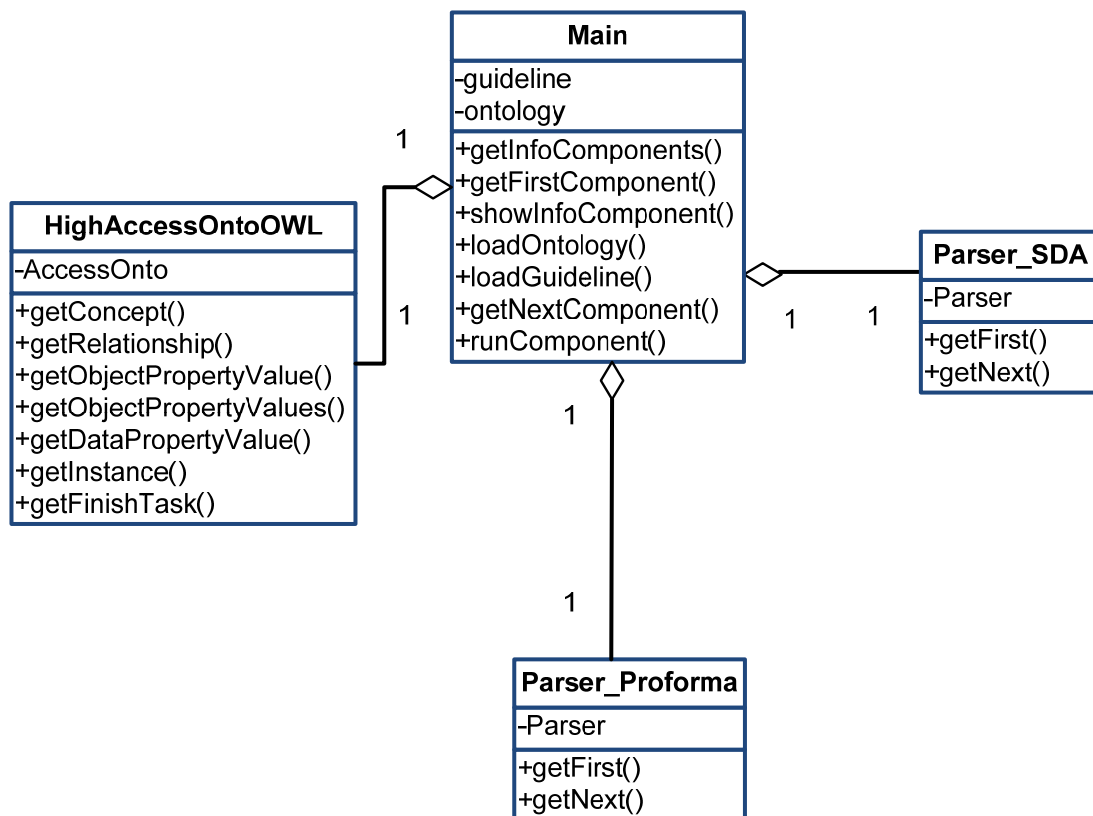


Figure 4.3 Class diagram of the application (executor)

4.5 Execution

Once explained the main elements that compose the developed application, the execution process will be explained in more detail.

Two examples of guidelines written in PROforma and in SDA* were selected in order to verify the usefulness of the developed ontology. In the case of PROforma a guideline for the treatment of the hypertension was selected. In the case of SDA*, the Home Care Patient (HCP) Discharge⁽²⁾ was selected. Both guidelines are shown in Annex 1.

4.5.1 Simulation of the execution of a guideline in Proforma

Next, the execution of a guideline in PROforma will be explained with more detail, according to the algorithm presented in figure 4.2. An example is used to show the execution of a guideline. The result of each step of this execution is shown in figure 4.7. The highlighted comments are explanations to clarify the execution to the reader, but they are not part of the information that is shown to the final user.

1. The first step is to obtain the element that begins the execution by means of the parser.

In the case of PROforma, it is usually a component of the type *Simple_Plan*, which drives the execution of the guideline. It has a set of components and they are executed in a precise order (see section 2.1.3). Concretely the first component is a plan named "*Hypertension*".

2. Once the component has been obtained, the next step is to run it. To run the component, the following steps are taken.
 - First, it is necessary to obtain the information allowing the location of the *Simple_Plan* "*Hypertension*" in the guideline.

For this purpose, the associated class is searched for in the ontology, as well as the instance of PROforma which contains the code of the component for this language. In this case, a component is a *Simple_Plan* and the needed information is the attribute name, eg "plan ::" (see table 3.3). With this regular expression and the name of the component obtained in the step 1 ("*Hypertension*"), the corresponding (*plan ::* "*Hypertension*") sequence is searched in the guideline. The code segment associated to this component is shown in Figure 4.4. The components that make up the "*Hypertension*" (*components ::*) will be processed in the next steps.

⁽²⁾ Clinical Guideline to decide the patient's discharge from the Home Care.

Note how the value of the attribute represents the components coded in each language and allows the recovery of its information in the corresponding sequence of the code guideline and it is show to the user independently of the representation language.

- Then, the information of the component is shown to the user: in this case "*Hypertension*".

```

plan :: 'Hypertension' ;
caption :: "Hypertension";
component :: 'Current_therapy' ;
    number_of_cycles :: 1;
    ltwh :: 33,118,42,36;
component :: 'Comorbidities' ;
    number_of_cycles :: 1;
    ltwh :: 203,186,42,36;
component :: 'Additional_drug_choice' ;
    schedule_constraint :: completed('Comorbidities') ;
    schedule_constraint :: completed('Current_therapy') ;
    number_of_cycles :: 1;
    ltwh :: 182,74,36,36;
component :: 'Target_BP_decision' ;
    autonomous :: yes ;
    schedule_constraint :: completed('Comorbidities') ;
    number_of_cycles :: 1;
    ltwh :: 358,194,36,36;
component :: 'Prescribe_additional_drug' ;
    schedule_constraint :: completed('Additional_drug_choice') ;
    schedule_constraint :: completed('Target_BP_decision') ;
    number_of_cycles :: 1;
    ltwh :: 478,82,36,36;
component :: 'Assess_response' ;
    schedule_constraint :: completed('Prescribe_additional_drug') ;
    number_of_cycles :: -1 ;
    ltwh :: 576,154,48,36;
end plan.

```

Figure 4.4 Proforma code that correspond to the *Simple_Plan* "*Hypertension*"

- Finally, we need the way for concluding the execution of this task, that is, the action to be taken when this element is present.

Again, it is necessary to request the ontology to obtain it. This information is modelled in the class *Finish_Action* and the possible actions are *Finish*, *Choose_Option* or *Read_Data*. In this case the action to carry out for a *Simple_Plan* component in Proforma is *Finish* (see table 3.3), which means that the execution of the sequence is just ended and the next element can be processed.

Independently of the component type, it is always verified if there are defined conditions to be evaluated. For example, all components have defined the relationship *hasPrecondition* and *hasPostcondition* for modelling the relationship between a task and a *Logical_Condition*. Through these

relationships, it is obtained how these conditions are coded in *Proforma* using the *Proforma_PreCondition* or *Proforma_PostCondition* instances of the class *Logical_Condition* (see table 3.9).

In this case, the information modelled in the ontology allows to create a component that is executed independently of the language representation, including the possible conditions that must be evaluated before or after the execution.

3. The next element to execute is obtained. Because the previous component was a *Simple_Plan*, the next one should be an *Enquiry*, an *Action*, a *Decision* or a *Simple_Plan* according to the definition of *Simple_Plan*. That is because this type of component contained a group of tasks which are modelled by the relationship *hasElement* (see table 3.2). Then, a class *Element* described exactly what kind of task it has, and in the case of *Proforma* instances are *Proforma_Enquiry*, *Proforma_Action*, *Proforma_Decision* and *Proforma_Simple_Plan* (see the table 3.4). In this case the next component is an *Enquiry*.

Then, the same procedure as in step 1 is carried out for finding the corresponding sequence in the guideline and processing it, the code segment associated to this *Enquiry* is shown in Figure 4.5. The information of the component "*Current_therapy*" is shown to the user (*component :: "Current_therapy"* which is part of "*Hypertension*" as is shown in Figure 4.4) and it is searched how to end this task. In this case, the action to carry out is *Read_Data* (see table 3.8).

This action means that information is required from the user. In this case the same procedure as in step 2 is executed again to find in the ontology what information should be asked to the user. The attribute *hasSource* of an *Enquiry* (*source:: "Current_Rx"*) contained a link to this data and then, when the associated sequence was found in the guideline, it is necessary to find the *Proforma_Data* according the relationship *hasData* (see table 3.8). The *Proforma_Data* is found in the guideline according to its name, and the information necessary is shown to the user by the attribute *hasCaption* (*caption ::*).

```
enquiry :: 'Current_therapy' ;
  caption :: "Current therapy";
  source :: 'Current_Rx' ;
end enquiry.

data :: 'Current_Rx' ;
  type :: text ;
  caption :: "Patients current therapy";
  range :: "ACEI", "Beta_blocker_ISA", "Beta_blocker_non_ISA",
  "Ca_blocker_DHP_long", "Ca_blocker_DHP_short", "Ca_blocker_non_DHP", "Thiazide_diuretic", "Loop_diuretic", "K_sparing_diuretic";
end data.
```

Figure 4.5 *Proforma* code that correspond to the *Enquiry* "*Current_therapy*" and their *Data* "*Current_Rx*"

The information that is requested to the user is validated by means of a group of defined attributes for this type of component. For example, the answer can be between a possible range of values or it can be of the true / false type. This is possible according to the information modelled in the instance *Proforma_Data* by the attribute *hasRange* and *hasType* (*range::* and *type::* respectively, see Figure 4.5). This information is shown in Figure 4.7a, as well as the results of the steps 1, 2 and 3.

Hypertension The components to be executed in "<i>Hypertension</i>" are, in this order: <i>Current therapy</i>, <i>Comorbidities</i>, <i>Additional drug choice</i>, <i>Target BP decision</i>, <i>Prescribe additional drug</i> and <i>Assess response</i>.
Current therapy Enter data <i>Patients current therapy (Choose one)</i> - "ACEI" - "Beta_blocker_ISA" - "Beta_blocker_non_ISA" - "Ca_blocker_DHP_long" - "Ca_blocker_DHP_short" - "Ca_blocker_non_DHP" - "Thiazide_diuretic" - "Loop_diuretic" - "K_sparing_diuretic"
The execution waits until the user gives the information and it should coincide with one of the possible given values.

Figure 4.7a Example of the execution of a guideline in PROforma, steps 1, 2 and 3.

- The next element to be executed is obtained and the same procedure described in step 2 is carried out. In this case, it is also an *Enquiry*, "*Comorbidities*" and several data are requested to the user. The information requested to the user in this step is shown in figure 4.7b.

Comorbidities Enter data Patients age group (Choose one) -Younger -Old Asthma Enter data Does the patient have asthma? (Choose one) -Yes -No COPD Enter data Does that patient have COPD? (Choose one) -Yes -No Heart_failure Enter data Does the patient have heart failure? (Choose one) -Yes -No Isolated_Sys_hypertension Enter data Isolated systolic hypertension? (Choose one) -Yes -No MI Enter data Previous MI? (Choose one) -Yes -No Pregnancy Enter data Is the patient pregnant? (Choose one) -Yes -No STD_Heart_block Enter data Second or third degree heart block (Choose one) -Yes -No Systolic_dysfunction Enter data Systolic dysfunction (Choose one) -Yes -No Type_1_Diabetes Enter data Type 1 diabetes? (Choose one) -Yes -No
The execution waits until the user gives the information and it should coincide with one of the possible given values.

Figure 4.7b Example of the execution of a guideline in PROforma, step 4.

5. The following component to be executed is a decision. In this case, the action associated to this type of component is *Choose_Option* (see table 3.6), which says to the executor that the user should choose an alternative. Then the alternatives to this decision are searched using the relationship *hasCandidate* (see table 3.6), which defined the relation between a *Decision* and their *Candidates*. Finally using this class, it will find how the alternatives are coded in *Proforma* by the instance *Proforma_Candidate* (see table 3.7). Then it follows the same operation as in step 2, and the information about the alternative is shown to the user. The code segment associated to this *Decision* (decision :: “*Additional_drug_choice*”) and their *Candidates* (candidate :: “*ACEI*”) is shown in Figure 4.6.

```

decision :: 'Additional_drug_choice' ;
caption :: "Additional drug choice";
candidate :: 'ACEI' ;
argument :: confirming, ( Type_1_Diabetes = Yes and Proteinuria <> None)
attributes
argument_name :: ' ( Type_1_Diabetes = Yes and Proteinuria <> None ) ' ;
end attributes ;
argument :: confirming, ( Heart_failure = Yes ) attributes
argument_name :: ' ( Heart_failure = Yes ) ' ;
end attributes ;
(...)
candidate :: 'Loop_diuretic' ;
argument :: confirming, ( Heart_failure = Yes ) attributes
argument_name :: ' ( Heart_failure = Yes ) ' ;
end attributes
(...)
candidate :: 'Ca_channel_dihydro_short' ;
(...)
candidate :: 'Ca_channel_dihydro_long' ;
(...)
candidate :: 'Non_ISA_beta_blocker' ;
(...)
candidate :: 'ISA_beta_blocker' ;
(...)
candidate :: 'Thiazide_diuretic' ;
(...)
candidate :: 'Ca_channel_non_dihydro' ;
(...)
end decision.

```

Figure 4.6 Proforma code that corresponds to the *Decision* “*Additional_drug_choice*” and their *Candidates*

In addition to displaying the information of the alternatives, it is also verified if they have any associated condition. In the case of *Proforma*, the arguments that support or oppose the selection of a certain candidate are also shown. This can be found also by the ontology, using the relationship *hasArgument* defined also in the *Candidate* class. The alternative arguments can be found using the instances *Proforma_Argument* of the class *Logical_Condition* (see table 3.9). The segment code associated is found using the value *argument_name::* of the attribute *hasName* (see Figure 4.6).

The result of this step is shown in figure 4.7c.

Additional_drug_choice

Choose_Option

1. ACEI

- confirming Type_1_Diabetes = Yes and Proteinuria <> None
- confirming Heart_failure = Yes
- confirming MI = Yes and Systolic_dysfunction = Yes
- excluding Pregnancy = Yes
- excluding Current_Rx = K_sparing_diuretic
- excluding Current_Rx = ACEI

2. ISA_beta_blocker

- excluding Asthma = Yes or COPD = Yes
- excluding STD_Heart_block = Yes
- excluding Current_Rx = Ca_blocker_non_DHP
- for Current_Rx = Thiazide_diuretic
- for Current_Rx = Ca_blocker_DHP_long or Current_Rx = Ca_blocker_DHP_short
- excluding Current_Rx = Beta_blocker_ISA or Current_Rx = Beta_blocker_non_ISA
- against Type_1_Diabetes = Yes and Proteinuria <> None

3. Non_ISA_beta_blocker

- excluding Asthma = Yes or COPD = Yes
- excluding STD_Heart_block = Yes
- confirming MI = Yes
- excluding Current_Rx = Ca_blocker_non_DHP
- for Current_Rx = Thiazide_diuretic
- for Current_Rx = Ca_blocker_DHP_long or Current_Rx = Ca_blocker_DHP_short
- excluding Current_Rx = Beta_blocker_non_ISA or Current_Rx = Beta_blocker_ISA
- against Type_1_Diabetes = Yes and Proteinuria <> None

4. Ca_channel_dihydro_long

- confirming Isolated_Sys_hypertension = Yes and Age_group = Older
- excluding Current_Rx = Ca_blocker_non_DHP
- for Current_Rx = Beta_blocker_ISA or Current_Rx = Beta_blocker_non_ISA
- excluding Current_Rx = Ca_blocker_DHP_long or Current_Rx = Ca_blocker_DHP_short or Current_Rx = Ca_blocker_non_DHP
- for Type_1_Diabetes = Yes and Proteinuria <> None

5. Ca_channel_non_dihydro

- excluding STD_Heart_block = Yes
- excluding Current_Rx = Beta_blocker_ISA or Current_Rx = Beta_blocker_non_ISA
- excluding Current_Rx = Ca_blocker_DHP_long or Current_Rx = Ca_blocker_DHP_short
- excluding Current_Rx = Ca_blocker_DHP_long or Current_Rx = Ca_blocker_DHP_short or Current_Rx = Ca_blocker_non_DHP
- for Type_1_Diabetes = Yes and Proteinuria <> None

6. Thiazide_diuretic

- confirming Isolated_Sys_hypertension = Yes and Age_group = Older
- for Current_Rx = Beta_blocker_ISA or Current_Rx = Beta_blocker_non_ISA

-excluding Current_Rx = Thiazide_diuretic or Current_Rx = K_sparing_diuretic or Current_Rx = Loop_diuretic 7. Loop_diuretic -confirming Heart_failure = Yes -confirming Isolated_Sys_hypertension = Yes and Age_group = Older -excluding Current_Rx = Thiazide_diuretic or Current_Rx = K_sparing_diuretic or Current_Rx = Loop_diuretic 8. K_sparing_diuretic -confirming Isolated_Sys_hypertension = Yes and Age_group = Older -excluding Current_Rx = ACEI -excluding Current_Rx = Thiazide_diuretic or Current_Rx = K_sparing_diuretic or Current_Rx = Loop_diuretic 9. Ca_channel_dihydro_short -for Type_1_Diabetes = Yes and Proteinuria <> None -excluding Current_Rx = Ca_blocker_non_DHP -excluding Current_Rx = Beta_blocker_ISA or Current_Rx = Beta_blocker_non_ISA -confirming Isolated_Sys_hypertension = Yes and Age_group = Older
The execution waits until the user chooses an alternative.

Figure 4.7c Example of the execution of a guideline in *Proforma*, step 5

6. The next element to execute is also a decision "*Target_BP_decision*".

In addition to showing the alternatives, as seen in step 2, if a component has conditions, they are shown to the user. In this case, the decision has a *Proforma_Postcondition*. This condition is very important because depending on the option chosen by the user, two variables *TargetSBP* and *TargetDBP* take different values, which determine the end of the treatment in the execution of the last component "*Assess_response*". The information of this step is shown in Figure 4.7d.

Target_BP⁽³⁾_decision Choose_Option 1. Standard -for Proteinuria = None and Type_1_Diabetes = No 2. Medium -for Type_1_Diabetes = Yes or Proteinuria <= 1 g / 24 hrs 3. Low -confirming Proteinuria = >1 g / 24 hrs TargetSBP = if(result_of(Target_BP_decision) = Standard, 140) TargetSBP = if(result_of(Target_BP_decision) = Medium, 130, 125) TargetDBP = if(result_of(Target_BP_decision) = Standard, 90) TargetDBP = if(result_of(Target_BP_decision) = Medium, 85, 75) The execution waits until the user chooses an alternative. The post-condition means: If the alternative "Standard" is chosen TargetSBP=130 and TargetDBP= 90 If the alternative "Medium" is chosen TargetSBP= 125 and TargetDBP= 85 This value will be used in the execution of the Assess_response component to finish the treatment
--

Figure 4.7d Example of the execution of a guideline in Proforma, step 6

7. The next sequence to be executed corresponds with an *Action* component, "*Prescribe_additional_drug*". In this case, the associate action is *Finish*, and the next element will be processed. The result of this step is shown in figure 4.7e.

Prescribe_additional_drug Do <i>Prescribe additional drug as chosen.</i>

Figure 4.7e Example of the execution of a guideline in Proforma, step 7

8. The last sequence to be executed corresponds to another *Simple_Plan* (as a subplan) "*Assess_response*". In this case, this plan executes two tasks in

⁽³⁾ BP: Blood Pressure.

this order, an *Enquiry* “*Monitor_BP*”, and an *Action* “*Await adequate response*” and it has the condition to finish the treatment. This condition is: “*DBP < targetDBP and SBP < targetSBP*” where:

DBP: Information that should be provided by the user.

SBP: Information that should be provided by the user.

targetDBP and targetSBP: Variables whose values depend on the alternative selected by the user in the “*Target_BP_decision*” (see Figure 4.7d)

The result of this step is shown in figure 4.7f.

Assess_response
The components to be executed in “ <i>Assess_response</i> ” are in this order: Monitor_BP and Await adequate response .
Monitor_BP Enter Data Diastolic BP (Numeric Value) Enter Data Systolic BP (Numeric Value)
The execution waits until the user gives the information and it should coincide with one of the possible given values.
Await adequate response Do Continue therapy until target BPs reached

Figure 4.7f Example of the execution of a guideline in Proforma, step 8

4.5.2 Simulation of the execution of a guideline in SDA*

The execution of the guideline in SDA* is very similar to the one carried out in PROforma, and the result of the execution is shown in figure 4.8. As in the previous case, the highlighted comments are additional explanations for the reader.

1. As discussed in section 2.2.4, the execution of the SDA* guideline starts at any of the states of the SDA* that are feasible entry points and, starting from this point, the treatment that will be given to the patient begins, that is, each component begins to be processed. In this case, there is a component *State* “*START*” which has associated the ending action *Finish*, therefore after showing the associated information to the user, the next element is executed.
2. The first component to be executed is obtained by means of the parser. In this case, the execution starts in a *Decision* “*Why is the HCRequest*

rejected? ". With the information obtained from the ontology about the way of coding this element in SDA* (see table 3.6), the information is shown to the user. Each alternative is presented, and the user should choose one of them, but if none are appropriate then the otherwise branch is selected. Depending on the alternative chosen, one way or another is taken. The results of the steps 1 and 2 are shown in figure 4.8a.

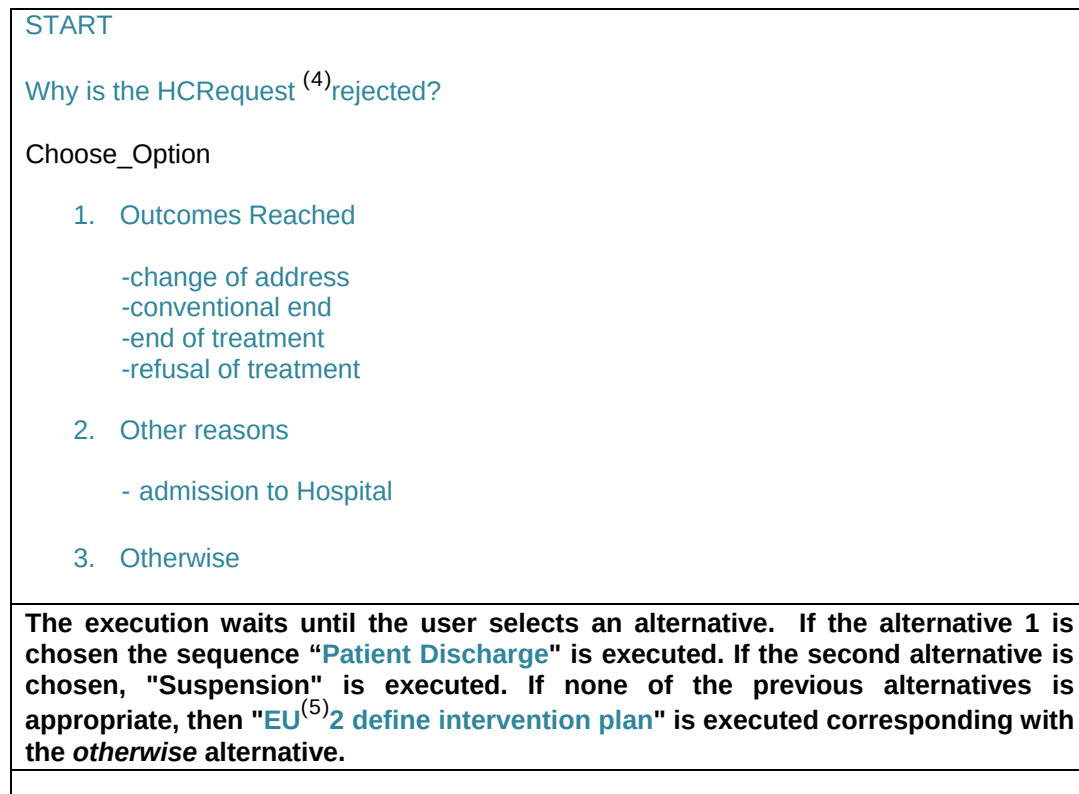


Figure 4.8a Example of the execution of a guideline in SDA *, steps 1 and 2

3. If the alternative 1 is selected, the next element to execute is a component *Action* "Patient Discharge". According to the ontology, this component has associated a *petitioner* and a *performer*; therefore they are also shown to the user. The next sentence to execute is again a *State*, in this case, an END, and the execution of the guideline is ended.
4. If the alternative 2 is selected, it goes towards a component *State*, showing only the associated information, and the next sentence is executed that in this case is also a component *Decision* "Should the patient be discharged from Hospital? ". In this decision there is only one alternative that if it is selected, the execution returns to the beginning of the guideline but if it is not appropriate, the otherwise path is followed and the execution of the guideline is ended.

⁽⁴⁾ Home Care (HC) Request permits the system to know if a person needs care.

⁽⁵⁾ EU: Evaluation Unit.

Selection of alternative 1
Do Patient Discharge petitioner[HN] ⁽⁶⁾ petitioner[PC] performer[HN] performer[PC] END
Selection of alternative 2
SUSPENSION Should the patient be discharged from Hospital? Choose_Option 1. Discharge from Hospital 2. Otherwise
The execution waits until the user selects an alternative. If the alternative 1 is chosen the sequence START is executed but if the alternative <i>otherwise</i> is selected the sentence END is executed.
Selection of the alternative 1
START
Selection of the alternative 2
END

Figure 4.8b Example of the execution of a guideline in SDA *, steps 3 and 4

⁽⁶⁾ PC: Physician in Charge of Home Care Unit
HN: Head Nurse

4.6 Conclusions

The application built for simulating a generic executor of clinical guidelines was presented in this chapter.

- This application was built with the aim at achieving the biggest possible generality, not only from the programming point of view, but also, looking for the greatest possible independence from the respective parsers of each language, giving more importance to the ontology.
- The ontology was tested with the execution of guidelines written in two languages, SDA* and PROforma.
- The main conclusion is that the basic elements needed to execute a guideline are the following:
 - A way to obtain the first element of the guideline.
 - A way to obtain, of each execution point, which is the next element of the guideline.
 - A method to execute each kind of basic element.
- Furthermore, to execute a basic element, we need:
 - Knowledge about how the element is coded in a given language.
 - A method that applies this knowledge to analyse the source coded and obtain all the attributes of the elements.
 - Specific knowledge on how to end the execution of the element

5 Conclusions and future work

A summary of the developed tasks and the conclusions will be presented in this final chapter, summarising the main ideas. Then, several aspects will be recommended for future research.

Up to now, each of the stages for fulfilling the proposed objectives of the present work has been described. Firstly, different clinical guidelines representation languages were studied to find the most important elements of each representation. Then an ontology was designed to unify these elements in a generic structure that allows the execution of guidelines independently of the language used to write them. Finally, the application was presented to test the designed ontology and the processing of two guidelines written in different languages was shown as examples.

- All the analyzed languages are composed of basic tasks which can be treated as classes with their attributes and although they have differences, in most of the cases they represent common concepts to all of them (a thread with different alternatives, a process containing some medical operations to be performed, a task that controls another task, conditions that must be true for continuing a sequence, entities for data entry when the user's information is required, etcetera).
- There are also specific elements explicitly defined in each language. Although these elements seem specific for each language, in many cases similarities between them can be found and, in other cases, they are defined implicitly in other languages.
- Two fundamental aspects should be considered in order to design the ontology allowing the guideline execution independently of the representation language and in the most possible generic way. The first one is to include the way for associating the information modelled in the ontology with the corresponding sequences in the guideline. For this reason, the instances represent each concept in each language, and they contribute to the generality of the solution, the attributes, and the code of each component in the specific language. On the other hand, it was also necessary to model the action to be taken for completing the processing of each component.
- Using the information modelled in the ontology for carrying out the guideline execution allows the execution of the clinical guides in a transparent way to the final user and without the necessity of knowing details of the representation language used.

As future work, it would be very important to improve the designed ontology, that is, to model other elements or features and other languages.

Also it will be interesting to complete the prototype of the created application. For example, it would be very useful to be able to evaluate the logical conditions of the components, at least, those that are more important, that would give more realism to the execution of the current guideline. In addition, it would include in the guideline execution the used of the temporal constraints, and the process of actions in parallel.

References

- AGREE, C. (2003). Development and validation of an international appraisal instrument for assessing the quality of clinical practice guidelines: the AGREE project. *Quality and Safety in Health Care*, 12(1), 18-23.
- Barahona, P., Azevedo, F., Veloso, M., Estêvão, N., & Gallego, R. (2001). Computerising a guideline for the management of diabetes. *International Journal of Medical Informatics*, 64(2-3), 275-284.
- Blaser, R., Schnabel, M., Biber, C., Bäumlein, M., Heger, O., Beyer, M., Opitz, E., Lenz, R., & Kuhn, K. A. (2007). Improving pathway compliance and clinician performance by using information technology. *International Journal of Medical Informatics*, 76(2-3), 151-156.
- Boxwala, A. A., Peleg, M., Tu, S. W., Ogunyemi, O., Zeng, Q., Wang, D., Patel, V. L., Greenes, R. A., & Shortliffe, E. H. (2004). GLIF3: A Representation Format for Sharable Computer-Interpretable Clinical Practice Guidelines. *Journal of Biomedical Informatics*, 37(3), 147-161.
- Cabana, M. D., Rand, C. S., Powe, N. R., Wu, A. W., Wilson, M. H., Abboud, P.-A. C., & Rubin, H. R. (1999). Why don't physicians follow clinical practice guidelines? A framework for improvement. *Journal of the American Association*, 282(15), 1458-1466.
- Campana, F., Moreno, A., Riaño, D., & Varga, L. (2008). K4Care: Knowledge-Based Homecare e-Services for an Ageing Europe. In R. Annicchiarico, U. Cortés & C. Urdiales (Eds.), *Agent Technology and e-Health* (pp. 95-115). Basel, Switzerland: Birkhäuser Basel.
- Choi, J., Currie, L. M., Wang, D., & Bakken, S. (2007). Encoding a clinical practice guideline using guideline interchange format: A case study of a depression screening and management guideline. *International Journal of Medical Informatics*, 76S(S2), S302-S307.
- Clercq, P. A. d., Blom, J. A., Korsten, H. H. M., & Hasman, A. (2004). Approaches for creating computer-interpretable guidelines that facilitate decision support. *Artificial Intelligence in Medicine*, 31(1), 1-27.
- Davis, D., Goldman, J., & Palda, V. A. (2007). *Handbook on Clinical Practice Guidelines*. Ottawa, CA: Canadian Medical Association.
- Field, M. J., & Lohr, K. N. (Eds.). (1990). *Clinical Practice Guidelines: Directions for a New Program*. Washington, DC: National Academy Press.
- Fox, J., & Das, S. (2000). *Safe and Sound*. Menlo Park, CA, USA: AAAI and MIT Press.
- Fox, J., Patkar, V., & Thomson, R. (2006). Decision Support for Healthcare: the PROforma evidence base. *Informatics in Primary Care*, 14, 49-54.
- Gennari, J. H., Musen, M. A., Fergerson, R. W., Grosso, W. E., Crubézy, M., Eriksson, H., Noy, N. F., & Tu, S. W. (2003). The Evolution of Protégé: An Environment for Knowledge-Based Systems Development. *International Journal of Human-Computer Studies*, 58(1), 89--123.
- Hrabak, K. M., Campbell, J. R., Tu, S. W., McClure, R., & Weida, R. (2007). Creating Interoperable Guidelines: Requirements of Vocabulary Standards in Immunization Decision Support. In K. A. Kuhn, J. R. Warren & T.-Y. Leong (Eds.), *Proc. of the Proceedings of the 12th World*

- Congress on Health (Medical) Informatics, MEDINFO 2007* (pp. 930-934). Brisbane, AU.
- Isern, D., Sánchez, D., & Moreno, A. (2007). Agents and clinical guidelines: Filling the Semantic Gap. In C. Angulo & L. Godó (Eds.), *Proc. of the Artificial Intelligence Research and Development, CCIA 2007* (pp. 67-76). Andorra la Vella, Andorra.
- Isern, D., & Moreno, A. (2008). Computer-Based Execution of Clinical Guidelines: A Review. *International Journal of Medical Informatics*, 77(12), 787-808.
- Marcos, M., Berger, G., van Harmelen, F., ten Teije, A., Roomans, H., & Miksch, S. (2001). Using critiquing for improving medical protocols: harder than it seems. In S. Quaglini, P. Barahona & S. Andreassen (Eds.), *Proc. of the 8th Conference on Artificial Intelligence in Medicine in Europe, AIME 2001* (pp. 431-441). Cascais, Portugal.
- Marcos, M., Balser, M., ten Teije, A., & van Harmelen, F. (2002). From informal knowledge to formal logic: a realistic case study in medical protocols. In A. Gómez-Pérez & R. Benjamins (Eds.), *Proc. of the 13th International Conference on Knowledge Engineering and Knowledge Management, EKAW 2002* (pp. 49-64). Sigüenza, Spain.
- Maviglia, S. M., Zielstorff, R. D., Paterno, M., Teich, J. M., Bates, D. W., & Kuperman, G. J. (2003). Automating complex guidelines for chronic disease: Lessons learned. *Journal of the American Medical Informatics Association*, 10(2), 154-166.
- McGuinness, D., & van Harmelen, F. (2004). OWL Web Ontology Language, Retrieved August 24, 2008, from <http://www.w3.org/TR/owl-features/>.
- Mersmann, S., & Dojat, M. (2004). SmartCareTM - Automated Clinical Guidelines in Critical Care. In R. L. de Mántaras & L. Saitta (Eds.), *Proc. of the 16th European Conference on Artificial Intelligence, ECAI 2004* (pp. 745-749). València, Spain.
- Miksch, S., Shahar, Y., & Johnson, P. (1997). Asbru: A Task-Specific, Intention-Based, and Time-Oriented Language for Representing Skeletal Plans. In E. Motta, F. V. Harmelen, C. Pierret-Golbreich, I. Filby & N. Wijngaards (Eds.), *Proc. of the 7th Workshop on Knowledge Engineering: Methods & Languages, KEML-97* (pp. 1-20). Milton Keynes, UK.
- Miller, M., & Kearney, N. (2004). Guidelines for clinical practice: development, dissemination and implementation *International Journal of Nursing Studies*, 41(7), 813-821.
- Neches, R., Fickes, R. E., Finin, T., Gruber, T. R., Senator, T., & Swartout, W. R. (1991). Enabling technology for knowledge sharing. *AI Magazine*, 12(3), 36-56.
- Pech-May, F., López-Arévalo, I., & Sosa-Sosa, V. (2009). Validator for Clinical Practice Guidelines. In S. Omatu (Ed.), *Proc. of the 10th International Work-Conference on Artificial Neural Networks, IWANN 2009, Part II* (pp. to appear). Salamanca, Spain.
- Peleg, M., Boxwala, A. A., Bernstam, E., Tu, S., Greenes, R. A., & Shortliffe, E. H. (2001). Sharable Representation of Clinical Guidelines in GLIF: Relationship to the Arden Syntax. *Journal of Biomedical Informatics*, 34(3), 170-181.
- Peleg, M., Tu, S. W., Bury, J., Ciccarese, P., Fox, J., Greenes, R. A., Hall, R., Johnson, P. D., Jones, N., Kumar, A., Miksch, S., Quaglini, S., Seyfang,

- A., Shortliffe, E., & Stefanelli, M. (2003). Comparing Computer-Interpretable Guideline Models: A Case-Study Approach. *Journal of the American Medical Informatics Association*, 10(1), 52-68.
- Peleg, M., Boxwala, A. A., Tu, S., Zeng, Q., Ogunyemi, O., Wang, D., Patel, V. L., Greenes, R. A., & Shortliffe, E. H. (2004). The InterMed approach to sharable computer-interpretable guidelines: motivations and lessons. *Journal of the American Medical Informatics Association*, 11(1), 1-10.
- Priori, S. G., Klein, W., & Bassand, J.-P. (2003). Medical Practice Guidelines: Separating science from economics. *European Heart Journal*, 24(21), 1962-1964.
- Raghupathi, W., & Tan, J. (2002). Strategic IT applications in health care. *Communications of the ACM* 45(12), 56-61.
- Riaño, D. (2004). Ordered Time-Independent CIG Learning. In J. M. Barreiro, F. Martín-Sánchez & V. Maojo (Eds.), *Proc. of the V International Symposium on Biological and Medical Data Analysis, ISBMDA 2004* (pp. 117-128). Barcelona, Catalonia.
- Riaño, D. (2007). The SDA* Model: A Set Theory Approach In P. Kokol, V. Podgorelec, M. Dušanka, M. Zorman & M. Verlic (Eds.), *Proc. of the 20th IEEE International Symposium on Computer-Based Medical Systems, CBMS 2007* (pp. 563-568). Maribor, Slovenia.
- Ricci, S., Celani, M. G., & Righetti, E. (2006). Development of clinical guidelines: methodological and practical issues. *Neurological Sciences*, 27(S3), 228-230.
- Schadow, G., Russler, D. C., & McDonald, C. J. (2001). Conceptual alignment of electronic health record data with guideline and workflow knowledge. *International Journal of Medical Informatics*, 64(2-3), 259-274.
- Seyfang, A., Miksch, S., Votruba, P., Rosenbrand, K., Wittenberg, J., von Croonenborg, J., Reif, W., Balser, M., Schmitt, J., van der Weide, T., Lucas, P., & Homersom, A. (2005). *PROTOCOLURE II Integrating formal methods in the development process of medical guidelines and protocols. D2.2a Specification of formats of intermediate, Asbru to KIV representations* (Deliv. 2.2a). <http://www.protocolure.org/deliverables.php>
- Shahar, Y., Miksch, S., & Johnson, P. (1998). The Asgaard project: a task-specific framework for the application and critiquing of time-oriented clinical guidelines. *Artificial Intelligence in Medicine*, 14(1-2), 29-51.
- Shiffman, R. N., Michel, G., Essaihi, A., & Thornquist, E. (2004). Bridging the Guideline Implementation Gap: A Systematic, Document-Centered Approach to Guideline Implementation. *Journal of the American Medical Informatics Association*, 11(5), 418-427.
- Studer, R., Benjamins, V. R., & Fensel, D. (1998). Knowledge Engineering: Principles and Methods. *IEEE Transactions on Knowledge and Data Engineering*, 25(1-2), 161-197.
- Sutton, D. R., & Fox, J. (2003). The syntax and semantics of the PROforma guideline modeling language. *Journal of the American Medical Informatics Association*, 10(5), 433-443.
- ten Teije, A., Marcos, M., Balser, M., van Croonenborg, J., Duelli, C., van Harmelen, F., Lucas, P., Miksch, S., Reif, W., Rosenbrand, K., & Seyfang, A. (2006). Improving medical protocols by formal methods. *Artificial Intelligence in Medicine*, 36(3), 193-272.

- Wang, D., & Shortliffe, E. H. (2002). GLEE – A Model-Driven Execution System for Computer-Based Implementation of Clinical Practice Guidelines. In AMIA (Ed.), *Proc. of the American Medical Informatics Association Annual Symposium, AMIA 2002* (pp. 855-859). San Antonio, TX, USA.
- Woolf, S. H., Grol, R., Hutchinson, A., Eccles, M., & Grimshaw, J. (1999). Potential benefits, limitations, and harms of clinical guidelines. *British Medical Journal*, 318(7297), 527-530.
- Yu, A. C. (2007). Methods in biomedical ontology. *Journal of Biomedical Informatics*, 39(3), 252-266.
- Zielstorff, R. D. (1998). Online Practice Guidelines. Issues, Obstacles, and Future Prospects *Journal of the American Medical Informatics Association*, 5(3), 227-236.

Appendix A Guideline examples

Below, it shows two examples of clinical guidelines are shown, written in *Proforma* and SDA* which were used in chapter 4 for testing the application built.

A.1 Guideline of hypertension treatment written in *Proforma*

This is a guideline for the treatment of hypertension written in plain text.

```

** PROforma (plain text) version 1.3.37.2 **/
plan :: 'Hypertension' ;
  caption :: "Hypertension";
  component :: 'Current_therapy' ;
    number_of_cycles :: 1;
    ltwh :: 33,118,42,36;
  component :: 'Comorbidities' ;
    number_of_cycles :: 1;
    ltwh :: 203,186,42,36;
  component :: 'Additional_drug_choice' ;
    schedule_constraint ::
completed('Comorbidities') ;
    schedule_constraint ::
completed('Current_therapy') ;
    number_of_cycles :: 1;
    ltwh :: 182,74,36,36;
    component :: 'Target_BP_decision' ;
      autonomous :: yes ;
      schedule_constraint ::
completed('Comorbidities') ;
        number_of_cycles :: 1;
        ltwh :: 358,194,36,36;
        component :: 'Prescribe_additional_drug'
;
      schedule_constraint ::
completed('Additional_drug_choice') ;
      schedule_constraint ::
completed('Target_BP_decision') ;
        number_of_cycles :: 1;
        ltwh :: 478,82,36,36;
        component :: 'Assess_response' ;
          schedule_constraint ::
completed('Prescribe_additional_drug') ;
            number_of_cycles :: -1 ;
            ltwh :: 576,154,48,36;
          end plan.

  enquiry :: 'Comorbidities' ;

  caption :: "Comorbidities";
  source :: 'Age_group' ;
  source :: 'Asthma' ;
  source :: 'COPD' ;
  source :: 'Heart_failure' ;
  source :: 'Isolated_Sys_hypertension' ;
  source :: 'MI' ;
  source :: 'Pregnancy' ;
  source :: 'Proteinuria' ;
  source :: 'STD_Heart_block' ;
    mandatory :: no ;
  source :: 'Systolic_dysfunction' ;
  source :: 'Type_1_Diabetes' ;
    mandatory :: no ;
  end enquiry.

  enquiry :: 'Monitor_BP' ;
    caption :: "Monitor BP";
    source :: 'DBP' ;
    source :: 'SBP' ;
  end enquiry.

  action :: 'Prescribe_additional_drug' ;
    caption :: "Prescribe additional drug";
    procedure :: 'Prescribe additional drug as
chosen.';
  end action.

  enquiry :: 'Current_therapy' ;
    caption :: "Current therapy";
    source :: 'Current_Rx' ;
  end enquiry.

  decision :: 'Additional_drug_choice' ;
    caption :: "Additional drug choice";
    candidate :: 'ACEI' ;

```

```

        argument :: confirming, (
Type_1_Diabetes = Yes and Proteinuria <>
None ) attributes
        argument_name :: ' (
Type_1_Diabetes = Yes and Proteinuria <>
None ) ' ;
        end attributes
        ;
        argument :: confirming, (
Heart_failure = Yes ) attributes
        argument_name :: ' ( Heart_failure
= Yes ) ' ;
        end attributes
        ;
        argument :: confirming, ( MI = Yes
and Systolic_dysfunction = Yes ) attributes
        argument_name :: ' ( MI = Yes
and Systolic_dysfunction = Yes ) ' ;
        end attributes
        ;
        argument :: excluding, ( Pregnancy =
Yes ) attributes
        argument_name :: ' ( Pregnancy =
Yes ) ' ;
        end attributes
        ;
        argument :: excluding, ( Current_Rx
= K_sparing_diuretic ) attributes
        argument_name :: ' ( Current_Rx
= K_sparing_diuretic ) ' ;
        end attributes
        ;
        argument :: excluding, ( Current_Rx
= ACEI ) attributes
        argument_name :: ' ( Current_Rx
= ACEI ) ' ;
        end attributes
        ;
        recommendation :: Netsupport(
Additional_drug_choice, ACEI ) >= 1 ;
        candidate :: 'Loop_diuretic' ;
        argument :: confirming, (
Heart_failure = Yes ) attributes
        argument_name :: ' ( Heart_failure
= Yes ) ' ;
        end attributes
        ;
        argument :: confirming, (
Isolated_Sys_hypertension = Yes and
Age_group = Older ) attributes
        argument_name :: ' (
Isolated_Sys_hypertension = Yes and
Age_group = Older ) ' ;
        end attributes
        ;
        argument :: excluding, ( Current_Rx
= Thiazide_diuretic or Current_Rx =
K_sparing_diuretic or Current_Rx =
Loop_diuretic ) attributes

```

```

        argument_name :: ' ( Current_Rx
= Thiazide_diuretic or Current_Rx =
K_sparing_diuretic or Current_Rx =
Loop_diuretic ) ' ;
        end attributes
        ;
        recommendation :: Netsupport(
Additional_drug_choice, Loop_diuretic ) >=
1 ;
        candidate :: 'Ca_channel_dihydro_short'
;
        argument :: for, ( Type_1_Diabetes =
Yes and Proteinuria <> None ) attributes
        argument_name :: ' (
Type_1_Diabetes = Yes and Proteinuria <>
None ) ' ;
        end attributes
        ;
        argument :: excluding, ( Current_Rx
= Ca_blocker_non_DHP ) attributes
        argument_name :: ' ( Current_Rx
= Ca_blocker_non_DHP ) ' ;
        end attributes
        ;
        argument :: for, ( Current_Rx =
Beta_blocker_ISA or Current_Rx =
Beta_blocker_non_ISA ) attributes
        argument_name :: ' ( Current_Rx
= Beta_blocker_ISA or Current_Rx =
Beta_blocker_non_ISA ) ' ;
        end attributes
        ;
        argument :: confirming, (
Isolated_Sys_hypertension = Yes and
Age_group = Older ) attributes
        argument_name :: ' (
Isolated_Sys_hypertension = Yes and
Age_group = Older ) ' ;
        end attributes
        ;
        argument :: excluding, ( Current_Rx
= Ca_blocker_DHP_long or Current_Rx =
Ca_blocker_DHP_short or Current_Rx =
Ca_blocker_non_DHP ) attributes
        argument_name :: ' ( Current_Rx
= Ca_blocker_DHP_long or Current_Rx =
Ca_blocker_DHP_short or Current_Rx =
Ca_blocker_non_DHP ) ' ;
        end attributes
        ;
        recommendation :: Netsupport(
Additional_drug_choice,
Ca_channel_dihydro_short ) >= 1 ;
        candidate :: 'Ca_channel_dihydro_long' ;
        argument :: for, ( Type_1_Diabetes =
Yes and Proteinuria <> None ) attributes
        argument_name :: ' (
Type_1_Diabetes = Yes and Proteinuria <>
None ) ' ;
        end attributes

```

```

;
argument :: excluding, ( Current_Rx
= Ca_blocker_non_DHP ) attributes
argument_name :: ' ( Current_Rx
= Ca_blocker_non_DHP ) ' ;
end attributes
;
argument :: for, ( Current_Rx =
Beta_blocker_ISA or Current_Rx =
Beta_blocker_non_ISA ) attributes
argument_name :: ' ( Current_Rx
= Beta_blocker_ISA or Current_Rx =
Beta_blocker_non_ISA ) ' ;
end attributes
;
argument :: confirming, (
Isolated_Sys_hypertension = Yes and
Age_group = Older ) attributes
argument_name :: ' (
Isolated_Sys_hypertension = Yes and
Age_group = Older ) ' ;
end attributes
;
argument :: excluding, ( Current_Rx
= Ca_blocker_DHP_long or Current_Rx =
Ca_blocker_DHP_short or Current_Rx =
Ca_blocker_non_DHP ) attributes
argument_name :: ' ( Current_Rx
= Ca_blocker_DHP_long or Current_Rx =
Ca_blocker_DHP_short or Current_Rx =
Ca_blocker_non_DHP ) ' ;
end attributes
;
recommendation :: Netsupport(
Additional_drug_choice,
Ca_channel_dihydro_long ) >= 1 ;
candidate :: 'K_sparing_diuretic' ;
argument :: confirming, (
Isolated_Sys_hypertension = Yes and
Age_group = Older ) attributes
argument_name :: ' (
Isolated_Sys_hypertension = Yes and
Age_group = Older ) ' ;
end attributes
;
argument :: excluding, ( Current_Rx
= Thiazide_diuretic or Current_Rx =
K_sparing_diuretic or Current_Rx =
Loop_diuretic ) attributes
argument_name :: ' ( Current_Rx
= Thiazide_diuretic or Current_Rx =
K_sparing_diuretic or Current_Rx =
Loop_diuretic ) ' ;
end attributes
;
argument :: excluding, ( Current_Rx
= ACEI ) attributes
argument_name :: ' ( Current_Rx
= ACEI ) ' ;
end attributes

```

```

;
recommendation :: Netsupport(
Additional_drug_choice, K_sparing_diuretic
) >= 1 ;
candidate :: 'Non_ISA_beta_blocker' ;
argument :: excluding, (
STD_Heart_block = Yes ) attributes
argument_name :: ' (
STD_Heart_block = Yes ) ' ;
end attributes
;
argument :: against, (
Type_1_Diabetes = Yes and Proteinuria <>
None ) attributes
argument_name :: ' (
Type_1_Diabetes = Yes and Proteinuria <>
None ) ' ;
end attributes
;
argument :: confirming, ( MI = Yes )
attributes
argument_name :: ' ( MI = Yes ) ' ;
end attributes
;
argument :: excluding, ( Current_Rx
= Ca_blocker_non_DHP ) attributes
argument_name :: ' ( Current_Rx
= Ca_blocker_non_DHP ) ' ;
end attributes
;
argument :: excluding, ( Current_Rx
= Beta_blocker_non_ISA or Current_Rx =
Beta_blocker_ISA ) attributes
argument_name :: ' ( Current_Rx
= Beta_blocker_non_ISA or Current_Rx =
Beta_blocker_ISA ) ' ;
end attributes
;
argument :: for, ( Current_Rx =
Ca_blocker_DHP_long or Current_Rx =
Ca_blocker_DHP_short ) attributes
argument_name :: ' ( Current_Rx
= Ca_blocker_DHP_long or Current_Rx =
Ca_blocker_DHP_short ) ' ;
end attributes
;
argument :: excluding, ( Asthma =
Yes or COPD = Yes ) attributes
argument_name :: ' ( Asthma =
Yes or COPD = Yes ) ' ;
end attributes
;
argument :: for, ( Current_Rx =
Thiazide_diuretic ) attributes
argument_name :: ' ( Current_Rx
= Thiazide_diuretic ) ' ;
end attributes
;

```

```

        recommendation :: Netsupport(
Additional_drug_choice,
Non_ISA_beta_blocker ) >= 1 ;
        candidate :: 'ISA_beta_blocker' ;
        argument :: excluding, (
STD_Heart_block = Yes ) attributes
        argument_name :: ' (
STD_Heart_block = Yes ) ' ;
        end attributes
        ;
        argument :: against, (
Type_1_Diabetes = Yes and Proteinuria <>
None ) attributes
        argument_name :: ' (
Type_1_Diabetes = Yes and Proteinuria <>
None ) ' ;
        end attributes
        ;
        argument :: excluding, ( Current_Rx
= Ca_blocker_non_DHP ) attributes
        argument_name :: ' ( Current_Rx
= Ca_blocker_non_DHP ) ' ;
        end attributes
        ;
        argument :: excluding, ( Current_Rx
= Beta_blocker_ISA or Current_Rx =
Beta_blocker_non_ISA ) attributes
        argument_name :: ' ( Current_Rx
= Beta_blocker_ISA or Current_Rx =
Beta_blocker_non_ISA ) ' ;
        end attributes
        ;
        argument :: for, ( Current_Rx =
Ca_blocker_DHP_long or Current_Rx =
Ca_blocker_DHP_short ) attributes
        argument_name :: ' ( Current_Rx
= Ca_blocker_DHP_long or Current_Rx =
Ca_blocker_DHP_short ) ' ;
        end attributes
        ;
        argument :: excluding, ( Asthma =
Yes or COPD = Yes ) attributes
        argument_name :: ' ( Asthma =
Yes or COPD = Yes ) ' ;
        end attributes
        ;
        argument :: for, ( Current_Rx =
Thiazide_diuretic ) attributes
        argument_name :: ' ( Current_Rx
= Thiazide_diuretic ) ' ;
        end attributes
        ;
        recommendation :: Netsupport(
Additional_drug_choice, ISA_beta_blocker )
>= 1 ;
        candidate :: 'Thiazide_diuretic' ;
        argument :: for, ( Current_Rx =
Beta_blocker_ISA or Current_Rx =
Beta_blocker_non_ISA ) attributes

```

```

        argument_name :: ' ( Current_Rx
= Beta_blocker_ISA or Current_Rx =
Beta_blocker_non_ISA ) ' ;
        end attributes
        ;
        argument :: confirming, (
Isolated_Sys_hypertension = Yes and
Age_group = Older ) attributes
        argument_name :: ' (
Isolated_Sys_hypertension = Yes and
Age_group = Older ) ' ;
        end attributes
        ;
        argument :: excluding, ( Current_Rx
= Thiazide_diuretic or Current_Rx =
K_sparing_diuretic or Current_Rx =
Loop_diuretic ) attributes
        argument_name :: ' ( Current_Rx
= Thiazide_diuretic or Current_Rx =
K_sparing_diuretic or Current_Rx =
Loop_diuretic ) ' ;
        end attributes
        ;
        recommendation :: Netsupport(
Additional_drug_choice, Thiazide_diuretic )
>= 1 ;
        candidate :: 'Ca_channel_non_dihydro' ;
        argument :: excluding, (
STD_Heart_block = Yes ) attributes
        argument_name :: ' (
STD_Heart_block = Yes ) ' ;
        end attributes
        ;
        argument :: for, ( Type_1_Diabetes =
Yes and Proteinuria <> None ) attributes
        argument_name :: ' (
Type_1_Diabetes = Yes and Proteinuria <>
None ) ' ;
        end attributes
        ;
        argument :: excluding, ( Current_Rx
= Beta_blocker_ISA or Current_Rx =
Beta_blocker_non_ISA ) attributes
        argument_name :: ' ( Current_Rx
= Beta_blocker_ISA or Current_Rx =
Beta_blocker_non_ISA ) ' ;
        end attributes
        ;
        argument :: excluding, ( Current_Rx
= Ca_blocker_DHP_long or Current_Rx =
Ca_blocker_DHP_short ) attributes
        argument_name :: ' ( Current_Rx
= Ca_blocker_DHP_long or Current_Rx =
Ca_blocker_DHP_short ) ' ;
        end attributes
        ;
        argument :: excluding, ( Current_Rx
= Ca_blocker_DHP_long or Current_Rx =
Ca_blocker_DHP_short or Current_Rx =
Ca_blocker_non_DHP ) attributes

```

```

        argument_name :: ' ( Current_Rx
= Ca_blocker_DHP_long or Current_Rx =
Ca_blocker_DHP_short or Current_Rx =
Ca_blocker_non_DHP ) ' ;
        end attributes
        ;
        recommendation :: Netsupport(
Additional_drug_choice,
Ca_channel_non_dihydro ) >= 1 ;
        end decision.

action :: 'Await_adequate_response' ;
        caption :: "Await adequate response";
        procedure :: 'Continue therapy until
target BPs reached';
        end action.

plan :: 'Assess_response' ;
        caption :: "Assess response";
        goal :: "DBP < targetDBP and SBP <
targetSBP";
        component :: 'Monitor_BP' ;
                number_of_cycles :: 1;
                ltwh :: 33,118,42,36;
        component ::
'Await_adequate_response' ;
                number_of_cycles :: 1;
                ltwh :: 88,118,36,36;
        end plan.

decision :: 'Target_BP_decision' ;
        caption :: "Target BP decision";
        postcondition :: TargetSBP = if(
result_of( Target_BP_decision) = Standard,
140, if( result_of( Target_BP_decision) =
Medium, 130, 125)) and TargetDBP = if(
result_of( Target_BP_decision) = Standard,
90, if( result_of( Target_BP_decision) =
Medium, 85, 75)) ;
        candidate :: 'Medium' ;
        argument :: for, ( Type_1_Diabetes =
Yes or Proteinuria = '<1 g / 24 hrs' )
attributes
        argument_name :: ' (
Type_1_Diabetes = Yes or Proteinuria =
'\<1 g / 24 hrs\' ) ' ;
        end attributes
        ;
        recommendation :: Netsupport(
Target_BP_decision, Medium ) >= 1 ;
        candidate :: 'Low' ;
        argument :: confirming, ( Proteinuria
= '>1 g / 24 hrs' ) attributes
        argument_name :: ' ( Proteinuria =
'\>1 g / 24 hrs\' ) ' ;
        end attributes
        ;
        recommendation :: Netsupport(
Target_BP_decision, Low ) >= 1 ;
        candidate :: 'Standard' ;

```

```

        argument :: for, ( Proteinuria = None
and Type_1_Diabetes = No ) attributes
        argument_name :: ' ( Proteinuria =
None and Type_1_Diabetes = No ) ' ;
        end attributes
        ;
        recommendation :: Netsupport(
Target_BP_decision, Standard ) >= 1 ;
        end decision.

data :: 'Age_group' ;
        type :: text ;
        caption :: "Patients age group";
        range :: "Younger","Older";
        end data.

data :: 'Asthma' ;
        type :: boolean ;
        caption :: "Does the patient have
asthma?";
        true_value :: 'Yes' ;
        false_value :: 'No' ;
        end data.

data :: 'COPD' ;
        type :: boolean ;
        caption :: "Does that patient have
COPD?";
        true_value :: 'Yes' ;
        false_value :: 'No' ;
        end data.

data :: 'Current_Rx' ;
        type :: text ;
        caption :: "Patients current therapy";
        range
:: "ACEI","Beta_blocker_ISA","Beta_blocker
_non_ISA","Ca_blocker_DHP_long","Ca_bl
ocker_DHP_short","Ca_blocker_non_DHP"
,"Thiazide_diuretic","Loop_diuretic","K_spar
ing_diuretic";
        end data.

data :: 'DBP' ;
        type :: integer ;
        caption :: "Diastolic BP";
        end data.

data :: 'Heart_failure' ;
        type :: boolean ;
        caption :: "Does the patient have heart
failure?";
        true_value :: 'Yes' ;
        false_value :: 'No' ;
        end data.

data :: 'Isolated_Sys_hypertension' ;
        type :: boolean ;
        caption :: "Isolated systolic
hypertension?";

```

```

    true_value :: 'Yes' ;
    false_value :: 'No' ;
end data.

```

```

data :: 'MI' ;
type :: boolean ;
caption :: "Previous MI?";
true_value :: 'Yes' ;
false_value :: 'No' ;
end data.

```

```

data :: 'Pregnancy' ;
type :: boolean ;
caption :: "Is the patient pregnant?";
true_value :: 'Yes' ;
false_value :: 'No' ;
end data.

```

```

data :: 'Proteinuria' ;
type :: text ;
caption :: "Proteinuria?";
range :: "None", "<1 g / 24 hrs", ">1 g /
24 hrs";
end data.

```

```

data :: 'SBP' ;

```

```

type :: integer ;
caption :: "Systolic BP";
end data.

```

```

data :: 'STD_Heart_block' ;
type :: boolean ;
caption :: "Second or third degree heart
block";
true_value :: 'Yes' ;
false_value :: 'No' ;
end data.

```

```

data :: 'Systolic_dysfunction' ;
type :: boolean ;
caption :: "Systolic dysfunction";
true_value :: 'Yes' ;
false_value :: 'No' ;
end data.

```

```

data :: 'Type_1_Diabetes' ;
type :: boolean ;
caption :: "Type 1 diabetes?";
true_value :: 'Yes' ;
false_value :: 'No' ;
end data.

```

A.2 Home Care Patient Discharge guideline written in SDA*

This is a guideline to decide the patient's discharge from the Home Care Service. It is written in XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<sda_procedure
xmlns:xsi="http://www.w3.org/2001/XMLSchema
hema-instance"
xsi:noNamespaceSchemaLocation="SdaSc
hema.xsd">
<sda_state id="0">
<sda_term name="START"/>
<next>
<element>3</element>
</next>
</sda_state>
<sda_state id="1">
<sda_term name="END"/>
</sda_state>
<sda_decision id="3" question="Why is the
HCRrequest rejected?">
<sda_branch>
<sda_term name="Outcomes Reached"/>
<sda_term name="change of address"/>
<sda_term name="conventional end"/>
<sda_term name="end of treatment"/>
<sda_term name="refusal of treatment"/>
<sda_connector>
<element>5</element>
</sda_connector>
</sda_branch>
<otherwise>
<element>6</element>
</otherwise>
<sda_branch>
<sda_term name="Other reasons"/>
<sda_term name="admission to Hospital"/>
<sda_connector>
<element>4</element>
</sda_connector>
</sda_branch>
</sda_decision>
<sda_actionblock id="5">
<sda_action name="00.10&#9;Patient
Discharge" type="[ANY]">
  <sda_document id="iid1"
newInstance="true"/>
<petitioner>[HN]</petitioner>
```

```
<petitioner>[PC]</petitioner>
<performer>[HN]</performer>
<performer>[PC]</performer>
</sda_action>
<next>
<element>1</element>
</next>
</sda_actionblock>
<sda_state id="2">
<sda_term name="END"/>
</sda_state>
<sda_state id="4">
<sda_term name="SUSPENSION"/>
<next>
<element>7</element>
</next>
</sda_state>
<sda_state id="8">
<sda_term name="END"/>
</sda_state>
<sda_actionblock id="6">
<sda_action name="EU.2  define
intervention plan" type="[ANY]">
<petitioner>[EU]</petitioner>
<performer>[EU]</performer>
</sda_action>
<next>
<element>2</element>
</next>
</sda_actionblock>
<sda_decision id="7">
<otherwise>
<element>8</element>
</otherwise>
<sda_branch>
<sda_term name="Discharge from
Hospital" question="Should the patient be
discharged from Hospital?"/>
<sda_connector>
<element>0</element>
</sda_connector>
</sda_branch>
</sda_decision>
</sda_procedure>
```